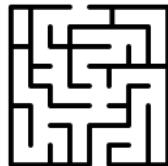
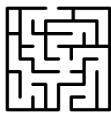


PI de Sistemas de Informação Distribuídos (24/25)

Licenciaturas em Engenharia Informática e Informática e Gestão de Empresas



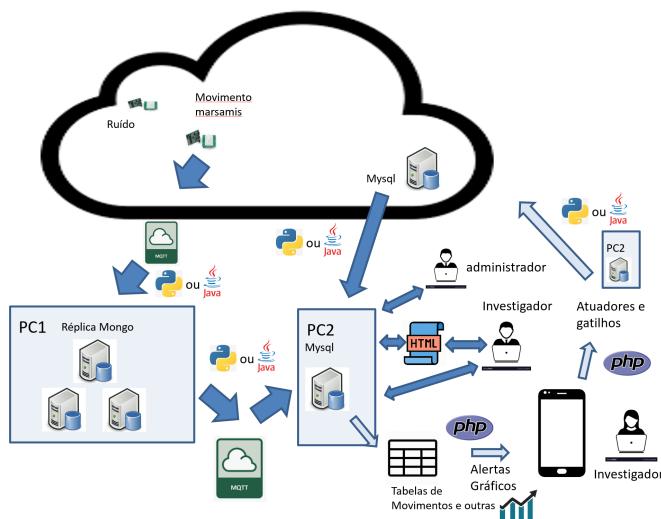
Grupo 9		Grupo 12	
110916 Davi de Mattos Balieiro dmbos@iscte-iul.pt		112028 Matilde Glória mggas@iscte-iul.pt	
110937 Ji Hua Zhu jhzua@iscte-iul.pt		111600 Inês Pedro Pinto ipflo@iscte-iul.pt	
111121 Guilherme da Mota Castilho Caramelo Riço gmccr@iscte-iul.pt		112412 Rita Ferreira Dias rfdsa3@iscte-iul.pt	
111213 Gonçalo Vieira Henriques gvhsa@iscte-iul.pt		110749 Elisabete Kirikov ekvel@iscte-ul.pt	
111255 Rodrigo Miguel Cosme dos Santos rmcss1@iscte-ul.pt		111188 Rodrigo Parente rjepe@iscte-ul.pt	
111257 Ricardo Fernandes Paulo Isidro rfpio@iscte-ul.pt		123456 Pedro Nogueira Ramos pnr@iscte-ul.pt	

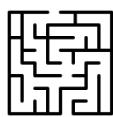


Instruções

Estas instruções são de cumprimento obrigatório. Relatórios que não cumpram as indicações serão penalizados na nota final.

- Podem (e em várias situações será necessário) ser adicionadas novas páginas ao relatório, mas não podem ser removidas páginas. Se uma secção não for relevante, fica em branco, não pode ser removida;
- A paginação tem de ser sequencial e não ter falhas;
- O índice tem de estar atualizado.
- O grupo que inicia o documento (coluna à esquerda na folha de rosto) preenche apenas a parte inicial (até ao fim da secção secção 1). Este documento word vai ser colocado no moodle para que o outro grupo (à direita da folha de rosto) possa descarregar e continuar a preenche-lo (secção 2)



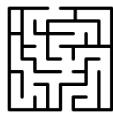


Índice

1	Especificação	5
1.1	Da Nuvem para o Mongo	5
1.2	Descrição Geral do Procedimento de Mongo Para Mysql	7
1.4	Tratamento de dados anómalos (valores de sensores errados)	11
1.5	Tratamento de outliers de temperaturas	13
1.6	Tratamento de Alertas de Som	15
1.7	Tratamento de número de marsamis numa sala (obter pontuação)	17
1.8	Especificação de Store Procedures SQL de apoio à migração e tratamento de dados	
19		
1.9	Especificação de Triggers de apoio à migração e tratamento de dados	20
1.10	Modelo Relacional	21
1.11	Utilizadores Base de Dados Mysql	23
1.12	Procedimentos Manutenção da Aplicação	24
1.13	Eventos de suporte à aplicação (caso existam)	25
1.14	Consulta por HTML/PHP	26
2	Implementação	28
2.1	Coleções a criar em cada uma das réplicas do Mongo	28
2.2	Descrição Geral do Procedimento de Mongo Para Mysql	31
2.3	Tratamento de dados anómalos (valores de sensores errados)	34
2.4	Tratamento de outliers de temperaturas	35
2.5	Tratamento de Alertas de Som	36
2.6	Tratamento de número de marsamis numa sala (obter pontuação)	38
2.7	Implementação de Stored Procedures SQL de apoio à migração e tratamento de	
dados	40	
2.8	Implementação de Triggers	41
1.1	Modelo Relacional	42
1.2	Utilizadores Base de Dados Mysql	43
1.3	Procedimentos Manutenção da Aplicação	45
1.4	Eventos de suporte à aplicação (caso existam)	48
1.5	PrintScreen dos formulários HTML implementados	50



1.6 PrintScreen do formulários Android com dados	53
Código de Triggers implementados	56
Código Stored Procedures implementados	59



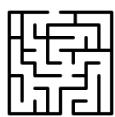
1 Especificação

Esta secção é onde o grupo que inicia o documento (coluna à esquerda na folha de rosto) coloca a especificação do que pretende implementar. Mais tarde pode implementar de outra maneira, mas aqui vão as primeiras ideias que serão avaliadas na primeira oral e que vão ser entregues a outro grupo para que analisem e vejam se aproveitam as vossas ideias.

1.1 Da Nuvem para o Mongo



Nome Coleção	O que armazena?
Movement	Player, Marsami, RoomOrigin, RoomDestiny, Status -> o movimento do Marsami captado pelo sensor da porta
Sound	Player, Hour, Sound -> o ruído no labirinto numa certa hora
Atuadores	Type, Player, RoomOrigin, RoomDestiny, Room



Para cada coleção exemplifica um documento

Coleção : movement

```
{ _id: ObjectId('67dbf608ad60fd16ebe719c0')
```

```
Player: 9
```

```
Marsami: 9
```

```
RoomOrigin: 1
```

```
RoomDestiny: 3
```

```
Status: 1 }
```

Coleção : sound

```
{ id: ObjectId('67dbf612ad60fd16ebe71a1d')
```

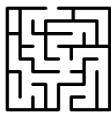
```
Player: 9
```

```
Hour: "2025-03-20 11:03:16.498354"
```

```
Sound: 19.196864203740272 }
```

