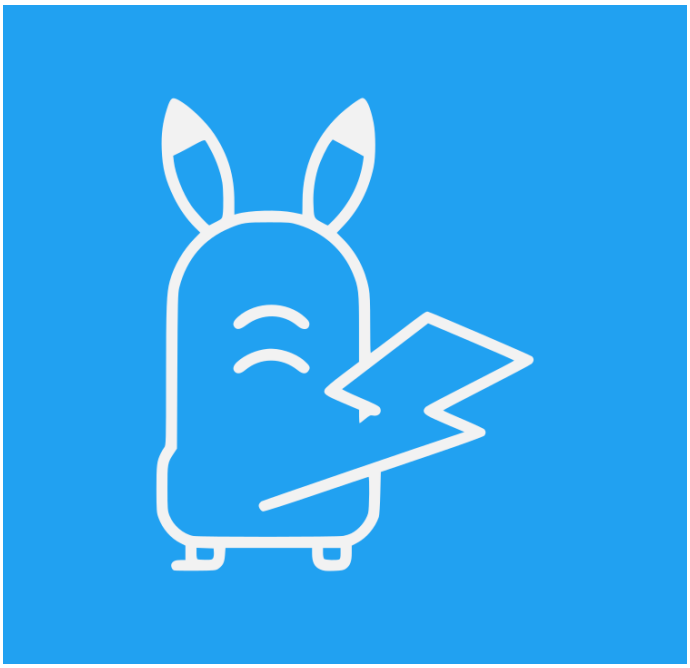
Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e Multimédia (LEIM)

DAM – 21/22

Projecto Final – Relatório Final



Professores: Eng.º Pedro Fazenda e Eng.ªAna Correia

Trabalho realizado por:	número
Miguel Ginga	46292

Lisboa, 20 de Junho de 2022

Índice de matérias

Índice de Imagens

1. Apresentação da App

A aplicação desenvolvida é designada por *Trainer Companion App* e é uma compilação de ferramentas para todos os fãs do universo Pokémon.

A *App* é composta por 4 componentes principais:

- *Team Builder* – onde o utilizador pode definir uma equipa de 6 *pokémon* que pode ser alterada a qualquer altura.
- *Pokédex* – semelhante ao desenvolvido no tutorial 4 da unidade curricular, mas neste caso não fará uso de uma base de dados, mas sim do acesso a uma API, podendo ainda ser filtrado por gerações.
- *Hall of Fame* – quando se acaba um jogo de Pokémon a equipa com a qual se finaliza o jogo fica registada no *Hall of Fame*, aqui o utilizador tem a possibilidade de guardar todas as suas equipas vencedoras, associando-as por jogo.
- *Shiny Tracker* – algo popular na série de Pokémon é uma prática denominada por *Shiny Hunting* que consiste em apanhar formas alternativas de um Pokémon extremamente raras. Esta funcionalidade permite ao utilizador gravar quais as formas *Shiny* que já obteve, podendo filtrar por formas obtidas ou pesquisando o nome de um Pokémon específico para visualizar a sua forma *shiny*.

A *App* terá como publico alvo qualquer pessoa que seja fã do universo Pokémon, casualmente ou de forma mais intensiva, tendo funções que serão uteis para qualquer tipo de utilizador.

Como funcionalidade premium teremos o *Shiny Tracker*, ou seja, será uma funcionalidade pelos quais os utilizadores serão incentivados a pagar para ter acesso.

2.Processo de Desenvolvimento

O processo de desenvolvimento foi iniciado por decidir quais serão as ferramentas que seriam uteis e de interesse ao Utilizador tendo em conta qual a finalidade da aplicação.

As ferramentas apresentadas e desenvolvidas foram determinadas como as mais interessantes e uteis para qualquer tipo de utilizador. Após decididas quais as ferramentas a implementar foi necessário visualizar como estas seriam apresentadas ao utilizador, entrando na fase de desenvolvimento do *Wire-Frame*. Este processo tem como objectivo, para o programador visualizar o conteúdo a ser implementado, não dando importância ao aspecto visual do mesmo.

2.1 Wire-Frame

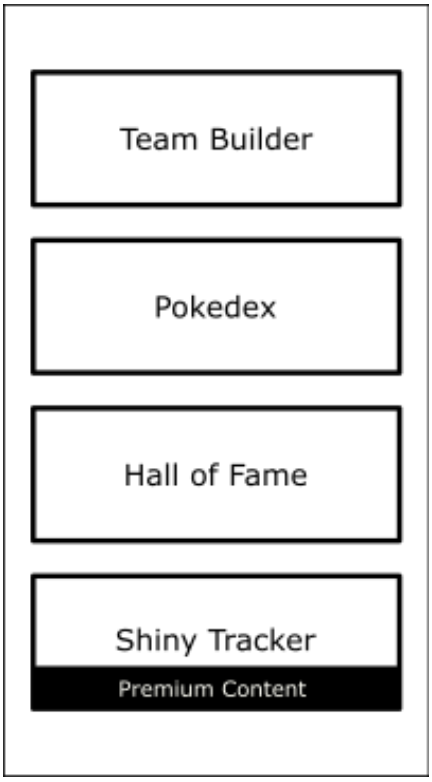


Imagem 1 – Main Activity

A actividade principal deve apresentar quais todas as ferramentas disponíveis para o utilizador, de forma directa e simples evitando obstáculos ao utilizador para quando ele quiser aceder a uma ferramenta especifica.

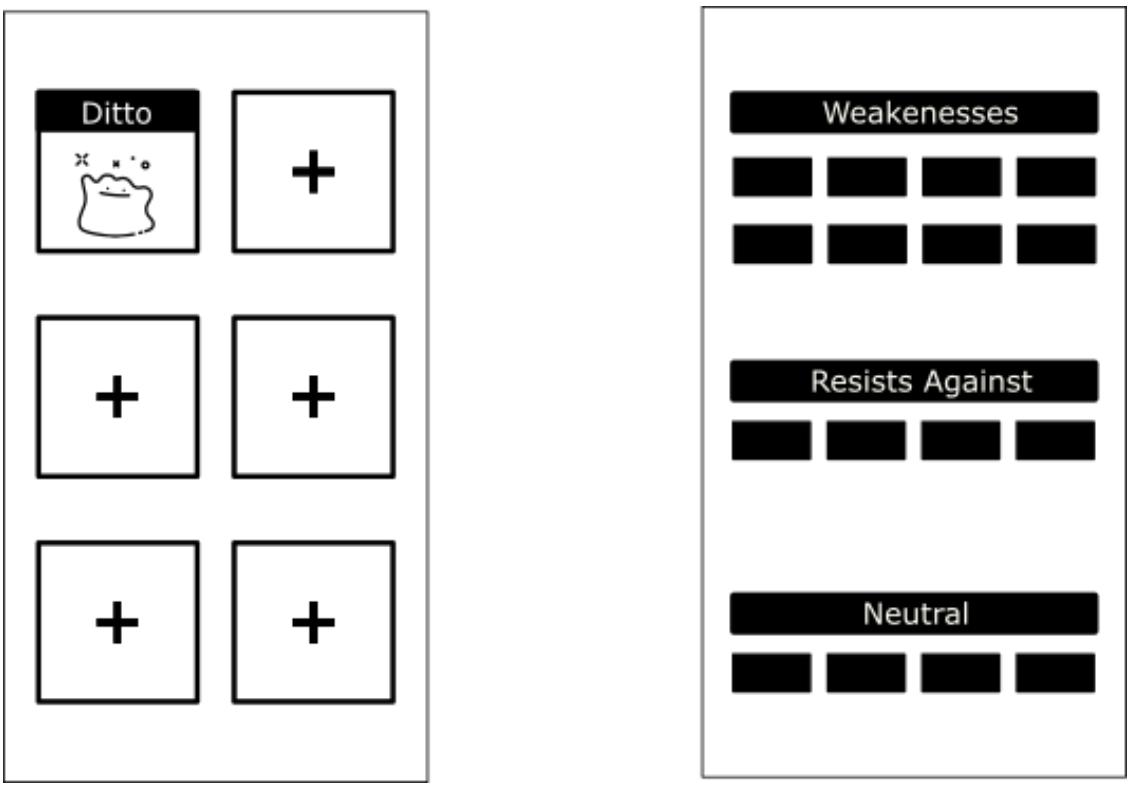


Imagem 2 – TeamBuilder Activity

A actividade de Team Builder como tem o objectivo de representar uma equipa completa, sendo esta composta por 6 elementos, terá 6 slots com a mesma função de pesquisar e adicionar um Pokémon.

