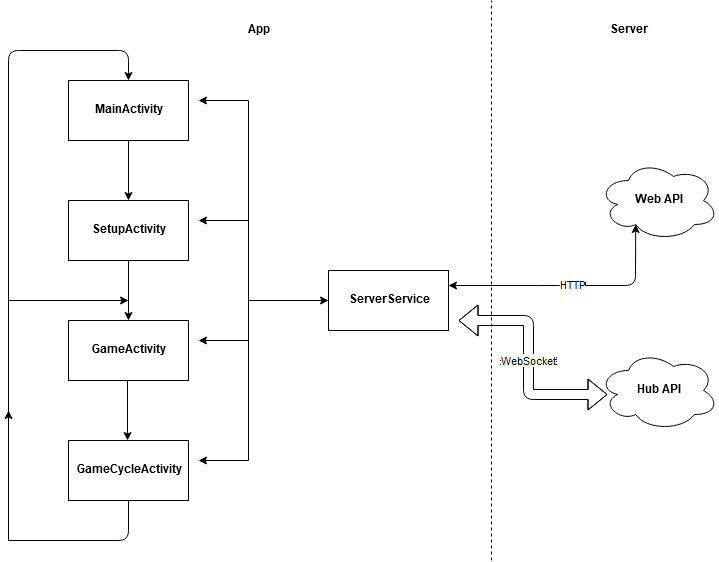
## Aplicação Android

O servidor trata de gerir as sessões, os jogadores e o jogo, a SPA trata de representar o estado de jogo que é partilhado a todos os utilizadores, já a aplicação Android tem como propósito servir de “comando”, é através dela que os utilizadores interagem com o jogo.

Registam-se na sessão de jogo, definem o seu *username*, desenham os temas que lhes são propostos e tentam adivinhar o tema dos desenhos dos outros jogadores que são apresentados.

## Arquitectura

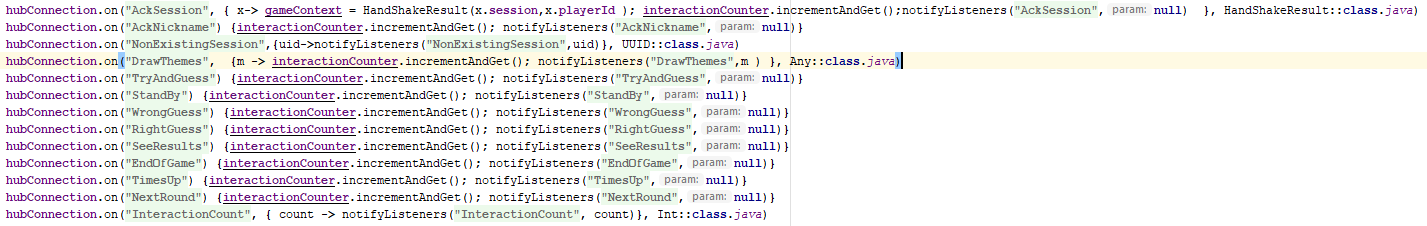
Desenvolvida em **Kotlin**, esta aplicação tira partido de algumas bibliotecas externas, sendo a mais central no seu desenvolvimento a biblioteca de *WebSockets* ***SignalR.*** A aplicação usa *WebSockets* para se conectar com o Servidor, com a qual mantém comunicação constante e se torna apta a escutar as alterações às fases de jogo, geridas pelo mesmo. Acede também à API do servidor, de forma a executar algumas ações que não se enquadram no paradigma das WebSockets, tal como enviar blocos grandes de data.



Figura

Por uma questão de organização, é apresentado de seguida um componente que é crucial a todo o fluxo da aplicação, o **ServerService**.