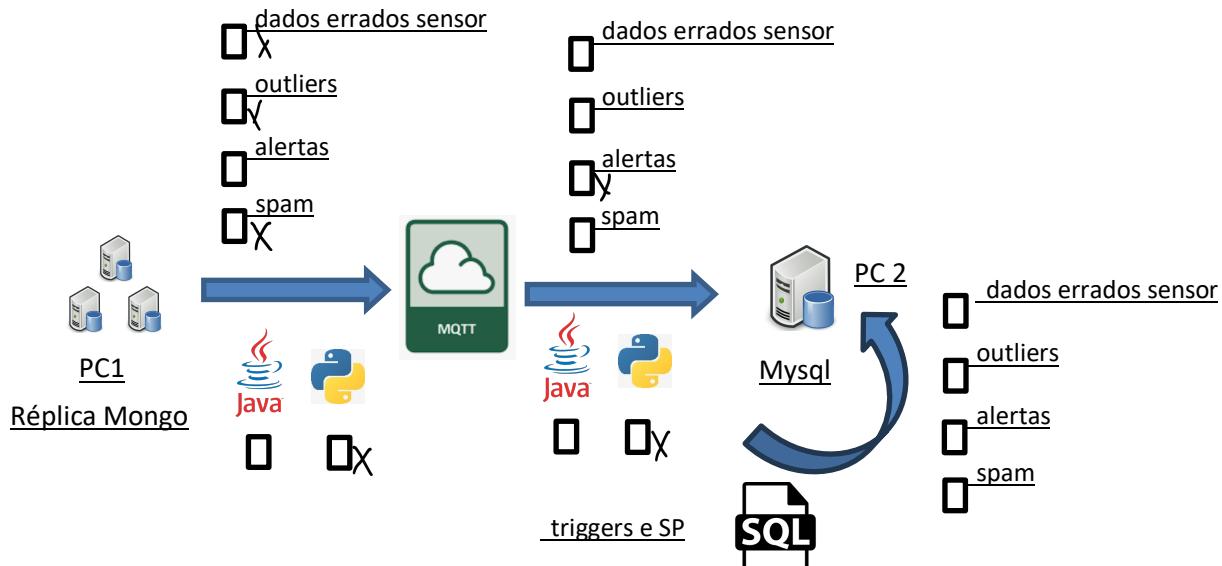
Coleção :

```
{ ... }
```



1.2 Descrição Geral do Procedimento de Mongo Para Mysql

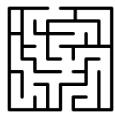
A passagem do Mongo para Mysql tem duas fases: a) enviar de Mongo para MQTT e b) receber de MQTT para Mysql. No diagrama deve ser indicado (nas check box) em qual das fases são programadas (em java e/ou python) as tarefas enumeradas (podem ser na fase a) b) ou ambas).



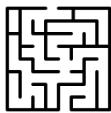
- Com que “periodicidade” (segundos) o programa vai buscar ao mongo: 2s
- Como garantem que não enviam duas vezes o mesmo documento do Mongo para o Mysql / MQTT (com base em datas ou booleano ou etc.):

Guardamos o ID do documento que foi enviado num ficheiro json, e depois na próxima iteração verificamos se esse documento está lá presente ou não.

- Pensam usar threads do Mongo para MQTT?: Não e do MQTT para Mysql Não?
- Quantas threads e/ou quantos maim em cada um dos passos?:
0



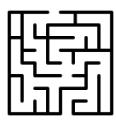
- No transporte MQTT que QOS vão utilizar? Porquê? Pretendemos utilizar o qos 2 porque este envia uma única vez a mensagem (neste caso, o documento) de modo a não precisar de fazer verificação de “duplicados” na inserção dos dados no MySql, pois este não iria receber o documento mais que uma vez (qos 1) e não utilizamos o qos 0 porque este não garante o sucesso na receção da mensagem o que significa pode haver documentos a não serem recebidos que achamos que irá influenciar na parte do algoritmo que vai “jogar” o jogo.

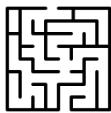


Aqui podem desenvolver informação que considerem relevante relativo ao processo de migração, aspectos que não esteja refletido nas secções seguintes.

No transporte para do Mongo para o MQTT, envia-se os documentos em batches, ou seja, conjuntos de documentos existentes no intervalo de 2 segundos, ou seja, a cada 2 segundos o programa deve ver que documentos é que existem no Mongo e enviar. Deve ser também implementado um mecanismo de modo a garantir que não se envia o mesmo documento para o recetor, que irá ser baseado em colocar os id's dos documentos que já foram enviados num txt e então verificar a cada iteração se o documento já fora enviado.

Isto foi uma alternativa que optámos ao envio de 1 documento de cada vez, e assim desta maneira conseguimos enviar informação dos movimentos e do ruído em conjunto, em vez de primeiro os movimentos todos e depois os ruídos. Como foi optado a escolha de utilizar QOS 2, se enviássemos uma mensagem de cada vez, o "tempo" gasto em handshake do QOS 2 seria maior do que se mandássemos em batches.





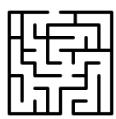
1.4 Tratamento de dados anómalos (valores de sensores errados)

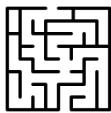
Aqui devem explicar o que fazer caso se detectem valores “errados” (datas impossíveis, caracteres estranhos, etc..). Se recorrerem a triggers ou SP então indicam em secção mais adiante.

Para verificar dados anómalos, será necessário definir um intervalo de valores possíveis de receber.

Por exemplo, som negativo, ID do player errado, pontos negativos, ... - são dados anómalos, ou seja, temos de verificar que estes dados não são enviados para o PC2

Temos de fazer verificações do que poderão ser valores impossíveis e fazer essa filtração antes de enviar os dados.





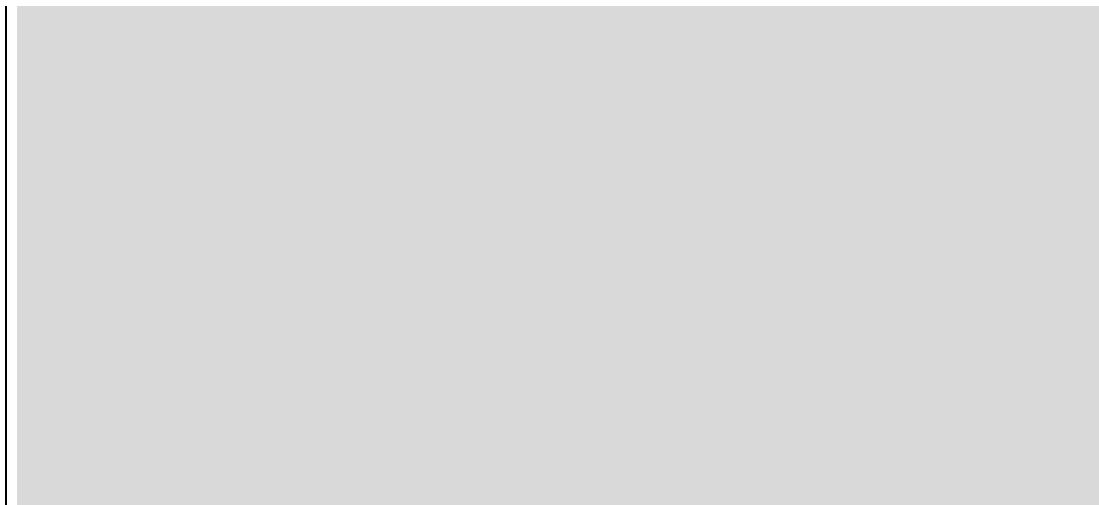
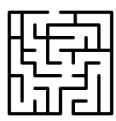
1.5 Tratamento de outliers de temperaturas

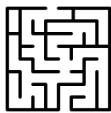
Aqui devem explicar o que fazer caso se detectem "outliers" (valores fisicamente possíveis mas irrealistas, como por exemplo variações muito bruscas do ruído apenas num segundo). Se recorrerem a triggers ou SP então indicam em secção mais adiante.

Para detetar outliers, teríamos de ter um intervalo de valores "normais" para receber. Para fazer tal, podemos utilizar o Método de Tukey que consiste dividir os valores de dados em quartis, e os valores, que tiverem dentro do 1º e 4º quartil são classificados como Outliers.

Caso se detetem outliers, consideramos que ao ser um valor possível, que talvez seja relevante, e por isso, não fazemos "nada" aos outliers, e deixamos que este seja enviado.

Mesmo sendo improvável consideremos que por ser possível este valor pode ser relevante por exemplo no caso em que todas as portas sejam fechadas/abertas.