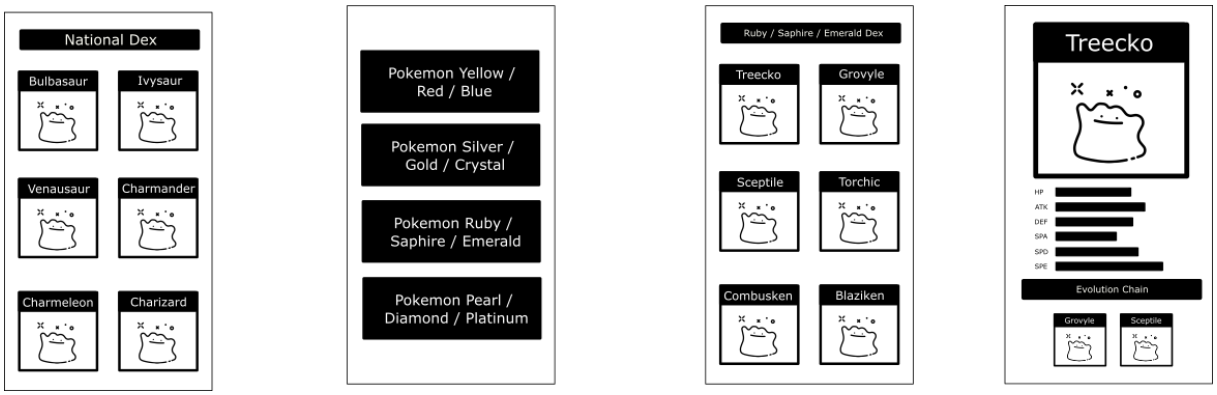


Imagem 3 – Pokedex Activity

A actividade pokédex será composta por uma lista de todos os Pokémon, sendo inicialmente apresentado o Pokédex Nacional, que será o pokédex completo, havendo a possibilidade de este ser filtrado por gerações. O Pokédex por geração será semelhante ao pokédex nacional mas apenas irá conter os Pokémon da geração associada. Ao carregar sobre um dos Pokémon é aberta a actividade que irá mostra os dados do mesmo, sendo estes restringidos aos mais relevantes, que serão os tipos e estatísticas.

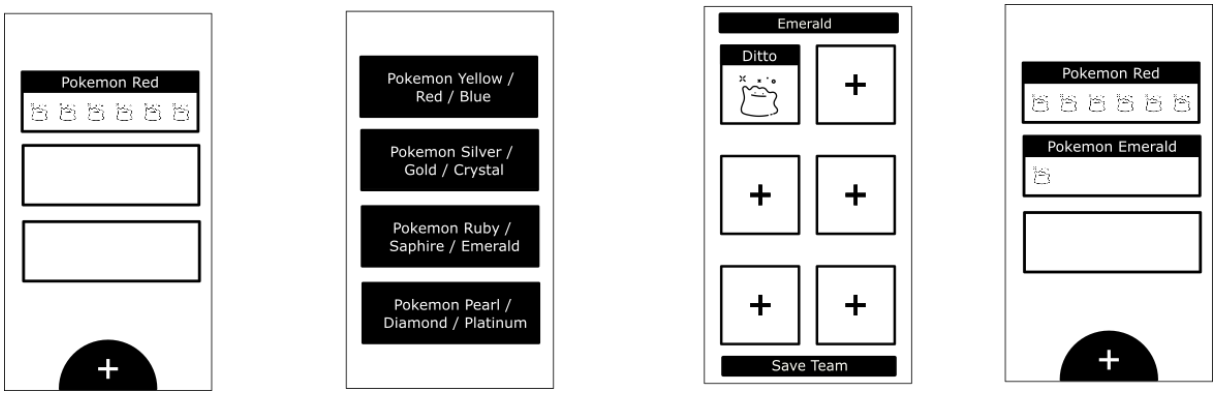


Imagem 4 – Hall of Fame Activity

O Hall of Fame por norma irá ser inicializado em vazio, dando relevância ao botão para se adicionar uma equipa. Ao carregar neste botão navegamos para uma actividade que apresnta uma lista com todos os jogos existentes. Depois de seleccionado um jogo podemos adicionar a nossa equipa, carregando no botão de salvar quando estivermos concluídos, caso contrário a equipa não será gravada. A qualquer momento o utilizador pode seleccionar uma das equipas e fazer alterações.