**ServerService** é um *Bound-Service*, que trata de estabelecer, e manter, a comunicação com o Servidor.

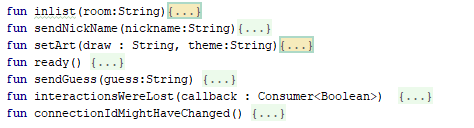
É criada uma instância de *HubConnection* através de um *HubConnectionBuilder*, dá lhe inicio chamando a função *start()* e de seguida define comportamento para todos os endpoints que se compromete a receber.

Figura

É criada uma lista de *listeners* do tipo da interface **ServiceListener**, de forma a que as *Activities* que pretendem tratar das notificações do **Servidor**, se possam colocar à escuta. O serviço contém também uma instância de um objeto do tipo **HandShakeResult** que é preenchida na primeira comunicação com o **Servidor**, que representa o contexto e contem os dados que identificam a *Sessão* e o *Jogador*, este contexto é passado em todas as comunicações posteriores.

De forma a ser utilizada a mesma instância do **ServerService** esta classe contem uma classe interna chamada **MyBinder** que estende de **Binder**, e contem um método “getService()” que retorna a instância do serviço.

Ao dar início ao Jogo, o servidor cria uma **Thread** que acorda de 5 em 5 minutos. Esta **Thread** é interrompida, e iniciada de novo a cada nova comunicação com o **Servidor**. Caso a **Thread** “acorde” dos seus 5 minutos de espera e não tenha sido interrompida então assume-se que o **Servidor** entrou em inatividade, como tal, o utilizador é alertado e a aplicação é encerrada. Esta observação só é quantificável após o início do jogo, pois a fase de inscrição não tem limite de tempo.

O serviço disponibiliza uma serie de funções, para serem chamadas pelas *Activities* quando querem comunicar com o **Servidor**

Figura

As Activities partilham uma estrutura geral. Todas ficam *bound* ao **ServerService**, implementam **ServiceListener** e ficam *listeners* do serviço, registam um *BroadcastReceiver* para detetar alterações de conectividade, e implementam comportamento para algumas notificações do servidor.

No *OnStart* das activities é registado o *BroadcastReceiver* com um *IntentFilter* instanciado com *WifiManager.NETWORK\_STATE\_CHANGED\_ACTION,* e definem o comportamento das alterações de conectividade.

É de notar que no **ServerService** existe um variável de controlo chamada “*connectionState*” do tipo AtomicBoolean, que é quando é detetada uma mudança de conexão é feita uma tentativa de alteração da variável pelas Activities, desta forma caso seja detetada mais que uma vez a alteração, ao avaliar se o “set” da variável foi feito com sucesso, as Activities sabem ao certo se o dispositivo já se encontrava nesse estado, ou que se trata de uma transição que implica as devidas ações.

Todas as interações entre o servidor e o serviço são registadas independentemente, durante a fase de jogo, caso a ligação à internet seja perdida e restabelecida, é feito um pedido http à API do servidor de forma a obter o valor do contador destas interações. Caso o contador do servidor, e do serviço sejam iguais, significa que não houve comunicações que se perdessem, e o fluxo da aplicação continua. Mas caso não sejam, tendo a aplicação perdido informação e podendo estar num estado diferente do atual estado do jogo, é apresentado ao utilizador que a falta de conectividade manifestou problemas de comunicação, e como tal não é possível prosseguir o jogo, reiniciando a aplicação.

De forma a identificar o dispositivo perante o servidor ao invocar o endpoint “*Inlist”* é passado um identificador único que será partilhado por todas as futuras sessões de futuro, e que serve para fazer persistir os desenhos numa base de dados de forma identificada.

Este identificador é auto gerado, e guardado nas **SharedPreferences**.

Foi tomada a decisão de gerar o identificador em vez de usar o ID do dispositivo, por uma questão conservadora em relação aos dados do utilizador.