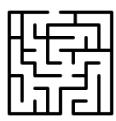
1.6 Tratamento de Alertas de Som

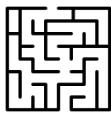
Aqui devem explicar o que fazer caso se detectem situações alarmantes relativos ao som. Se recorrerem a triggers ou SP então indicam em secção mais adiante. Quais situações despoletam os alertas, onde e como são armazenados. Explicar se existem mecanismos para evitar “spam” (demasiadas mensagens). É aconselhável recorrer a esquemas gráficos para explicar o mecanismo.

Os alertas são despoletados quando o nível de ruído / som ultrapassam thresholds de modo a avisar o jogador que está cada vez mais perto de perder o jogo.

Estes são armazenados na base de dados MySql, na tabela “Mensagens”.

Para evitar spam, implementámos uma maneira de enviar as mensagens de 2 em 2 segundos em forma de batches.





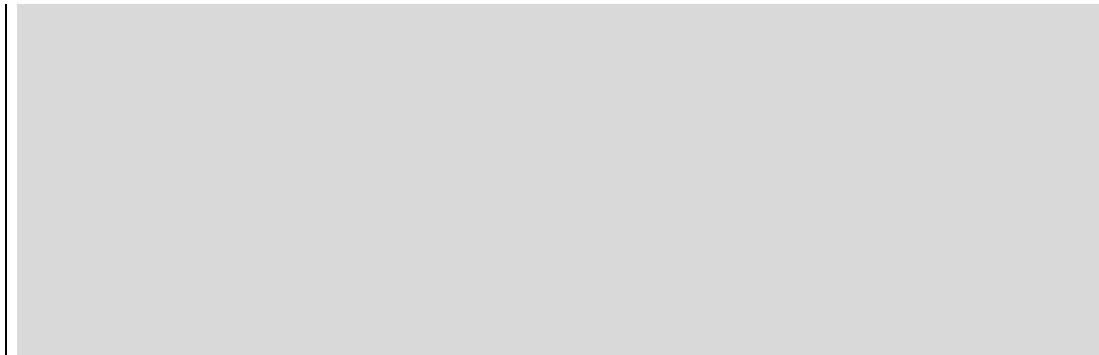
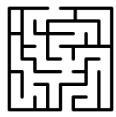
1.7 Tratamento de número de marsamis numa sala (obter pontuação)

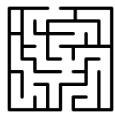
Aqui devem explicar como funciona o mecanismo que detecta que o número de marsamis odd é (ou vai ser) igual ao número de marsamis even. Como detecta, e o que desencadeia.

Quando o gatilho é acionado numa certa sala (o número de gatilhos teria de ser menor que 3), o número de gatilhos nessa sala é incrementado, depois é verificado a quantidade de marsamis odd e even, se a quantidade for igual, então os pontos nessa sala são incrementados, se o número de marsamis odd e even não estiverem equilibrados, a sala perde 0.5 pontos.

Para detectar estes acontecimentos temos de ter um algoritmo, e verificar em que salas os marsamis estão.

Tal algoritmo baseia-se na utilização de dois maps, sendo estes *mapMarsami*, que mapeia a sala em que cada Marsami se encontra (garantindo assim uma verificação extra nos movimentos dos Marsamis e aumentando a fiabilidade do algoritmo), e *mapSalas*, que guarda um tuplo da quantidade de Marsamis pares e ímpares em cada sala.

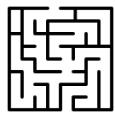




1.8 Especificação de Store Procedures SQL de apoio à migração e tratamento de dados

Nas secções anteriores foram descritos mecanismos que podem ou não necessitar de recorrer a Store Procedures. É nesta tabela que eles deverão ser listados. Na descrição apenas colocar informação que não seja óbvia.

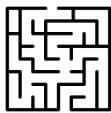
Nome SP	Argumentos	Muito breve descrição
...



1.9 Especificação de Triggers de apoio à migração e tratamento de dados

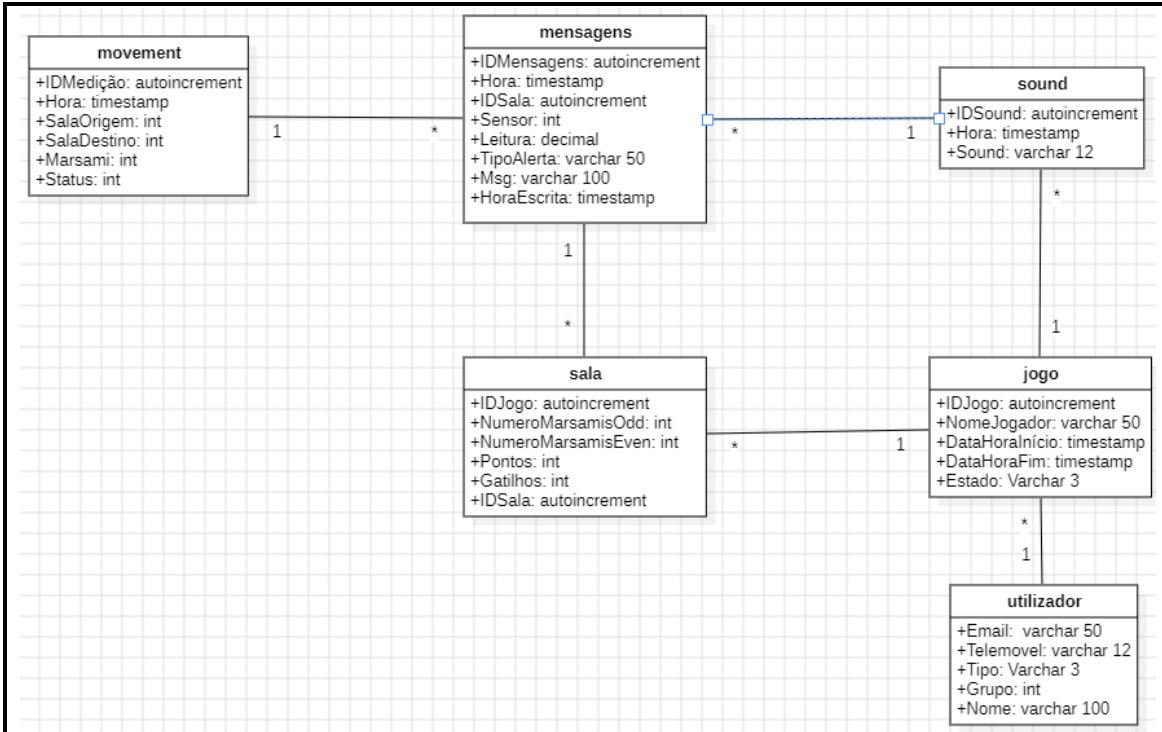
Nas secções anteriores foram descritos mecanismos que podem ou não necessitar de recorrer a Triggers. É nesta tabela que eles deverão ser listados. Nas Notas apenas colocar informação que não seja óbvia.

Nome Trigger	Tabela	Tipo de Operação (I,U,D)	Evento (After, Before)	Notas
'''				



1.10 Modelo Relacional

Diagrama relacional completo. Alterações à base de dados original terão de ser justificadas aqui. Caso seja pertinente podem ser adicionados comentários a justificar opções pouco óbvias.



Alterações:

Tabela Movement: Alteramos o nome da tabela "MediçõesPassagens" para "Movement".

Tabela Mensagens: Adicionamos como Chave Estrangeira o campo IDSala.

Tabela Sala: Alteramos o nome da tabela "OcupaçãoLabirinto" para "Sala".

Adicionamos como Chave Estrangeira o campo IDJogo e como Chave Primária o campo IDSala.

Criamos o campo Pontos, pois em cada sala é possível obter no máximo 3 pontos e foi adicionado também o campo Gatilhos para guardar a informação sobre o número de vezes que um gatilho foi acionado.