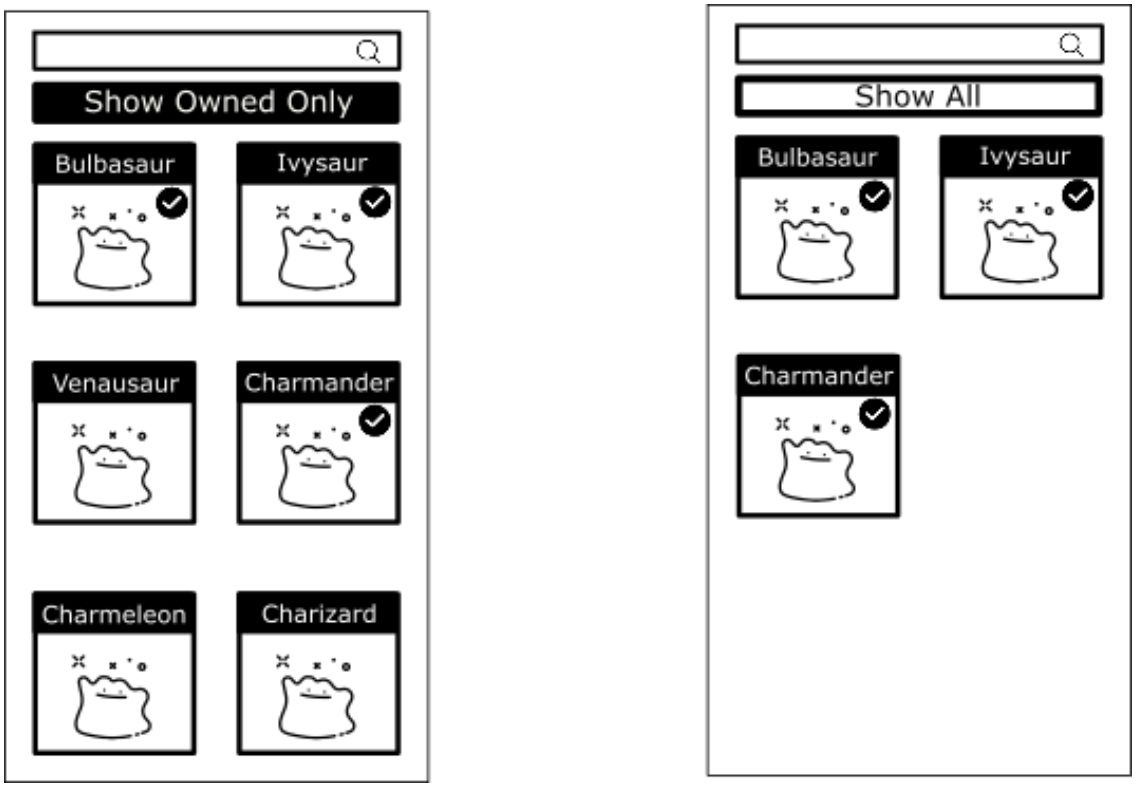


Imagem 5 – Shiny Hunting Activity

A base da actividade shiny huting será semelhante ao Pokédex mas neste caso teremos de dar relevância a se um Pokémon está seleccionada ou não. Haverá ainda um botão para filtrar a lista por apanhados ou então pesquisar mesmo um Pokémon específico.

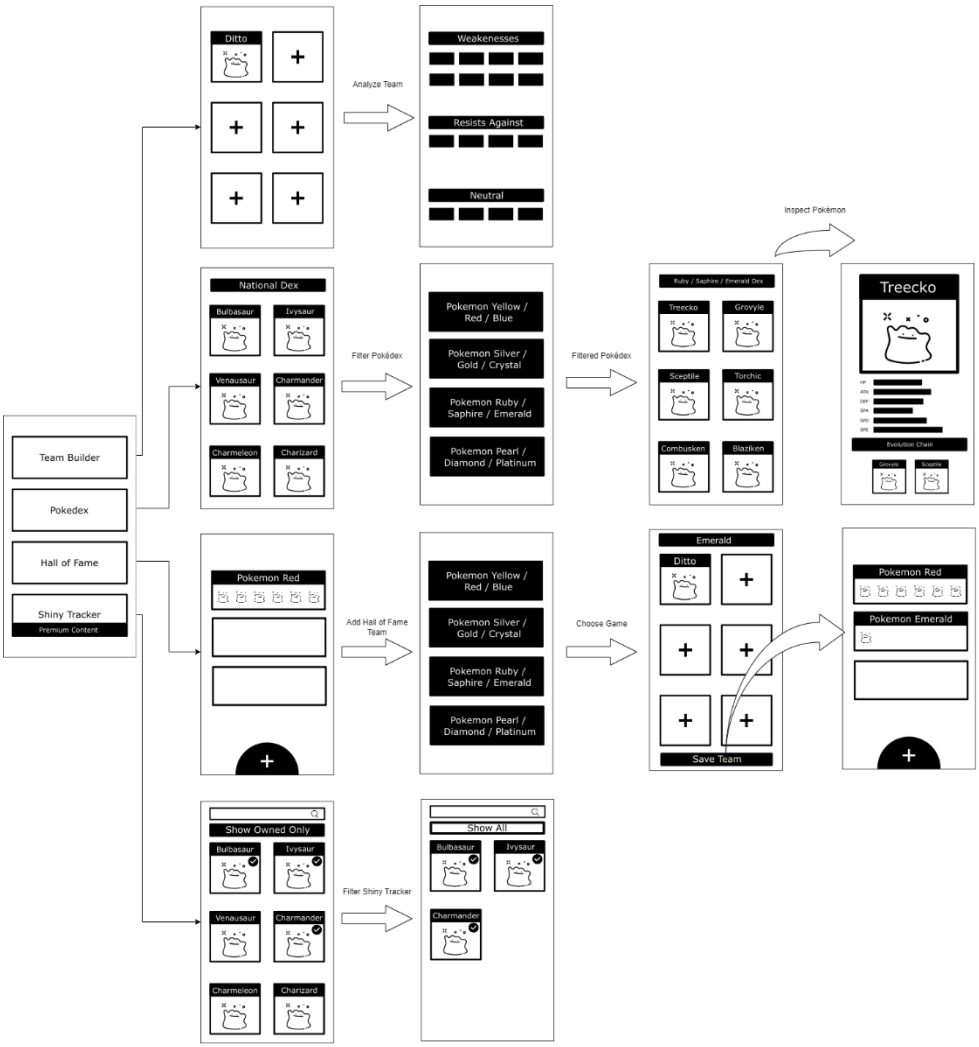


Imagem 6 – Wire Frame Navigation

2.2 MockUps

Os Mockups devem representar como será a aplicação apresentada após o seu desenvolvimento, adicionando então ao trabalho feito no WireFrame, conteúdos visuais.

O primeiro passo foi definir quais as cores principais do projecto a serem utilizadas. Isto torna o design mais fácil de executar e coerente à medida que vai ser desenvolvido. As cores definidas foram conseguem atribuir um aspecto relevante à aplicação sem serem demasiado intensas para o utilizador. Numa fase inicial foi equacionada uma cor variante de laranja mas esta mostrava-se demasiado intensa e difícil de equilibrar com outra cor como cor de detalhe.

