Uma imagem com texto

Descrição gerada automaticamente

Socialmind

Computação Móvel

Luís Chaves-114683

Miguel Monteiro-111763

Nuno Veloso-114715

Mestrado em Desenvolvimento de Jogos Digitais

Novembro de 2022

**Index**

[Equipa 3](#_Toc118682862)

[Motivação 4](#_Toc118682864)

[Requerimentos e Funcionalidades 5](#_Toc118682865)

[Arquitetura e Implementação 6](#_Toc118682866)

[Apreciação Global 13](#_Toc118682867)

[Cumprimento de Objetivos 14](#_Toc118682868)

[Contribuição Individual 15](#_Toc118682869)

[Tutorial de Utilização 16](#_Toc118682870)

Equipa

Grupo 3 – Socialnary/Socialmind **Link:** <https://github.com/luischavesdev/Socialmind>

* A picture containing wall, person, person, indoor

  Description automatically generatedLuís Chaves – 114683
* A picture containing necktie, person, wall, person

  Description automatically generatedMiguel Monteiro – 111763
* Uma imagem com parede, pessoa, óculos, homem

  Descrição gerada automaticamenteNuno Veloso - 114715

Motivação

Após o impacto que a recente pandemia teve na população mundial, a sociedade começa, gradualmente, a voltar ao normal. Com esta tendência, a necessidade de socializar e conectar com outros passa a ser viável outra vez, e que forma melhor de o fazer do que através do clássico jogo *Mastermind*. Através da combinação de características de teor social de aplicações de Realidade Aumentada, como *Pokemon Go*, e a simplicidade de *Mastermind*, surge a nossa aplicação, *Socialmind*.

Requerimentos e Funcionalidades

Em termos de requerimentos, o nosso jogo precisa de permitir aos jogadores encontrarem outros jogadores de forma a poderem ser realizadas partidas de *Mastermind*, assim como personalizar um perfil pessoal com o intuito de haver progressão no jogo e haver diferenciação entre jogadores. Com isto em mente, algumas funcionalidades foram pensadas desde o início, as quais irão ser apresentadas de seguida:

* Iniciar um jogo ou entrar num já iniciado por outro jogador
* Alterar o nome e avatar
* Localizar jogadores próximos num mapa
* Alterar a linguagem da aplicação

Dadas estas funcionalidades, foram inicialmente definidos os seguintes elementos / *plugins* que seriam necessários para concretizar estas funcionalidades:

* Câmara / Carregamento de imagem
* Utilização de sensores movimento para o *gameplay*
* *Broker MQTT* para fazer a comunicação de jogo
* *GPS* / *Google Maps*
* Base de dados
* Linguagem / Internacionalização

Arquitetura e Implementação

Uma imagem com texto

Descrição gerada automaticamentePara implementar a funcionalidade de alterar o nome, foi definida uma página inicial da aplicação, onde o utilizador pode inserir um nome à sua escolha ou alterá-lo, caso já tenha criado um em sessões anteriores.

A página em si é relativamente simples, apresentando apenas um campo de texto e um botão para submeter o nome e entrar na aplicação em si. No entanto, de forma a conseguir fazer com que o nome persista mesmo após o fecho da aplicação, foi utilizado o *plugin shared\_preferences*, que disponibiliza armazenamento local persistente para dados simples.

Após obter um *handler* para poder realizar operações de guardar/recuperar dados, através do método *SharedPreferences.getInstance()*, foi apenas necessário recorrer às funções *getString()*  e *setString()* para, respetivamente, obter o nome guardado anteriormente e colocar/substituir pelo novo

No que diz respeito a possibilitar ao utilizador definir um avatar através da utilização direta da câmara do dispositivo ou através do carregamento de uma imagem do armazenamento, tirámos partido do *plugin image\_picker*, através do método *pickImage()*. Neste método, é possível especificar a fonte da imagem, podendo assim definir-se se a imagem vem da câmara (resultando na criação de uma fotografia no momento) ou da galeria (resultando no carregamento da imagem do armazenamento interno). A nível da interface, esta página é mais complexa, sendo usado o *widget Stack*, para poder empilhar os botões sobre o *display* da imagem, dando assim o efeito de sobreposição. Adicionalmente, os botões necessitaram de ser envolvidos num *widget Positioned*, de forma a poder posicionar individualmente cada um dentro do mesmo espaço.