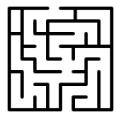
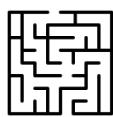


Tabela Jogo: Eliminamos o campo Descrição por ser irrelevante no contexto. Adicionamos o campo Estado que permite guardar o estado do jogo (ativo, finalizado, cancelado).

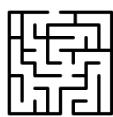


1.11 Utilizadores Base de Dados Mysql

Nesta secção deverá ser explicado de que forma deverá ser feita a manutenção de utilizadores Mysql. Nomeadamente deverá ser indicado, para cada tipo de utilizador, que privilégios ele tem sobre que tabelas e Stored Procedures (todos os SP usados na Aplicação)

Tabela	Tipo de Utilizador		
	Administrador	Utilizador	Software
mensagens	-	L	-
movement	-	-	L
sound	-	L	L
jogo	-	U/I/L	-
utilizador	I/D/L	-	-
sala	-	L	L
Stored Proc.			
SP (reiniciar o processo de migração)	X	-	-
SP (criar utilizador)	X	-	-
SP (remover utilizador)	X	-	-
SP (alterar utilizador)	X	-	-
SP (criar jogo)	-	X	-
SP (alterar jogo)	-	X	-

Em que U=Update, I Insert, D- Delete, L=Leitura, X=Executar SP e - sem permissões.



1.12 Procedimentos Manutenção da Aplicação

Nesta secção deverão ser listados os SP para a manutenção de utilizadores e jogos (apenas os obrigatórios)

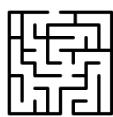
Nome SP	Argumentos	Muito breve descrição
Criar_utilizador	Email, tipo de utilizador.	Cria um utilizador com chave primaria igual ao email fornecido.
Remover_utilizador	Email do utilizador.	Eliminar o utilizador.
Alterar_utilizador	Email do utilizador.	Editar os dados do utilizador. Campos possíveis a alterar são: Nome, Telemóvel, Grupo.
Criar_jogo	ID Jogo, dataHoraInicio.	Cria um jogo.
Alterar_jogo	ID Jogo.	Alterar um jogo.



1.13 Eventos de suporte à aplicação (caso existam)

Deverão ser indicados todos os eventos relevantes para o processo de migração, eventos do Windows e do Mysql. Por eventos entende-se as tarefas do Windows ou eventos do Mysql

Nome Evento	Local Execução (Mysql/Windows)	Muito breve descrição



1.14 Consulta por HTML/PHP

Desenhar o layout dos formulários pretendidos, se relevante colocar texto a explicar a funcionalidade pretendida. Formulários:

Fazer login

Criar (ou selecionar de uma lista de jogos um para alterar) um jogo e, para esse jogo, editar os valores associados. Quando está a alterar não pode alterar valores de chaves estrangeiras e primárias.

Indicar para cada botão qual o SP que deverá ser executado

Log in

Host

User

Password

Database

Botão “Log in” – SP: Criar_utilizador, se utilizador não existir.

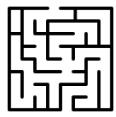
Selecione o jogo que pretende alterar:

Jogos

- Jogo 1
- Jogo 2
- Jogo 3
- Jogo 4
- Jogo 5

Botão “Continuar” - levar para a página de alteração.

Botão “Criar novo Jogo”- SP: Criar_jogo



Jogo _

Valor a alterar:

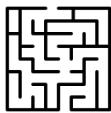
Nome do jogador:

Alterar jogo

Iniciar

Botão “Iniciar” – SP: php que chama executável.

Botão “Alterar jogo” – SP: Alterar_jogo



2 Implementação

*Esta secção é para ser preenchida pelo grupo que recebeu o documento. Aqui vão falar da implementação que fizeram. A implementação é o "best of", ou seja, o que **agora** acham que é a melhor solução, com base em tudo o que aprenderam (com as vossas experiências, com as ideias do outro grupo, discussões com o professor, google, chatgpt, etc), no limite podem não seguir nada do que tinham especificado no documento que entregaram ao outro grupo.*

2.1 Coleções a criar em cada uma das replicas do Mongo

Versão	Número Coleções
Especificação inicial	4
Recebida outro grupo	3
Implementada	4

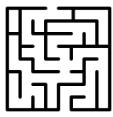
Justificação da escolha

Texto justificativo da opção final. O que for idêntico à inicial (enviada a outro grupo) não preencher

Para cada coleção **implementada** exemplifica um documento

Coleção : sensores movimento

```
{  
  "_id": {  
    "$oid": "67d94a4a0519f895186e6cc1"  
  },  
  "payload": {  
    "Player": 16,  
    "x": 100,  
    "y": 100  
  }  
}
```

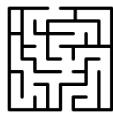


```
"Marsami": 15,  
"RoomOrigin": 3,  
"RoomDestiny": 2,  
"Status": 1  
}  
"processed": true  
,  
"topic": "pisid_mazemov_16"  
}
```

Coleção : sensores_ruido

```
{  
  "_id": {  
    "$oid": "67d85937e955024c65add6be"  
  },  
  "payload": {  
    "Player": 16,  
    "Hour": "2025-03-17 17:17:11.168077",  
    "Sound": 19.04  
  },  
  "topic": "pisid_mazesound_16"  
}
```

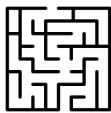
Coleção : dados_invalidos



```
{  
    "tipo": "som" ou "movimento",  
    "payload": {  
        conteúdo original recebido via MQTT  
    },  
    "erro": "Descrição do erro",  
    "timestamp": ISODate("AAAA-MM-DDTHH:MM:SSZ")  
}
```

Coleção : game_configs

```
{  
    "_id": ObjectId("..."),  
    "maxSoundLevel": 75.0,  
    "soundVariationLimit": 15.0,  
    "roomsConfig": [  
        {"roomId": 1, "connectedTo": [2, 3]},  
        {"roomId": 2, "connectedTo": [1, 4]},  
        {"roomId": 3, "connectedTo": [1, 5]},  
        {"roomId": 4, "connectedTo": [2, 5]},  
        {"roomId": 5, "connectedTo": [3, 4]}  
    ]  
}
```



2.2 Descrição Geral do Procedimento de Mongo Para Mysql

Na checkbox da esquerda indicam o que especificaram inicialmente, na do meio a especificação do outro grupo e na da direita a vossa implementação.

Justificação da escolha

Aqui, tal como explicamos mais à frente, passamos a tratar dos alertas de ruído e spam de mensagens antes de inserir para o MySQL.

Versão	Periodicidade vai buscar Mongo	Como evitam enviar duas vezes para MQTT	Número Threads	QOS
Especificação inicial	2s	campo “processed”	0	QOS1
Recebida outro grupo	2s	id do documento em ficheiro json	0	QOS2
<u>Implementada</u>	2s	campo “processed”	0	QOS1

Justificação da escolha



Texto justificativo da opção final. O que for idêntico à inicial (enviada a outro grupo) não preencher



Nas próximas quatro secções devem, na descrição, resumidamente descrever num texto escorreito e legível, a forma como foi implementada. Têm de ficar muito explicitamente indicado a o que não resultou da especificação inicial (cor preta), o que foi aproveitado da especificação que receberam de outro grupo cor azul) e o que resultou de ideias posteriores vossas (cor verde). Na justificação sejam muito objectivos a explicar a razão de terem alterado a especificação inicial

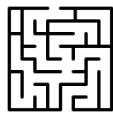
Por exemplo (não necessariamente correcto), para deteção de valores anómalos:

1 Descrição

Tratámos dados anómalos resultantes de formatos de datas inválidos (por exemplo, meses com mais de 31 dias), e também considerámos sons negativos como anómalos. Os dados anómalos foram assinalados na coleção como anómalos em vez de serem apagados

2 Justificação das alterações caso tenham havido

Tínhamo-nos esquecido dos valores negativos do som e achamos que pode ser útil alguém querer consultar os valores anómalos posteriormente.