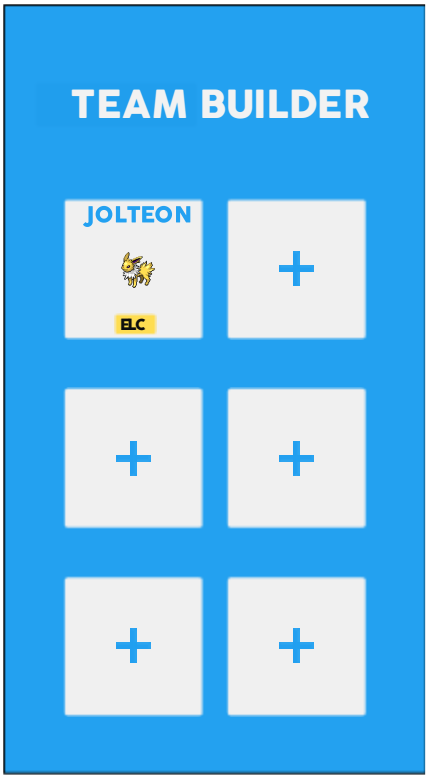


Imagem 7 – MockUp TeamBuilder

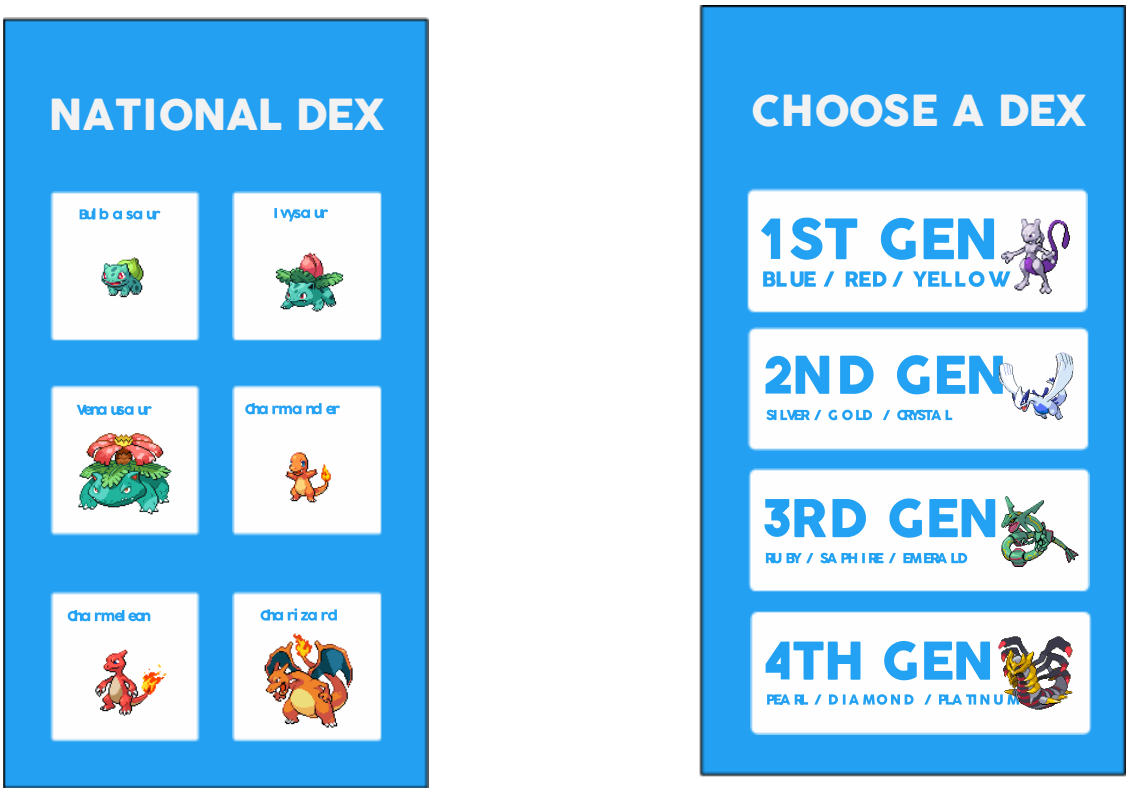


Imagem 8 – Mockup PokéDex e FilterPokédex

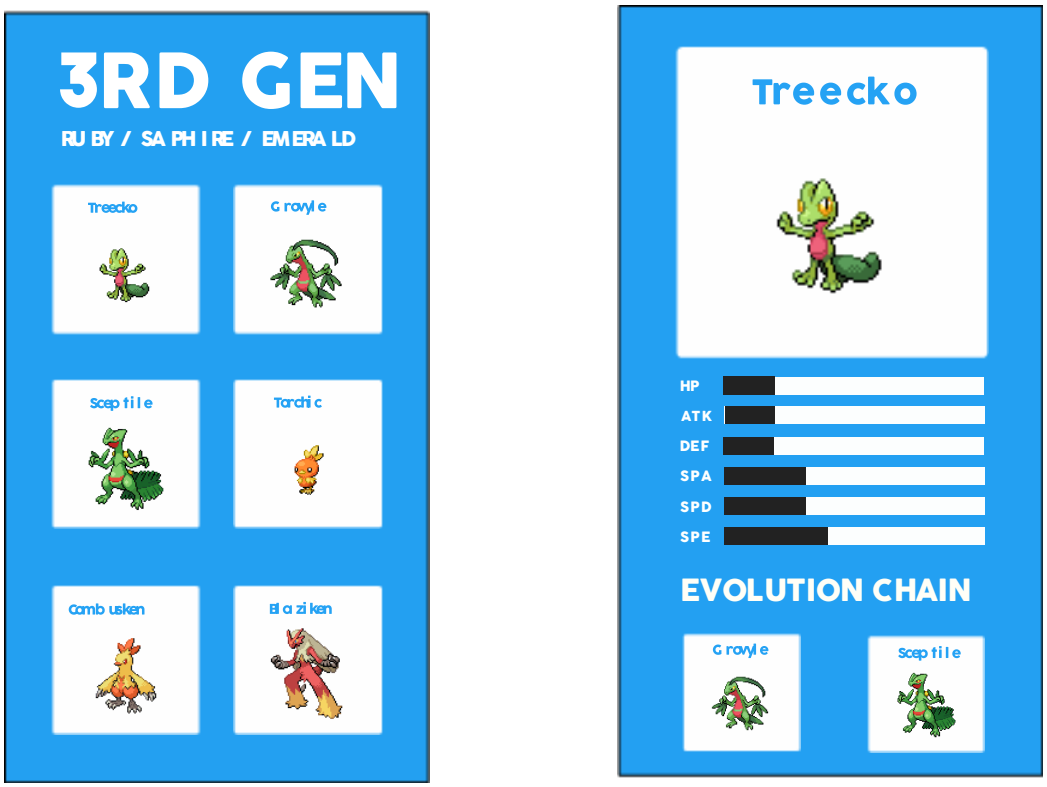


Imagem 9 – Mockup PokéDex filtrado e PokédexSelection

Imagem 10 – Mockup Actividade Hall of Fame e Selecção de Jogo

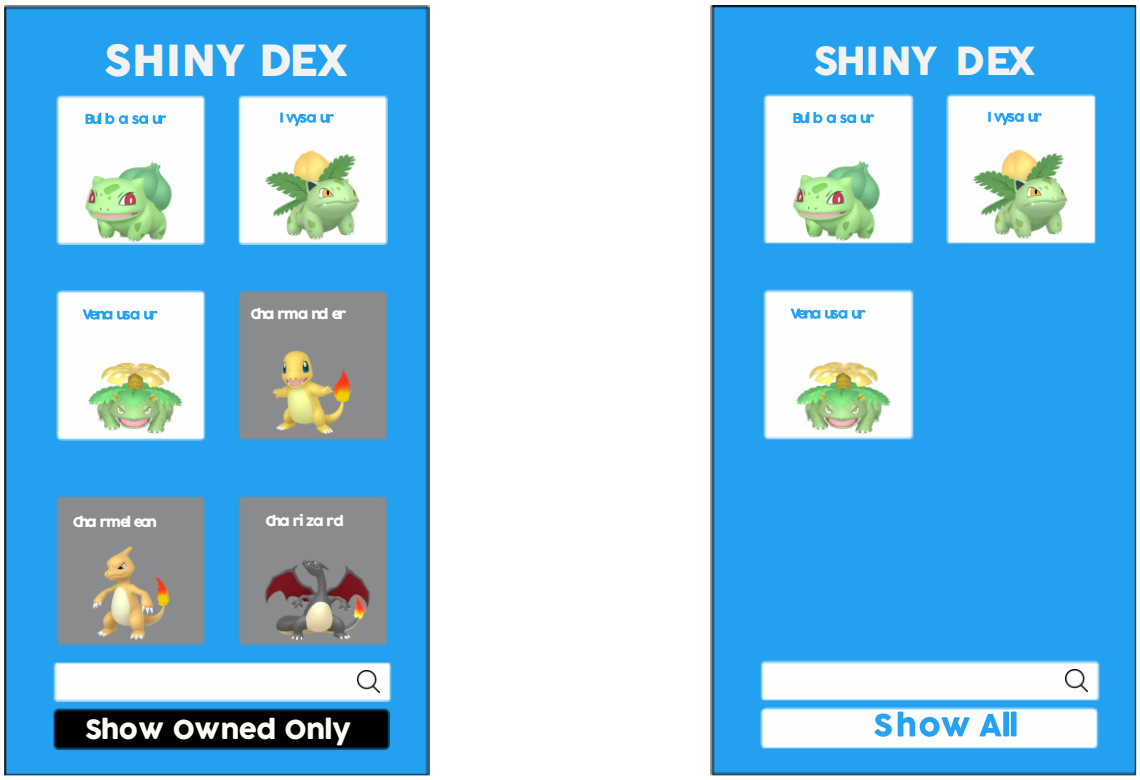


Imagem 11 – Mockup Actividade ShinyDex

2.3 Screenshots Finais da Aplicação

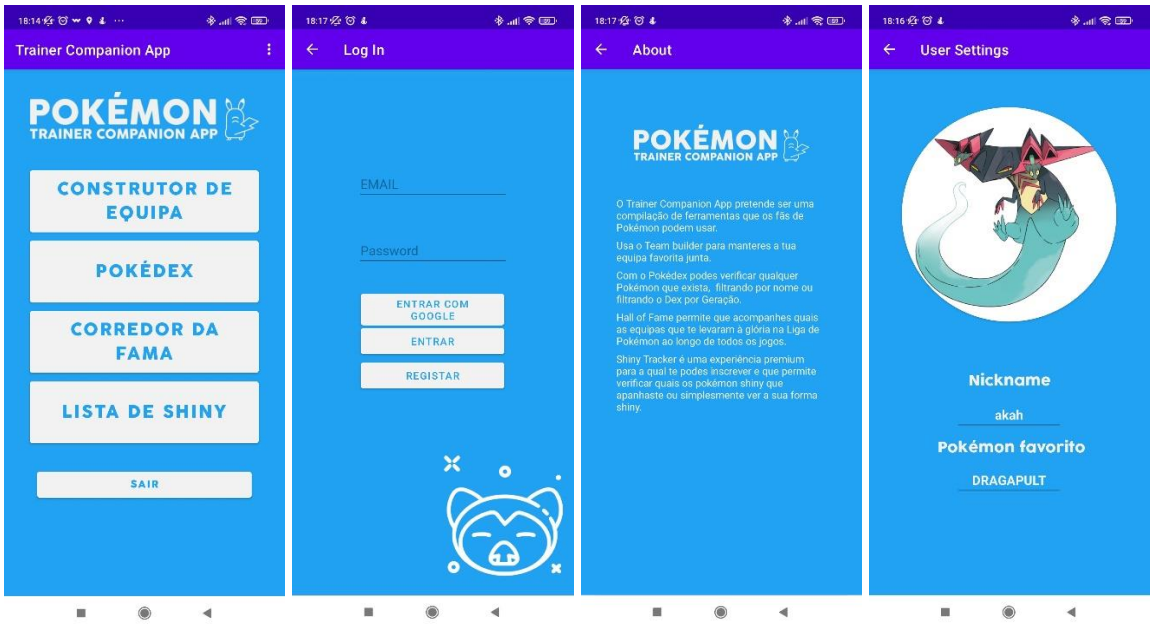


Imagem 12 – Actividade principal, Log In e Sobre a App, Definições de Utilizador

Imagem 13 – Actividade TeamBuilder, Pokédex e DexFilter

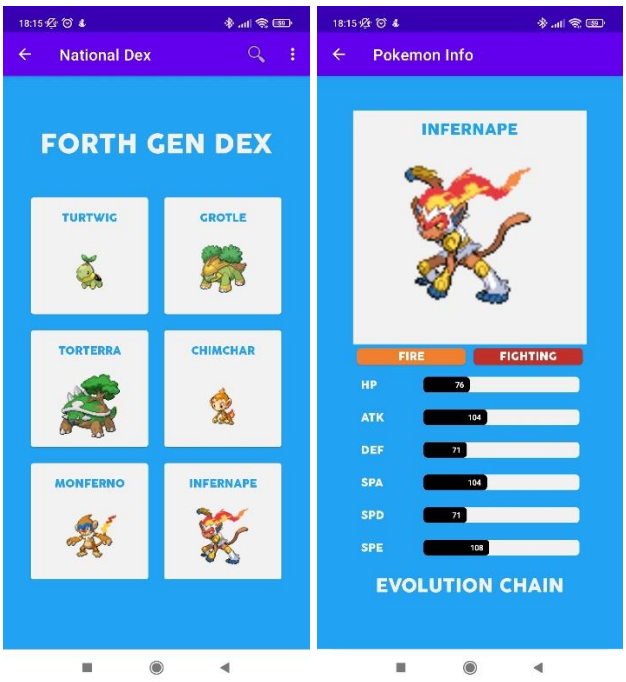


Imagem 14 – Pokédex Filtrado, Selecção de Pokédex

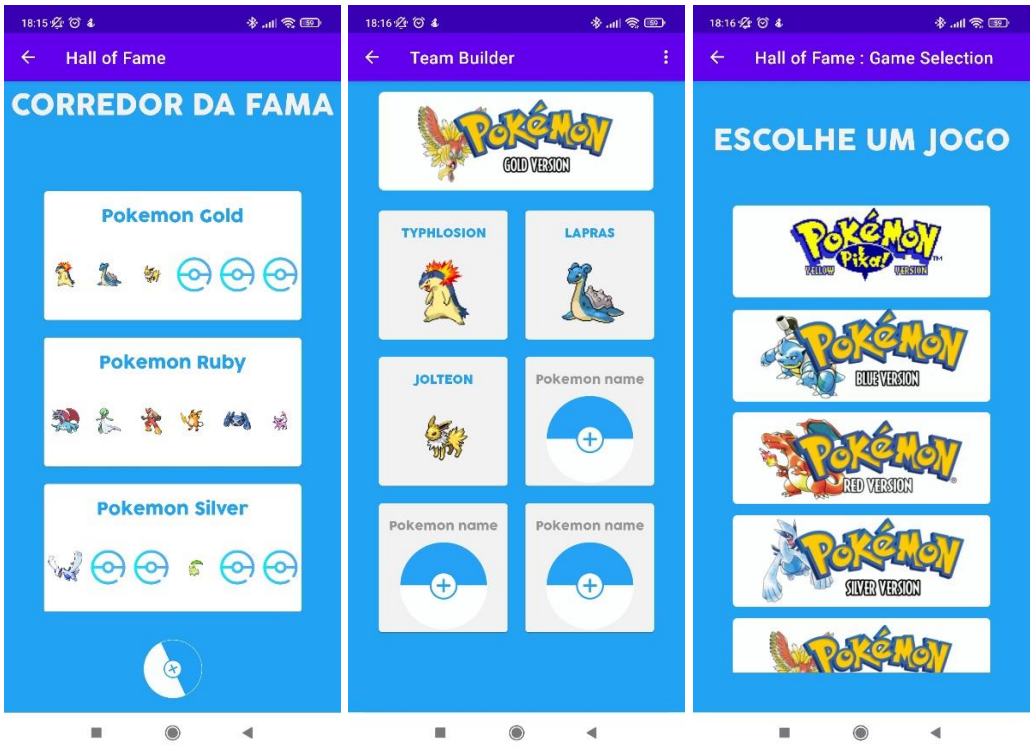


Imagem 15 – Acitividade Hall of Fame, HofTeam, HofGameSelection

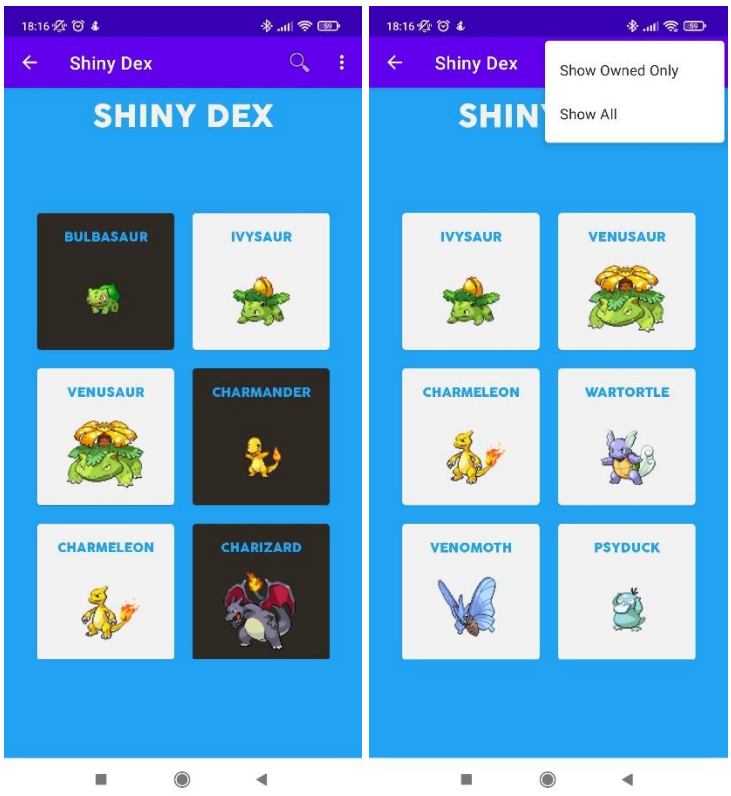


Imagem 16 – Actividade ShinyDex e filtro

2.4 Diagrama de Base de Dados

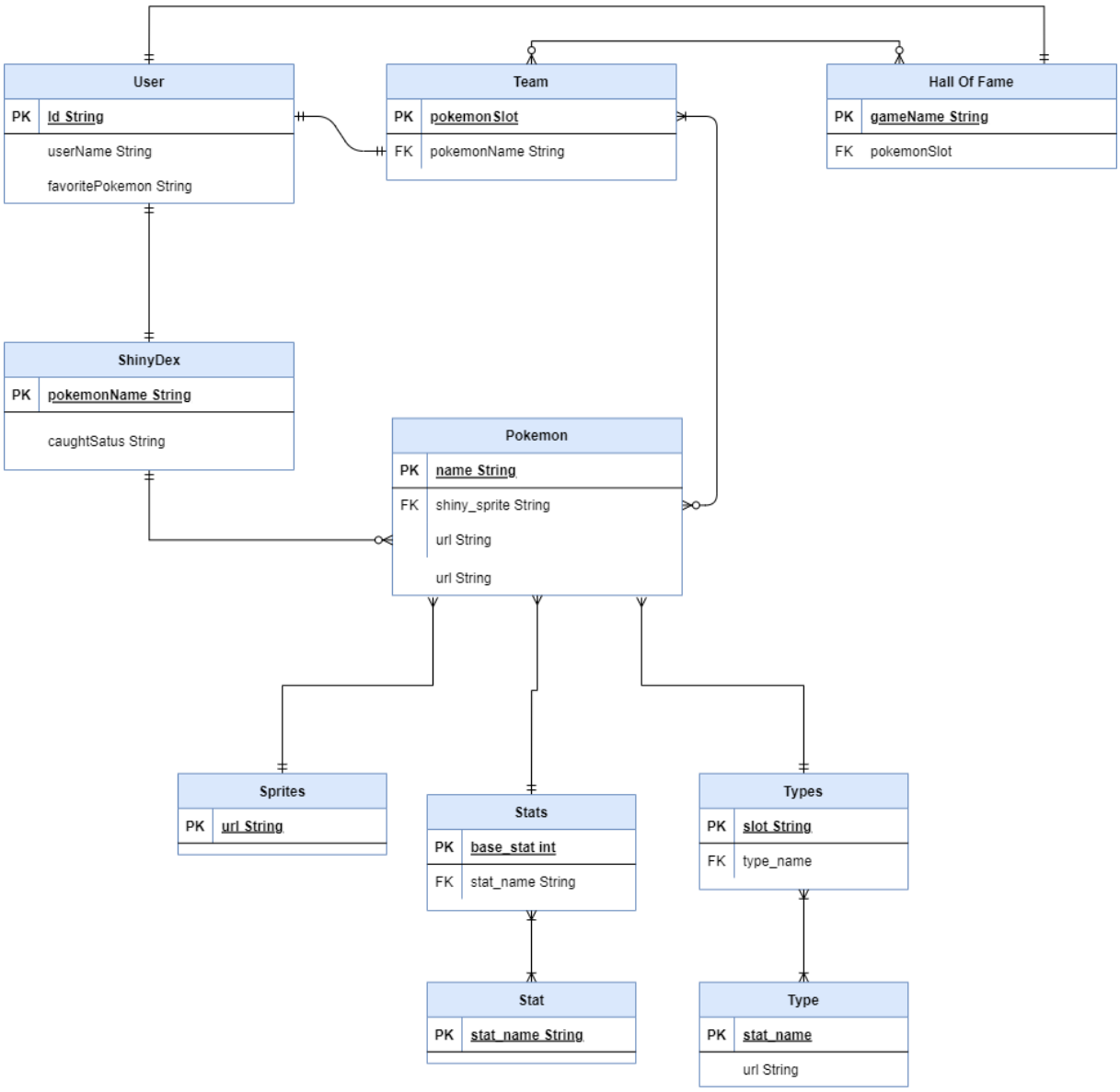


Imagem 17 – Diagrama da Base de Dados

2.5 Diagrama UML de Classes

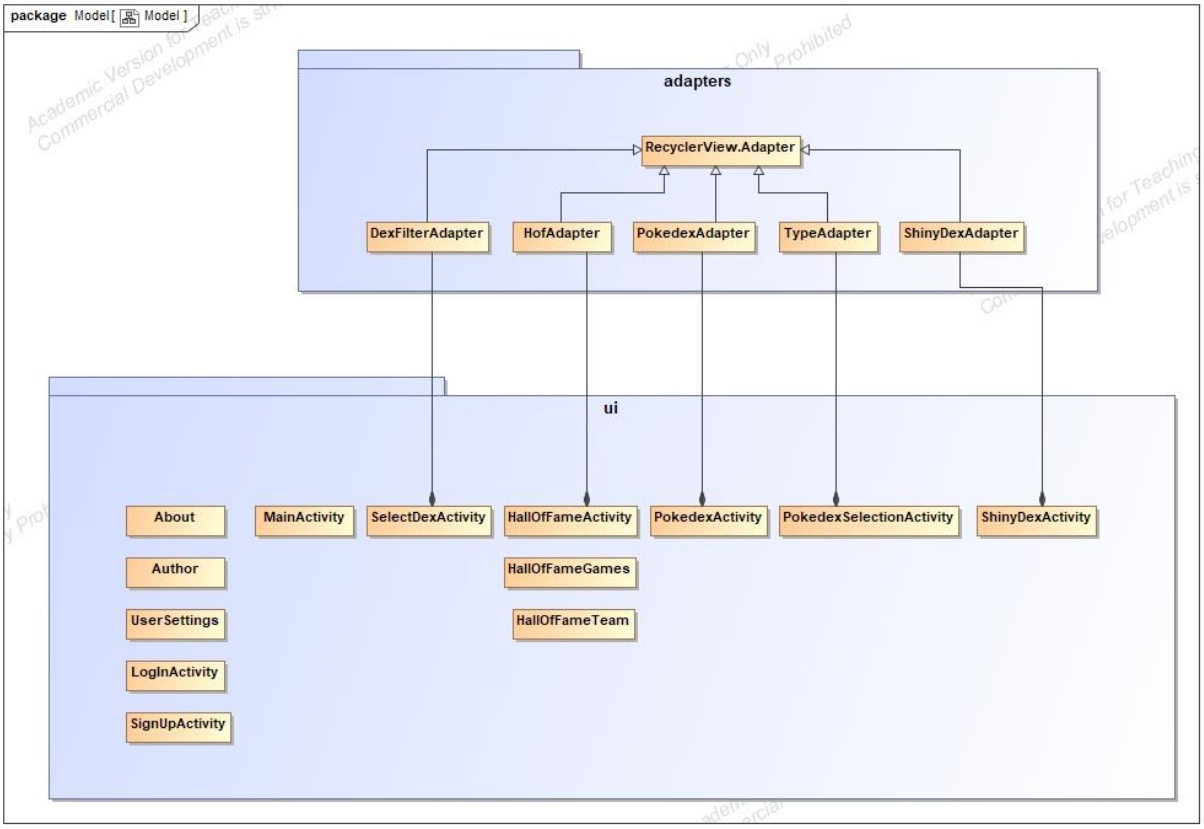


Imagem 18 – Diagrama UML de Classes