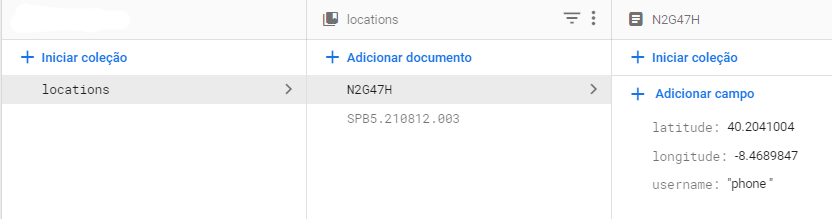
**Graphical user interface

Description automatically generated with low confidence**

É importante notar ainda que, em ambas as páginas, foram usados *widgets* do tipo *FutureBuilder* para poder construir a interface com base em valores que estavam a ser obtidos de serviços assíncronos (no primeiro caso, o nome guardado em *Shared Preferences* e, no segundo, o caminho para a imagem, guardado também através de persistência local).

Relativamente à localização de jogadores próximos num mapa, foi criada uma página específica que, embora seja composta por um único *widget*, tira partido de múltiplos *plugins*:

* *google\_maps\_flutter*: através deste *plugin*, é possível criar um *widget* do tipo *GoogleMap*, que cria um mapa geográfico com centro na localização do utilizador, e ao qual é possível associar um método a correr quando este é criado.
* *location*: permite obter a localização do utilizador e criar um *listener* para fazer alterações no mapa sempre que esta muda, no método referido acima.
* *cloud\_firestore*: dentro do método referido acima, é também utilizado o suporte de uma base de dados, *Firestore*, de forma a guardar os dados de cada utilizador (coordenadas geográficas, nome e identificador único do dispositivo). É possível também obter estes dados por um utilizador, de forma a mostrar a localização de todos os utilizadores do sistema no seu mapa.
* *device\_info\_plus*: permite obter informações sobre o dispositivo físico, incluindo um identificador único, utilizado para representar cada utilizador na base de dados.



Informações de localização na base de dados.

O resultado é um mapa que apresenta constantemente a localização de um utilizador, assim como a de outros que estejam na mesma página da sua aplicação, através de marcadores com as informações de cada.

Uma imagem com texto, árvore

Descrição gerada automaticamente

É de notar que a implementação desta página foi de forma que o utilizador passe a ser ou deixe de ser visível a outros utilizadores se, respetivamente, entrar ou sair desta página.

A nível da alteração de idioma da aplicação, foi utilizado o padrão de internacionalização em *Flutter*[[1]](#footnote-1), que permite, através de ficheiros específicos do tipo *.arb*, definir valores diferentes para diferentes elementos de texto, que se adaptam ao idioma utilizado pelo dispositivo.

A screenshot of a computer

Description automatically generated

Após definir estes valores, estes podem ser acedidos diretamente na construção da aplicação, para permitir texto flexível com a linguagem do utilizador. Isto resulta em interfaces adaptadas às preferências de idioma do jogador.

Uma imagem com texto, captura de ecrã, ecrã

Descrição gerada automaticamente

Isto resulta em interfaces adaptadas às preferências de idioma do jogador.

**Graphical user interface, application

Description automatically generated**Graphical user interface, application

Description automatically generated

Em termos de comunicação MQTT, foi implementado um broker através do *RabbitMQ*, que apesar de não providenciar um broker por defeito que comunique com o protocolo previamente mencionado, disponibiliza um plugin para esse efeito. Após criar o broker e defini-lo para correr na máquina local, grande parte do desenvolvimento foi feito utilizando dois emuladores a correr a aplicação, apesar de ter sido criado um *user* do *broker* de forma a permitir a conexão com outros dispositivos desde que estejam ligados à mesma rede local. Isto por sua vez foi testado, e conseguimos comunicar com o broker através de um dispositivo físico.

Do lado da implementação, grande parte da funcionalidade vem do *package mqtt\_client*. Através deste, podemos criar um cliente, o qual tem que ser configurado com o IP/*port* corretos e um ID único, que nos permite subscrever a um tópico criado de antemão no *broker*. Isto faz com que quaisquer mensagens mandadas para esse tópico, sejam recebidas no cliente através do método *listen().* Toda a comunicação do jogo é feita através de um único tópico, sendo cada tópico o correspondente a uma sala de jogo, no qual só podem fazer parte dois jogadores.