**ServiceListener** é a interface que define o protocolo de comunicação entre o **ServerService** e as Activities, e contem apenas uma função com a seguinte assinatura “*fun Interaction(action:String, param:Any?)*”, quando o serviço recebe uma notificação do servidor, invoca esta função passando como parâmetros a ação enviada pelo servidor e os respetivos parâmetros aos *listeners.*

A **MainActivity** é o ponto de entrada da aplicação, apresenta ao utilizador uma caixa de texto onde este deve inserir o código da sala de jogo*,* inicia o serviço **ServerService** e fica *Bounded* ao mesmo.

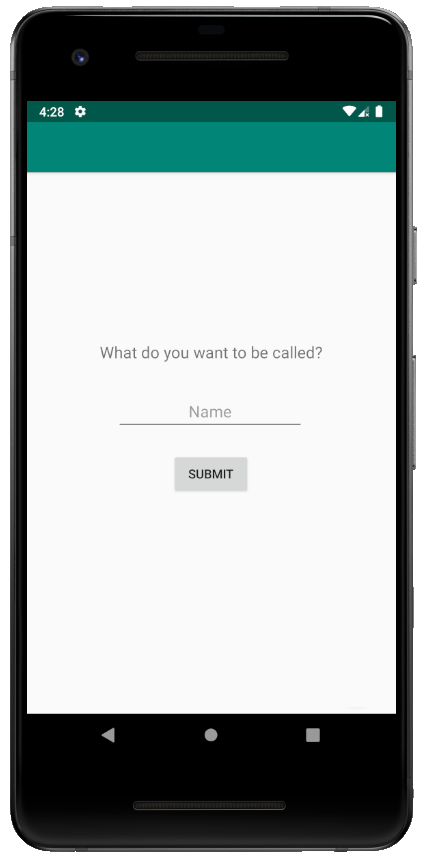
Figura

Visto que é este o início da aplicação, é testada a conectividade do dispositivo à internet. Caso o dispositivo se encontre conectado, o fluxo prossegue, caso esteja desconectado, o utilizador recebe uma mensagem alertando para o facto.

No caso de a aplicação ser iniciada sem conectividade, só depois do utilizador ligar o acesso à internet é que é iniciado o serviço.

Estas mudanças de conectividade definem também se botões são ou não clicáveis.

Após inserido o código e de ser comunicado ao servidor, o botão é desativado, de forma a que um utilizador não se consiga registar mais que uma vez, e a **MainActivity** aguarda passivamente pela notificação do servidor, que tanto pode ser “*AckSession”* ou *“NonExistingSession”,* no caso da primeira é lançada a **SetupActivity**, na segunda é ativado o botão de novo de forma a que o utilizador possa reinserir o código.



Figura

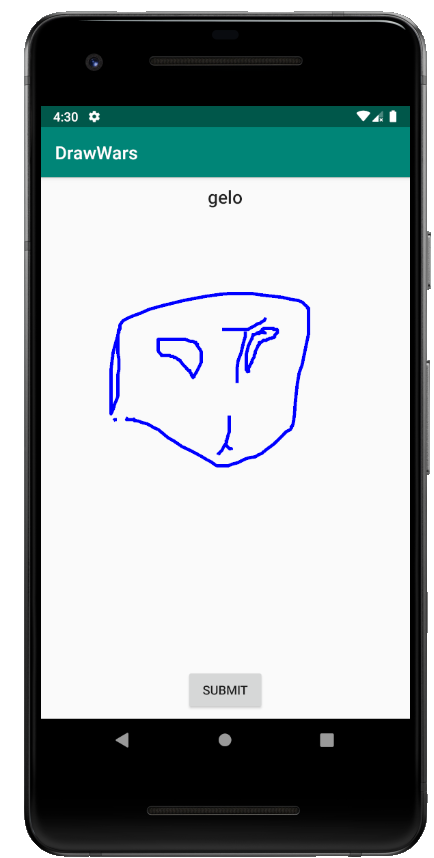
Na **SetupActivity** o utilizador já se encontra associado à sessão de jogo, e deve configurar o seu *username* para a sessão.