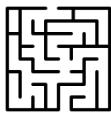


2.3 Tratamento de dados anómalos (valores de sensores errados)

1 Descrição

Mantivemos a abordagem definida na nossa especificação inicial, implementando a deteção de dados anómalos na fase de passagem do MongoDB para o MQTT. Sempre que são detectados valores incorretos — como mensagens mal formatadas, dados incompletos ou datas inválidas (por exemplo, valores negativos de som ou RoomOrigin nulo) — esses documentos são automaticamente classificados como inválidos e movidos para a coleção `dados_invalidos` no MongoDB. Cada entrada nessa coleção inclui o tipo de erro, o payload original e um timestamp, permitindo rastreabilidade e possível análise futura.

2 Justificação das alterações caso tenham havido



2.4 Tratamento de outliers de temperaturas

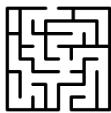
1 Descrição

O tratamento de outliers de ruído foi feito na fase de receção dos dados via MQTT, antes da sua inserção na base de dados MySQL. Foi definido um critério simples e eficaz: se o valor de som recebido for superior a 75% do valor anterior (i.e., uma variação abrupta), esse valor é considerado um outlier e é ignorado.

Esta abordagem permite filtrar valores que, embora possíveis, são altamente improváveis no contexto do jogo e poderiam despoletar alertas falsos ou distorcer a análise do ruído no labirinto.

2 Justificação das alterações caso tenham havido

A nossa especificação inicial previa o uso de um intervalo máximo de variação configurável por jogo e também a filtragem de valores redundantes em intervalos de tempo curtos. No entanto, durante a implementação, simplificamos a lógica adotando apenas a verificação de variações superiores a 75% em relação ao valor anterior, o que nos permitiu manter o sistema mais leve e eficiente. Decidimos também não aplicar a filtragem por redundância, por considerarmos que não tinha impacto significativo no desempenho ou na integridade da base de dados.



2.5 Tratamento de Alertas de Som

1 Descrição

Na implementação final, mantivemos a ideia de três níveis de alerta, mas optámos por uma abordagem mais dinâmica em vez de utilizar percentagens fixas do valor máximo. Níveis de alerta:

1 - alerta amarelo: quando o valor de som ultrapassa `normalNoise + variacaoMax * 0.5`

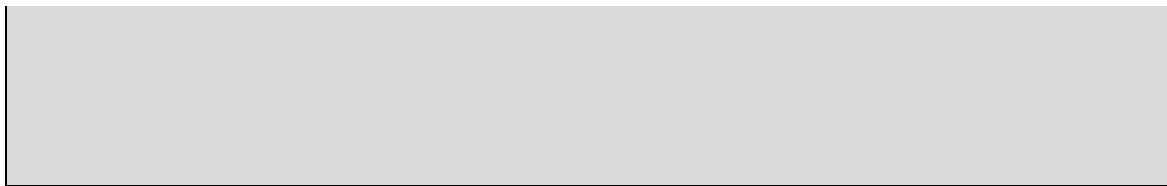
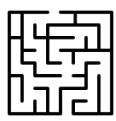
2 - alerta vermelho: quando o valor de som ultrapassa `normalNoise + variacaoMax * 0.8`

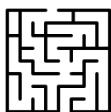
3 - alerta de limite Máximo: quando o valor de som ultrapassa o `maxSound` definido para o jogo

A verificação é feita no script de MQTT para MySQL. Se um desses limites for ultrapassado, é criada uma entrada na tabela mensagens com o tipo de alerta correspondente. Para evitar spam, apenas um alerta do mesmo tipo pode ser enviado para o mesmo jogador dentro de um intervalo de 10 segundos.

2 Justificação das alterações caso tenham havido

Na especificação inicial, os níveis de alerta eram fixos (70%, 85% e 100% do ruído máximo). No entanto, durante a implementação decidimos tornar os thresholds mais adaptáveis ao contexto de cada jogo, utilizando `normalNoise` e `variacaoMax`, que são valores configuráveis por jogo. Isto tornou os alertas mais coerentes com a configuração real do labirinto, melhorando a sensibilidade e precisão do sistema. Esta abordagem também se mostrou mais robusta do que a do Grupo 9, que só previa um único limiar de som.





2.6 Tratamento de número de marsamis numa sala (obter pontuação)

1 Descrição

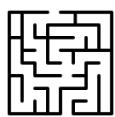
Para otimizar o controlo do número de marsamis "odd" e "even" em cada sala, decidimos modificar a abordagem inicialmente planeada. A nossa ideia inicial consistia em utilizar um trigger que a cada atualização da tabela OcupacaoLabirinto numa sala, verificava automaticamente o equilíbrio de marsamis e, caso houvesse, criava uma nova mensagem (com um trigger) que seria lida pelo bot da tabela Mensagem. No entanto, após começarmos a implementar, optámos por uma abordagem mais direta e eficiente uma vez que as mensagens enviadas eram muitas, misturavam-se com os alertas de ruído e não conseguíamos tomar decisões eficientes para jogadas.

Na primeira fase também não tínhamos pensado bem na logística do número de gatilhos limite por sala e durante a implementação optámos por guardar esse número de gatilhos num mapaGatilhos apenas no próprio programa que joga o jogo (bot) e que o próprio vai atualizando conforme as suas jogadas. O bot só vai verificar o equilíbrio nas salas que não tenham já atingido o limite de gatilhos (3). Não vimos necessidade de guardar o número de gatilhos na base de dados como foi sugerido pelo outro grupo uma vez que a consulta é mais rápida e direta em memória e evita fazer updates e mais comparações na base de dados.

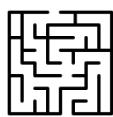
Agora, o bot acessa diretamente a tabela OcupacaoLabirinto, controlando continuamente as ocupações por sala para poder identificar se o número de marsamis "odd" e "even" em alguma sala está igual. Se houver equilíbrio, o bot aciona o gatilho de pontuação (score_trigger), garantindo assim que a pontuação seja atualizada conforme o esperado. Para obter a pontuação temos também uma função que a retorna (fornecida pelo professor) e que permite ao jogador consultar os pontos por cada sala.

2 Justificação das alterações caso tenham havido

Optámos por esta implementação por três razões que trazem vantagem para o nosso projeto:



- 1 - Reduzimos o número de acesso à base de dados uma vez que não é necessário atualizar o número de gatilhos na tabela e evitando também dar um acesso desnecessário ao perfil de jogador para fazer essa atualização.
- 2- O bot vai possuir controlo direto sobre a lógica de verificação de equilíbrio e o acionamento do gatilho, não dependendo da leitura de mensagens e aumentando a rapidez das decisões.
- 3 - Dá maior flexibilidade, pois podemos fazer ajustes rápidos sem entrar na lógica de modificar estruturas complexas na base de dados.



2.7 Implementação de Stored Procedures SQL de apoio à migração e tratamento de dados

É nesta tabela que deverão ser listados os SP que implementam mecanismos anteriores. Na quarta coluna têm de colocar um dos seguintes símbolos:

= - igual ao especificado

A(og) – Alterado com base em ideia de outro grupo

A(ni) – Alterado com base em novas ideias

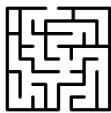
N(og) - Novo com base em ideia de outro grupo

N(ni) – Novo com base em ideia novas

A – Alterado com base em novas ideias

Nome SP	Argumentos	Descrição	
...	