Text

Description automatically generated

Criação do cliente

Uma vez o estado do jogo está em constante mudança, resolvemos utilizar um *ChangeNotifierProvider* para realizar alterações em termos de UI de acordo com as mudanças da lógica. Situações concretas incluem, por exemplo, o tempo de espera até que ambos os jogadores tentem adivinhar o código para o turno em questão. Ao clicar no botão para submeter a tentativa de adivinhar o código adversário, é publicada uma mensagem no broker, que por sua vez vai ser recebida por ambos os jogadores. Como o método que publica mensagens no *broker* associa o ID de cada jogador à mensagem propriamente dita, quando esta é recebida, o *manager* das comunicações MQTT sabe se a mensagem foi enviada por ele próprio, ou pelo oponente, atualizando as variáveis booleanas respetivas. Quando for recebida uma mensagem de submissão de código e a ambas as variáveis que coordenam quem já submeteu forem verdadeiras, uma outra variável do *Provider* responsável por manter atualizado o número do turno em questão é incrementada, chamando os *notifiers.* Neste caso, as linhas de UI com as várias tentativas são identificadas como *Consumers*, portanto ao receberem a nova variável dos turnos, vão atualizar a sua visibilidade corretamente.

Text

Description automatically generated

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generated

Processamento da mensagem

Incremento do turno

Definição dos *Providers* na base da árvore

Publicação da mensagem de tentativa



Definição da *Row* como *Consumer* e transferência da variável dos turnos

Relativamente à estruturação do jogo, os diversos elementos foram construídos da forma mais modular possível, sendo que a *play\_page* contém todos os elementos de forma mais ou menos hierárquica. Os botões que permitem mudar a cor vêm primeiro, seguidos dos ícones que representam a precisão da tentativa do jogador. Ambos estes elementos são depois agrupados em *widgets* que tratam a informação respetiva de vários botões/ícones, os quais são referidos como *Button Row,* e *Guess Square.* Por fim, temos os botões que o jogador usa para definir estados de jogo, que são o *Set Code Button*, para definir o código a adivinhar pelo oponente, e o *Submit Button*, para submeter uma tentativa de adivinhar o código do adversário. No *widget* da página em si, são os widgets mencionados são construídos de forma a criar a estrutura da página, sendo que métodos de certos *widgets* são acedidos através de *Controllers*, que guardam referênciais para funções.

Apreciação Global

Em primeiro lugar, achamos que é importante referir que grande parte da nossa abordagem ao desenvolvimento deste projeto, foi com uma mentalidade de que o projeto tinha de funcionar como prova de conceito, e não necessariamente como uma aplicação final com um nível de refinamento respetivo. Desta forma, se calhar focámo-nos demasiado em implementar os aspetos técnicos, e fazer com que os mesmos funcionassem. Dito isto, as funcionalidades principais, como o mapa, a customização do perfil com uma foto, o multijogador através de MQTT e até mesmo outras funcionalidades como a utilização de sensores, foram implementadas com sucesso, mesmo que estas se calhar funcionem apenas em determinado ambiente, ou não contemplem todas as variáveis que poderiam existir num cenário de aplicação final.

Para além disto, os diferentes elementos do grupo tiveram de experimentar tecnologias novas, havendo sempre um processo de habituação associado a este tipo de experiências. Por outro lado, esta obrigação em sair da zona de conforto e experimentar outras coisas revela-se sempre bastante enriquecedora, tanto pelo facto de aplicarmos na prática e ficarmos mais à vontade com conceitos abordados na aula, como os *Providers,* mas também no sentido oposto, como por exemplo a comunicação através de MQTT, a qual esperávamos que tivesse mais funcionalidades, e deste modo ficamos consciencializados em relação à utilização da mesma em projetos futuros.

Cumprimento de Objetivos

Tal com foi dito anteriormente, os objetivos principais foram cumpridos, apesar destes objetivos principais divergirem um bocado da ideia inicial do projeto. Efetivamente a ideia de passar de um jogo que se focava no desenho através do ecrã do telemóvel para um jogo mais estático, foi uma boa ideia, uma vez que o próprio processo de comunicação entre jogadores implicou algum trabalho. Dito isto, funcionalidades como o uso de Bluetooth foram retiradas em prol do uso exclusivo de MQTT. De forma a facilitar o processo de comunicação entre aplicações fora do contexto de um jogo 1v1, como por exemplo a partilha da localização no mapa, decidiu-se utilizar a Firebase como plataforma de base de dados, uma vez que já havia alguma experiência com esta por alguns elementos do grupo, permitindo assim o acelerar do processo de desenvolvimento. Outras funcionalidades foram simplesmente deixando-se ficar para trás, acabando por não haver tempo para as implementar, como por exemplo uma tabela de lideranças que seria atualizada de cada vez que um jogo era concluído, ou o suporte para várias salas de jogo em simultâneo.