Esta Activity após enviar o *nickname* para o servidor, também através da interface do **ServerService**, aguarda pela notificação de “*AckNickname*” e quando a recebe progride para a **GameActivity**.

**GameActivity**, nesta activity é apresentada uma mensagem que alerta o jogador que ainda não se juntaram todos os jogadores, e como tal, este deve esperar, ou avisar, através de um botão (ALL READY)”, que todos os jogadores estão prontos. Quando clicado, esta mensagem é transmitida ao servidor, deixando a **GameActivity** aguardando passivamente pela notificação de “*DrawThemes”.*

Figura

Visto que é a partir de este ponto que o jogo efetivamente começa, esta Activity trata de implementar comportamento para a notificação de inatividade do servidor, por parte do **ServerService**.



Figura

A **GameActivity** tira proveito do ViewModel, instanciando **DWCanvasViewModel** que estende de **ViewModel**, esta classe contém uma instância **DWCanvas** como **MutableLiveData** com o objetivo de poder observar e manter viva a instância através do *lifecycle* da Activity.

**DWCanvas** é uma *Custom View* que implementa o comportamento de desenho num Canvas, e que é capaz de o retornar em formato String Base64.

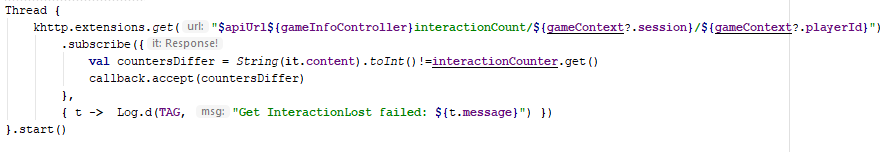
A receção da notificação de “*DrawThemes*” é acompanhada do tema que deve ser desenhado pelo utilizador, e é apresentado em ecrã. A aplicação ao interpretar esta notificação, ativa a tela permitindo que o utilizador esboce o tema apresentado.

A ação de “*SUMBIT”* chama, do serviço, uma função que trata de enviar o desenho em Base64 para o servidor.

Como o desenho tem um tamanho considerável, esse envio não é feito via *WebSocket* mas sim via POST Http para a Web API do servidor, através da biblioteca **Khttp**.

**Khttp**[1]é uma biblioteca HTTP para **JAVA** e **Kotlin**, que agiliza a comunicação via http, com a sua interface limpa e simplista, permite executar pedidos como POST e GET com bastante facilidade.

Figura



Figura

Se o pedido de POST executar com sucesso, o servidor é notificado via *WebSockets* que já foi feito o upload do desenho e a **GameActivy** aguarda passivamente por Feedback.

Caso o servidor notifique a aplicação com a mensagem “*DrawSubmitted*”, é lançada a **GameCycleActivity**.

No caso de o utilizador não enviar o desenho atempadamente, tendo em conta o temporizador que é apresentado no frontend durante esta fase, a activity será notificada que o tempo de desenho acabou e é forçada a enviar o desenho.



Figura

Figura

**GameCycleActivity**, esta Activity corresponde ao ciclo de apresentação dos desenhos do jogo. Caso o desenho apresentado no frontend seja o enviado pelo utilizador, então este será informado que é o seu desenho a ser apresentado (Figura 11).

Caso seja o desenho de um outro utilizador, é apresentada uma caixa de texto onde devem ser inseridas as *Guesses*.

O feedback das *Guesses* informa a app se o utilizador errou, devendo continuar a tentar até ao tempo acabar, ou se acertou.

Ao terminar a ronda, o servidor avaliará se é o fim de jogo, ou ainda existem turnos a ser jogados. Essa informação recebida pela aplicação em forma de notificação, e a **GameCycleActivity** decide, se deve lançar de novo a **GameActivity** ou **MainActivity** nomeadamente se a mensagem for “NextRound” ou “EndOfGame” respectivamente.

(Talvez uma imagem do fluxo de jogo)