Texto justificativo da opção final. O que for idêntico à inicial (enviada a outro grupo) não preencher



2.8 Implementação de Triggers

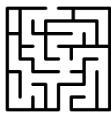
É nesta tabela que deverão ser listados os triggers. Na sexta coluna têm de colocar um dos seguintes símbolos:

- = - igual ao especificado
- A(og) - Alterado com base em ideia de outro grupo
- A(ni) - Alterado com base em novas ideias
- N(og) - Novo com base em ideia de outro grupo
- N(ni) - Novo com base em ideia novas
- A - Alterado com base em novas ideias

				da sala origem e da sala destino, sempre que recebe uma medicao de passagem	

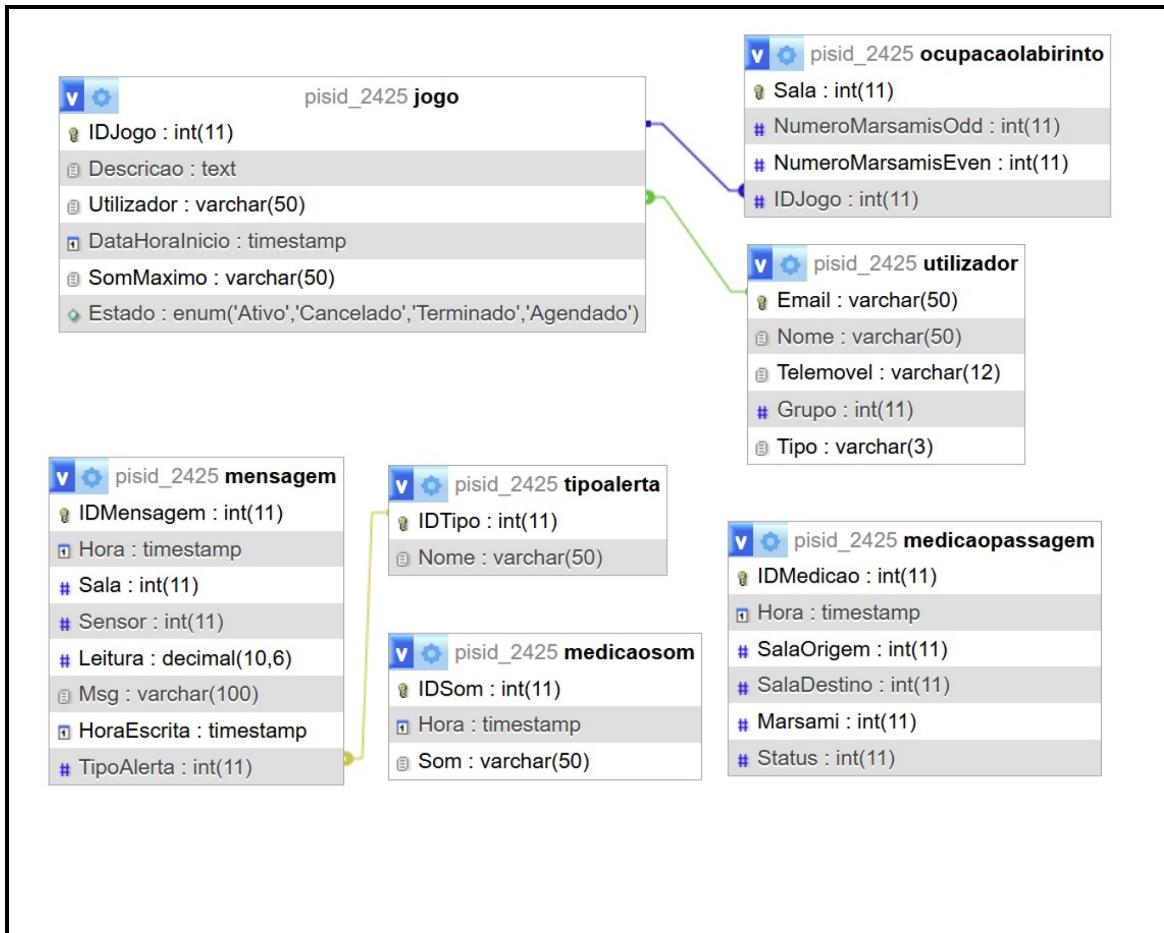
Em relação aos triggers, o grupo da especificação não tinha nenhum trigger e na nossa implementação decidimos usar. Optámos por modificar também a nossa abordagem inicial onde tínhamos planeado utilizar três triggers: (atualizar_ocupacao_movimento, verificar_equilibrio_marsamis e alerta_ruido) e decidimos manter apenas o trigger de atualizar_ocupacao_movimento.

Esta decisão está relacionada com aquilo que é explicado nas secções 2.5 e 2.6. Em relação ao alerta_ruido, como referimos na secção 2.5 tínhamos pensado gerar as mensagens com base em triggers, mas durante a implementação percebemos que era mais fácil e eficiente dispararmos os alertas de ruído diretamente no script que envia os dados do mqtt para o mysql onde conseguimos gerir melhor a gestão do spam de mensagens. O verificar_ruido_marsamis foi descartado também com base nas novas alterações explicadas na secção 2.6 sobre a forma como o bot interage e toma decisões no jogo.

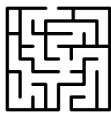


1.1 Modelo Relacional

Diagrama relacional completo implementado. Assinalar a azul alterações derivadas de outro grupo e a verde alterações novas.



Para o modelo relacional decidimos seguir uma implementação que foi mais de encontro ao que nós tínhamos especificado do que à especificação do outro grupo. A maioria das ligações das tabelas não percebemos ao certo qual seria a intenção do grupo e por isso optámos por seguir a nossa lógica. Também no nosso modelo alterámos coisas, as tabelas de medições (MedicaoPassagem e MedicaoSom) e de Mensagem deixaram de estar ligadas à tabela de Jogo. Não vimos necessidade de o fazer uma vez que tínhamos pensado numa logística em que todo o modelo era gerido por um IDJogo mas percebemos que não precisávamos de fazer essa logística. Acrescentamos o campo “Estado” sugerido pelo outro grupo no entanto os nossos estados não são exatamente os mesmos (passaram a ser “Ativo”, “Terminado”, “Agendado” e “Cancelado”) para ir de encontro



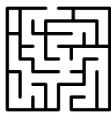
às novas implementações (explicadas nas outras secções). Não adicionámos nem o campo de “Gatilhos” nem de “Pontos” também pelos motivos que explicámos na secção 2.6. As restantes alterações são eventuais diferenças em nomes de tabelas e de campos mas que seguem a mesma lógica anterior.

Adicionamos ainda uma tabela *TipoAlerta* que nos fez sentido tendo em conta os nossos 3 tipos de alerta de ruído (“Amarelo”, “Vermelho” e “Limite Atingido” explicados anteriormente). Cada tipo tem um *IDAlerta* que é associado às *Mensagens* geradas.

1.2 Utilizadores Base de Dados Mysql

Nesta secção deverão ser indicados os utilizadores e perfis implementados (têm de constar todos os SP usados). Assinalar a azul alterações derivadas de outro grupo e a verde alterações novas.

Tabela	Tipo de Utilizador		
	Administrador	Jogador	Migrador
Jogo	I/U/D/L	L	U/L
MedicaoSom	L	L	I
MedicaoPas sagem	L	L	I
OcupacaoLa birinto	I/U/D/L	L	-
Utilizador	I/U/D/L	-	-
Mensagem	L	L	I
TipoAlerta	I/U/D/L	L	L
Stored Proc.			

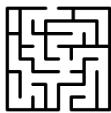


CriarUtilizador	X	-	-
EditarUtilizador	X	X	-
RemoverUtilizador	X	-	-
CriarJogo	X	X	-
IniciarJogo	X	X	-
EditarJogo	X	X	-
CancelarJogo	X	X	-

Em que U=Update, I Insert, D- Delete, L=Leitura, X=Executar SP e - sem permissões.