Contribuição Individual

De seguida, nesta secção tentamos apresentar uma perspetiva sobre a contribuição dos diferentes elementos do grupo em relação ao desenvolvimento deste projeto:

* Luís Chaves: Responsável por implementar a lógica de jogo e pela parte técnica relacionada com as comunicações através de MQTT, que por sua vez implicaram a gestão de dados através de um *ChangeNotifierProvider*. **Percentagem de contribuição:** 40%
* Miguel Monteiro:
* Nuno Veloso: Responsável pela criação da página inicial, mapa e secção do utilizador e, a nível técnico, responsável pela implementação dos *plugins* relacionados com a base de dados *Firestore*, obtenção de fotos através do *plugin image\_picker*, implementação do mapa através do *plugin flutter\_google\_maps*, *shared\_preferences* e também a secção relacionada com a internacionalização. Responsável ainda por criar um dos *providers* utilizados na aplicação. **Percentagem de contribuição**: 38%

Tutorial de Utilização

Nesta secção descreve-se uma utilização típica da aplicação, o que no nosso caso é relativamente intuitivo devido à simplicidade da mesma.

O ecrã inicial é sempre o mesmo, e é apenas utilizado para o jogador registar o seu nome que é depois utilizado para identificar o mesmo num jogo contra outro jogador, uma vez que ao ambos jogadores receberem as mesmas mensagens, é preciso uma forma de distinguir que foi o responsável por enviar determinada mensagem.

Graphical user interface, application

Description automatically generated

Ecrã Inicial

Para além do ecrã inicial, a aplicação encontra-se dividida em três páginas. Uma página do mapa para permitir ao jogador encontrar outros jogadores nas proximidades, outra página de utilizador que permite personalização através de uma foto, e por fim uma página que permite ao jogador conectar-se ao servidor e jogar com outra pessoa que tente fazer o mesmo.

Em relação ao jogo em si, relativamente a aspetos técnicos, este funciona de forma que o jogo progrida apenas se ambos os jogadores estiverem no mesmo estado de jogo. Por exemplo: um jogador só pode jogar após se conectar ao broker, e receber a notificação de outra conexão, assim como só pode começar a tentar adivinhar o código do oponente depois de ambos os jogadores definirem o código secreto, tal como cada tentativa sucessiva só é disponibilizada após os dois jogadores submeterem uma tentativa no turno anterior. No que toca à lógica do jogo, este funciona tal como o *Mastermind* clássico, com a única diferença de que na nossa versão o multijogador é simétrico, em vez de haver um jogador com a função de criar um código, e outro com a função de o decifrar. Neste sentido, ambos os jogadores criam um código composto por 4 elementos, todos os quais podem ter 1 de 6 cores. Nos turnos seguintes, é suposto adivinhar o código do adversário o mais rápido possível, uma vez que o primeiro a fazê-lo ganha o jogo. Caso o jogo chegue até à ronda 10, é declarado o estado de empate. Conforme o jogador vai submetendo tentativas, um algoritmo é corrido assim que o botão de submissão é pressionado, e este usa os ícones ao lado do código de cores para registar um ícone branco caso uma das cores tenha sido corretamente adivinhada, mas colocada no sítio errado, ou um ícone preto caso o jogador tenha colocado uma cor igual no mesmo sítio do código do adversário. Através destas pistas e um consequente processo de dedução, é quase sempre possível adivinhar o código do adversário dentro dos 10 turnos disponibilizados.

Bubble chart

Description automatically generated with medium confidence

Ecrã de Jogo

Por fim, ao ganhar um jogo era suposto atualizar uma variável na base de dados que correspondia a uma pontuação global que depois seria visível numa tabela de lideranças, mas como foi referido anteriormente, esta funcionalidade acabou por ficar para trás.

1. Processo de implementação explicado em: <https://docs.flutter.dev/development/accessibility-and-localization/internationalization> [↑](#footnote-ref-1)