3. Resultados Obtidos

Como se pode observar da imagem 12 à 16 podemos observar que os resultados obtidos vão de encontro relativamente ao que foi projectado desde a Wire-Frame até ao Design do projecto. A construção do projecto foi toda feita na base da api **pokeapi.co**, que é uma api que contém toda a informação que é obtida no mesmo.

Para obtermos a informação da API foram usadas duas ferramentas externas designadas por Retrofit e HttpOkClient. A ferramenta HttpOkClient permite transformar uma página HTML numa string e em conjunto com o Retrofit, que é uma ferramenta que permite manipular esta mesma informação, transformando-a numa interface JAVA.

```
companion object{
    var BASE_URL = "https://pokeapi.co/api/v2/"

    fun create() : PokemonApi {
        val loggingInterceptor = if (BuildConfig.DEBUG){
            HttpLoggingInterceptor().setLevel(HttpLoggingInterceptor.Level.BODY)
        } else {
            HttpLoggingInterceptor().setLevel(HttpLoggingInterceptor.Level.NONE)
        }
        val client = OkHttpClient.Builder()
            .addInterceptor(loggingInterceptor)
            .build()
        val retrofit = Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .client(client)
            .build()
        return retrofit.create(PokemonApi::class.java)
    }
}
```

Imagem 18 – Função que manipula a API

Os métodos construídos dentro desta interface, PokemonApi, fazem uso das classes de dados criadas no ficheiro PokemonModel. Os atributos das classes de dados têm de corresponder, aos atributos obtidos no query efectuado pelo PokemonApi, sendo que é possível obter resultados, encapsulados em resultados se seguirmos a documentação fornecida pelo próprio fornecedor da API. Por vezes isso torna-se complexo e gera problemas que já iremos observar mais à frente.

```
val pokemonApi = PokemonApi.create().getPokemon(pokemonName)

pokemonApi.enqueue(object : Callback<Pokemon> {
    override fun onResponse(call: Call<Pokemon>, response: Response<Pokemon>) {
        Log.d( tag: "Pokemon",  msg: "Success")
        if(response.body() != null){

            var name: String = response.body()!!.name
            var url : String = response.body()!!.getImageUrl()
            text.text = name.uppercase()
            Glide.with(image.context)