Na parte dos utilizadores e privilégios, a nossa solução é bastante diferente tanto da sugerida pelo grupo como da que tínhamos definido inicialmente. Considerámos as permissões que recebemos na especificação demasiado restritas (por exemplo o administrador n tinha permissões em nenhuma tabela tirando na de Utilizador) e optámos por seguir uma lógica diferente.

Em discussões com o professor nas reuniões chegámos à conclusão de que podem haver erros no nosso sistema e que é sempre bom o Administrador conseguir alterar campos e tabelas de forma a reagir a esses erros. Seguindo esta lógica, decidimos dar ao Administrador todas as permissões nas tabelas de Jogo, Utilizador, OcupacaoLabirinto e Tipo Alerta. Deste modo, havendo algum problema com os dados, o Administrador tem autonomia para os resolver. Em relação ás tabelas de medições (MedicaoPassagem e MedicaoSom) e de Mensagem decidimos não dar estas permissões (apenas tem permissões de leitura) porque interpretamos estes registo como dados temporários que vão sendo “reciclados” e que apenas têm importância para o momento em que são feitas as inserções por isso não vimos vantagens em dar essas permissões.



Para o Jogador (Investigador na especificação) apenas tem permissões de leitura nas tabelas (tirando a tabela de utilizadores como o grupo especificou) uma vez que todas as ações que ele tem sobre as tabelas vão ser geridas pelas Stored Procedures que temos garantido que conseguimos dar-lhe as permissões específicas que queremos. A nossa lógica foi criar SPs que tanto o jogador como o administrador têm permissões para executar e fazer a validação dentro da própria SP dos pormenores respectivos a cada um.

No caso do Migrador (Sistema na especificação) na nossa implementação decidimos permitir ao Migrador fazer updates na tabela de jogo para poder alterar o estado do jogo (é no script migrador que obtemos a informação de jogo terminado) e também para poder indicar na tabela de Jogo qual o valor do SomMaximo do jogo que está a decorrer (que vai buscar à nuvem). Isto facilitou-nos o processo de gestão dos estados do jogo. O Migrador não tem permissões de execução sobre nenhuma SP.

1.3 Procedimentos Manutenção da Aplicação

É nesta tabela que deverão ser listados os SP que implementam mecanismos anteriores. Na quarta coluna têm de colocar um dos seguintes símbolos:

= - igual ao especificado

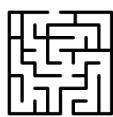
A(og) - Alterado com base em ideia de outro grupo

A(ni) - Alterado com base em novas ideias

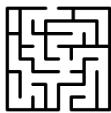
N(og) - Novo com base em ideia de outro grupo

N(ni) - Novo com base em ideia novas

A - Alterado com base em novas ideias



Nome SP	Argumentos	Descrição	
CriarUtilizador	p_dados JSON (inclui o email, nome, telemóvel tipo e password)	Insere os dados retirados do JSON (alguns campos são obrigatórios como o email e a password) na tabela Utilizador. São atribuídos os Roles ao tipo respetivo	A(ni)
EditarUtilizador	p_dados JSON (inclui o email, nome, telemovel e password)	Edita os dados de um utilizador (opcionais).	A(ni)
RemoverUtilizador	email	Remove um utilizador e todos os jogos associados	=
CriarJogo	p_dados JSON (inclui o email, descricao, dataHoraInicio)	No caso de ser o Jogador a chamar só pode criar um jogo para si, no caso de ser administrador cria para outro jogador (email parâmetro). Devolve o Id do jogo criado.	A(ni)
IniciarJogo	idJogo	Inicia um jogo (passa para o estado "Ativo") se não houver mais nenhuma a decorrer	N(ni)
EditarJogo	p_dados JSON (inclui o	Edita um jogo com os dados fornecidos	A(ni)

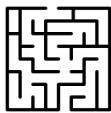


	idJogo, descricao, dataHoraInicio , estado)	tendo em conta as validações referidas mais à frente	
Cancela rJogo	idJogo	Cancela um jogo (muda o estado para “Cancelado”)	N(ni)

A principal alteração que fizemos nesta secção foi em relação aos argumentos de entrada das nossas SPS. Em discussão nas reuniões com o professor, chegámos à conclusão que não era muito prático termos que chamar as SPS sempre com todos os argumentos mesmo que não precisássemos deles (por exemplo no EditarJogo se só quiséssemos alterar a descrição teríamos que por os outros campos todos a null). Desta forma, alterámos os argumentos de entrada para o formato JSON (nas SPs indicadas) onde conseguimos indicar exatamente quais os campos que queremos editar e as validações dos dados obrigatórios são feitas na mesma nas próprias SPs.

Adicionámos também 2 SPs novas, uma para iniciar um jogo (IniciarJogo) e outra para cancelar (CancelarJogo). Estas SPs estão relacionadas com a alteração do estado do jogo uma vez que com a nossa inovação de ter jogos no estado “Agendado” precisámos de criar novas SPs para poder gerir esse estado. O CriarJogo neste caso passou a ser apenas uma criação de registo de jogo, não o inicia.

Também fizemos pequenas alterações em relação a campos editáveis que está relacionado com o que foi explicado na secção acima em relação ao tipo de utilizador é que está a executar a SP.



1.4 Eventos de suporte à aplicação (caso existam)

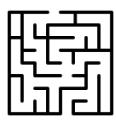
É nesta tabela que eles deverão ser listados todos os eventos relevantes para o processo de migração Na quarta coluna têm de colocar um dos seguintes símbolos:

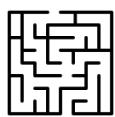
- = - igual ao especificado
- A(og) - Alterado com base em ideia de outro grupo
- A(ni) - Alterado com base em novas ideias
- N(og) - Novo com base em ideia de outro grupo
- N(ni) - Novo com base em ideia novas
- A - Alterado com base em novas ideias

Nome Evento	Local Execução (Mysql/Windows)	Muito breve descrição
LimpardadosTemporarios	MySQL	A cada 24 horas, apagamos todas as medições das tabelas de MedicaoSom, MedicaoPassagem e Mensagem em que a hora seja pelo menos 24 horas antes da hora atual
CorrerJogos	Windows	Programa que consulta a tabela de Jogos ciclicamente e vê se há algum jogo no estado "Agendado" para iniciar.

O LimpardadosTemporarios foi uma inovação que implementámos tanto na nossa especificação como na do outro grupo. Tomamos esta decisão porque mais uma vez consideramos estes dados de registo "temporário" e não vemos vantagens em acumular e sobreregar a base de dados com muitas medições.

Em relação ao CorrerJogos, está relacionado com a inovação de termos jogos agendados. Criámos um programa para gerir os jogos agendados e poder iniciá-los uma vez que não conseguíamos correr o executável através a partir do MySQL. Deste modo, se for possível (verificado através da SP IniciarJogo), inicia-se o jogo (altera pra modo "Ativo") e corremos o mazerun





1.5 PrintScreen dos formulários HTML implementados

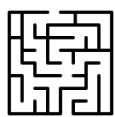
Nos ecrãs que vamos mostrar a seguir, não englobamos todas as opções de mensagens de erro que temos porque seriam bastantes formulários, mas tentamos representar exemplos da nossa lógica de implementação.