                .load(url)
                .into(image);
        }
        else
            Toast.makeText(applicationContext,  text: "Invalid Name or Id", Toast.LENGTH_SHORT).show()
    }

    override fun onFailure(call: Call<Pokemon>, t: Throwable) {
        Log.d( tag: "Pokemon",  msg: "Failed")
    }
```

Imagem 18 – Exemplo do uso de PokemonApi numa actividade

Dentro de uma actividade chamamos então um objecto PokemonApi, em que o seu método enqueue, irá gerar dois métodos:

- onResponse que obtém informação se o pedido tiver sucesso, sendo que a informação obtida é depois manipulada pelas classes de dados de PokemonModel.
- onFailure caso o pedido tiver falhado

Relativamente à base de dados Firebase, podemos verificar que a informação que está a ser gravada e obtida se encontra correcta e coesa.

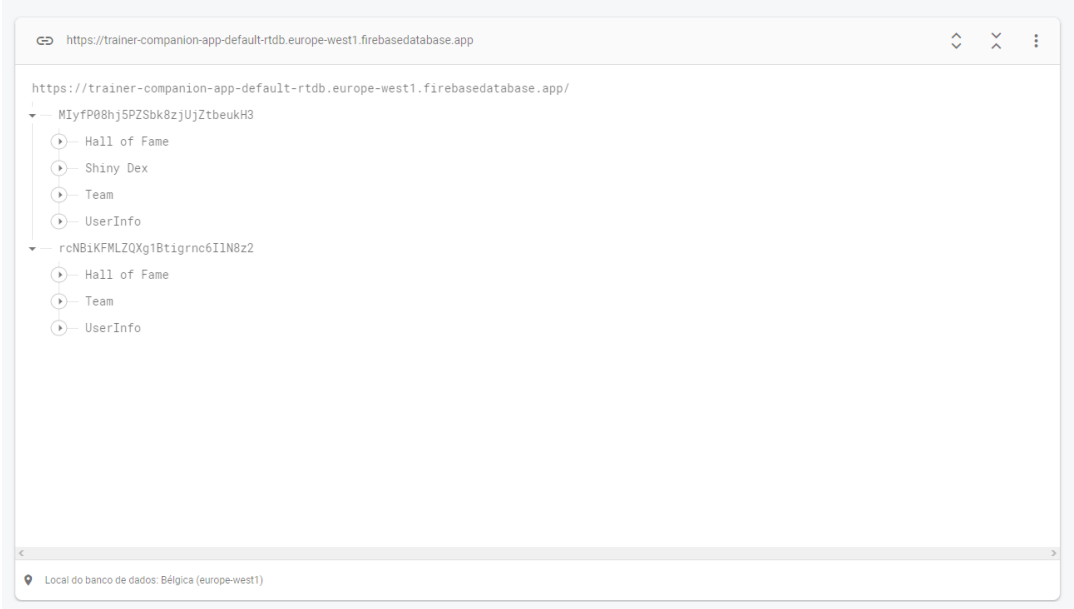


Imagem 19 – Resultados da Realtime Database de Firebase

Os valores iniciais que vemos nestas árvores, são os identificadores únicos dos clientes. De seguida podemos observar que na árvore da base de dados temos a informação relativamente à aplicação guardada.

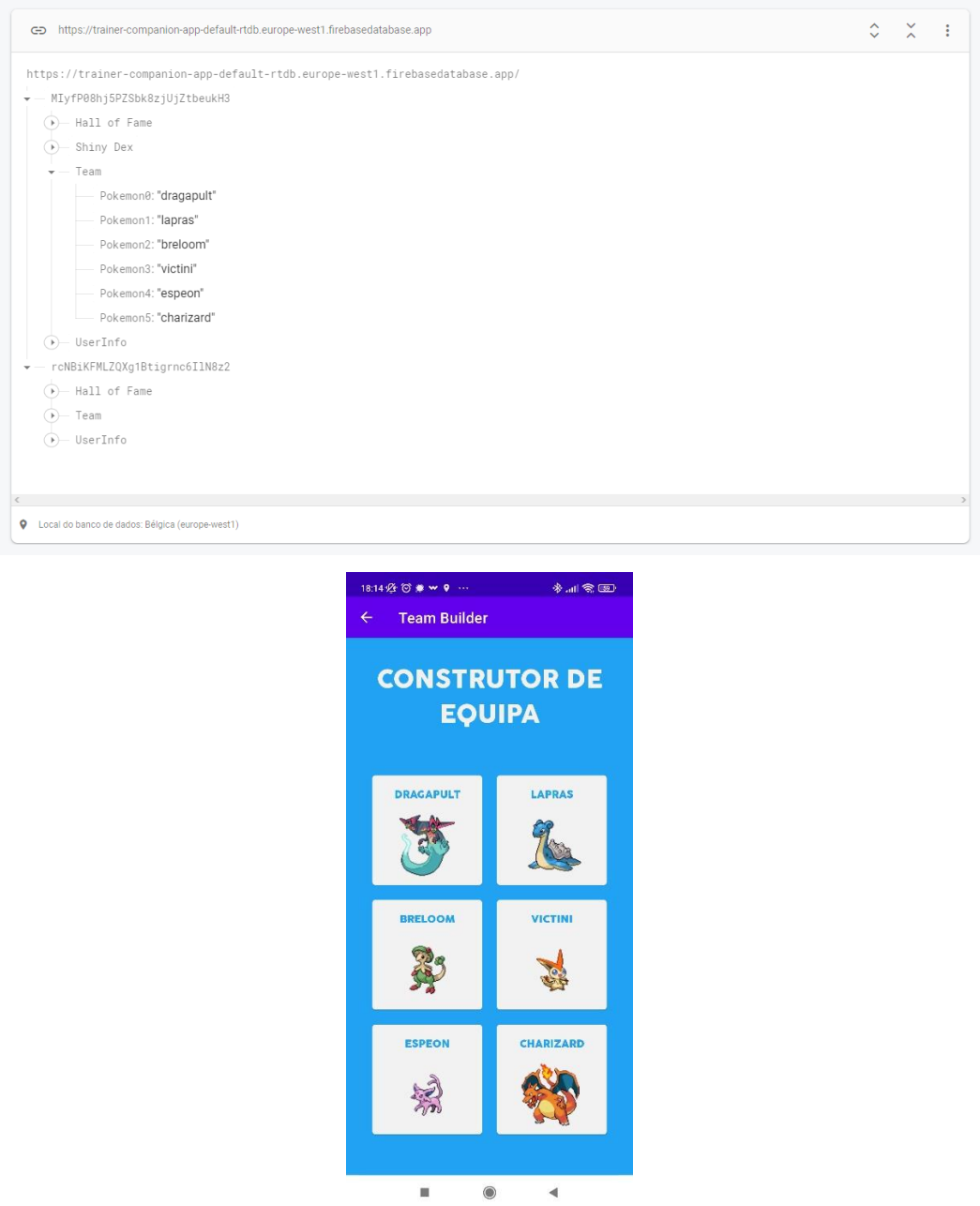


Imagem 19 – Resultados da Realtime Database de Firebase relativamente à Aplicação

Neste exemplo vemos que a informação presente na base de dados vai exactamente de encontro à que temos na aplicação, vendo assim que está a existir uma troca informação correcta entre ambas as partes.

Na aplicação são usados sons que são os sons originais do jogo Pokémon Red quando é efectuada a navegação entre actividades e em certos onClickListeners.

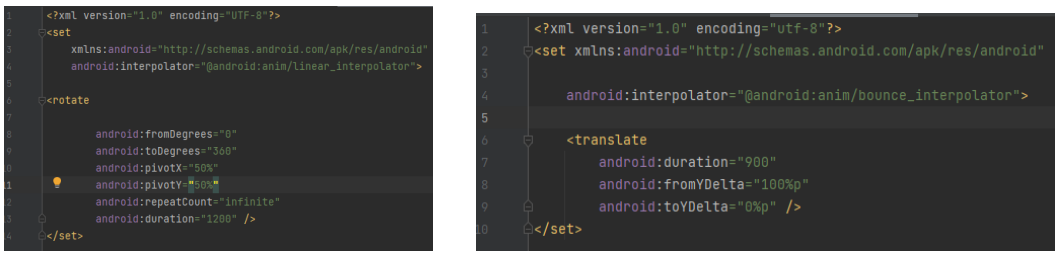


Imagem 20 – Animações Usadas

Nas animações usadas podemos ver o uso de *Interpolators* diferentes. No primeiro exemplo vemos um *Linear_Interpolator*, que faz com que a animação se mantenha sempre constante ao longo do tempo, evitando observar pequenos "*sutters*" quando a animação recomeça quando esta está em *loop*, como é o caso.

Já o *bounce_interpolator* faz com que a animação tenha um pequeno efeito de *bounce*/salto elástico quando é efectuada.

Quando olhamos para o que estava projectado e o que foi desenvolvido existem dois pontos que na altura que este relatório está a ser escrito não foram implementados. Dentro da actividade TeamBuilder existia uma funcionalidade para analisar os tipos da equipa e obter as fraquezas, os pontos fortes e tipos neutros. No entanto, embora seja possível obter os tipos de cada Pokémon, seria necessário fazer uma elevada quantidade de código para associar a cada tipo as suas fraquezas, os tipos contra os quais é forte e os tipos contra os quais é fraco. Depois disso seria necessário reunir todos os tipos reunidos na equipa e comparar todos estes pontos entre eles, sendo que existem actualmente 18 tipos possíveis e para cada 1 tipo tem de se ver qual o efeito dos restantes 17.

Outro ponto seria adicionar na página do Pokémon quando seleccionado do pokédex a sua cadeia evolucionária. Neste caso o problema encontrado foi a complexidade de navegação entre a API, visto que na página de um Pokémon não se encontram as suas evoluções apenas uma referência, que por sua vez leva-nos a outra referência, tornando o Path efectivo relativamente complexo, sendo que a página final onde se encontram as evoluções por sua vez não é de fácil entendimento, sendo que a documentação disponível apresenta um exemplo para um Pokémon com uma única evolução, onde os casos mais comuns é existirem duas e existem casos para os quais não existe nenhuma.

4. Conclusões

O projecto final obtido no geral deixou um sentimento de satisfação, não só pelo produto final em si, mas pela quantidade de informação que foi adquirida e consolidada ao longo do projecto.

A manipulação de API's e o uso da base de dados Firebase são ferramentas que serão imensamente valiosas para o futuro, seja a nível profissional ou individual.

Algo que foi muito usado neste trabalho e também se provou uma ferramenta essencial foi o uso de adapters para RecyclerViews, sendo uma das ferramentas mais flexíveis disponíveis para a manipulação de informação em grande escala, especialmente quando não sabemos quanta informação vamos ter disponível.

Quase todos os problemas encontrados tiveram sempre uma solução associada, seja esta obtida por pesquisa ou por consulta ao que foi anteriormente desenvolvido nos outros trabalhos da unidade curricular.

Este trabalho foi exclusivamente feito em Kotlin, pelo que daqui foram adquiridos novos conhecimentos. A linguagem Kotlin provou ser algo que simplifica imenso a construção de devido à sua sintaxe, e de fácil aprendizagem pois é muito semelhante a JAVA em vários aspectos.

Retirando os pontos negativos, ficou por implementar o uso de ViewModels neste projecto, que embora seja uma ferramenta poderosa, não foi de fácil de compreensão daí o seu uso não estar presente.