The screenshot shows a web-based login interface. At the top, a header reads "Bem-vindo ao sistema do Labirinto". Below it, a sub-header says "Login do Jogador". The main area contains a form with the following fields and controls:

- Email:
- Password: Mostrar password
-

Figura 1 - Ecrã de Login

- SPS: sem SPS (não utiliza SPS faz autenticação pelas credenciais no MySQL)



PI de Sistemas de Informação Distribuídos, 23/24

Página inicial

Criar novo Jogo

Os seus Jogos

Editar perfil

Logout

Criar Novo Jogo

Descrição:
Jogo novo para teste

Iniciar Agora
 Agendar

Data de Início:
24/05/2025, 21:57

Criar Jogo

Jogo agendado com sucesso para dia 24/05/2025 às 21:57

Figura 2- Ecrã de Criação de novo jogo (agendar para uma data)

- SPS: CriarJogo (com msg de sucesso) - **Botão Criar Jogo**

Página inicial

Criar novo Jogo

Os seus Jogos

Editar perfil

Logout

Criar Novo Jogo

Descrição:
Insira uma descrição para o jogo...

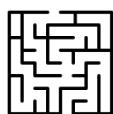
Iniciar Agora
 Agendar

Criar Jogo

Jogo iniciado com sucesso

Figura 3-Ecrã de Criação de novo Jogo (iniciar agora)

- SPS: CriarJogo + IniciarJogo - **Botão Criar Jogo**



PI de Sistemas de Informação Distribuídos, 23/24

Página inicial

Criar novo Jogo

Os seus Jogos

Editar perfil

Logout

Bem-vindo ao Labirinto dos Marsamis

Este é o teu centro de controlo.

- Criar novo Jogo: inicia ou agenda um novo jogo no labirinto de marsamis.
- Os seus Jogos: consulta todos os jogos que já foram criados por ti.
- Editar perfil: atualiza as informações do teu perfil.

Utiliza o menu à esquerda para navegar pelas funcionalidades disponíveis.

Dica: inicia um novo jogo para veres os marsamis em ação!

Figura 4 - Ecrã da Página Inicial

- SPS: sem SPS

Página inicial

Criar novo Jogo

Os seus Jogos

Editar perfil

Logout

Os Meus Jogos

ID	Descrição	Data	Estado	Ações
300	o meu jogo	2025-05-10 22:58:03	Agendado	Editar Iniciar Cancelar
299	teste	2025-05-06 22:22:03	Cancelado	Editar Iniciar Cancelar
296	Sem descrição	2025-05-10 23:35:03	Agendado	Editar Iniciar Cancelar
294	Jogo importante	2025-05-10 22:13:03	Cancelado	Editar Iniciar Cancelar
293	Sem descrição	2025-05-10 12:22:03	Cancelado	Editar Iniciar Cancelar

Terminados

ID	Descrição	Data	Ações
298	DESCRICAO VAZIA	2025-05-10 21:26:03	Editar
297	Sem desc	2025-05-04 22:22:03	Editar
295	Sem descrição	2025-05-02 22:22:03	Editar

Figura 5 - Ecrã de Listar os Jogos

- SPS: EditarJogo (**Botão Editar**); IniciarJogo (**Botão Iniciar**); CancelarJogo (**Botão Cancelar**)

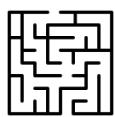


Figura 6 - Ecrã de Editar Perfil

- SPS: EditarUtilizador (**Botão Guardar Alterações**)

1.6 PrintScreen do formulários Android com dados

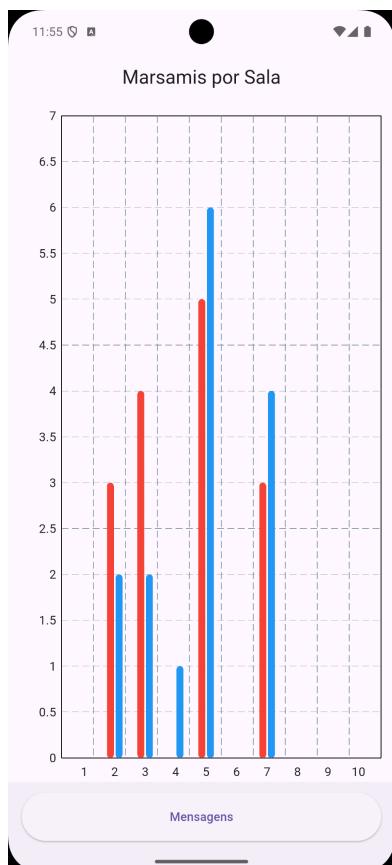
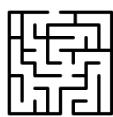


Figura 7- Dist marsamis por sala

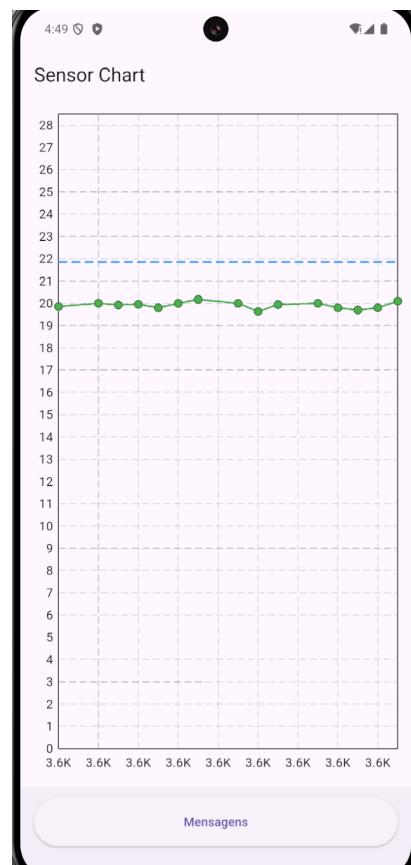
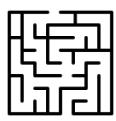


Figura 8 - Ruído do labirinto

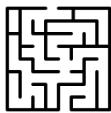
A table titled "Mensagens" showing player messages. The columns are "Msg", "Leitura", and "Sala". The data is as follows:

Msg	Leitura	Sala
ALERTA VERMELHO	19	1
ALERTA VERMELHO	19	1
ALERTA AMARELO	18	1
ALERTA VERMELHO	19	1
ALERTA AMARELO	18	1
ALERTA VERMELHO	19	1
ALERTA AMARELO	19	1
ALERTA AMARELO	19	1
ALERTA VERMELHO	19	1
ALERTA RUIDO MÁXIMO ATINGIDO	20	1
ALERTA AMARELO	18	1

Figura 9 - Lista de mensagens do jogador



Anexo Código SQL



Código de Triggers implementados

1. Nome Trigger: atualizar_ocupacao_movimento

Este trigger a cada entrada na tabela de MedicaoPassagem atualiza a tabela de OcupacaoLabirinto tendo em conta a movimentação do marsami. Para garantir que no caso de serem perdidas medições pelo meio, o número de marsamis final é o correto, o marsami é retirado da última sala onde há registo de ter sido inserido pela última medição. A restante explicação é feita em comentários ao longo do código.

CÓDIGO:

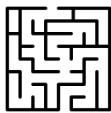
```
CREATE OR REPLACE DEFINER=root@localhost TRIGGER
atualizar_ocupacao_movimento
AFTER INSERT ON MedicaoPassagem
FOR EACH ROW
BEGIN

    DECLARE marsami_paridade INT;
    DECLARE id_jogo INT;
    DECLARE sala_dst_anterior INT;
    DECLARE sala_ori_anterior int;
    DECLARE countSala INT;

    IF NEW.Status = 1 THEN
        SET marsami_paridade = NEW.Marsami % 2;

        -- Obtém ID do jogo ativo
        SELECT IDJogo INTO id_jogo
        FROM Jogo
        WHERE Estado = 'ativo'
        LIMIT 1;

        -- Última sala conhecida do Marsami
        SELECT salaDestino , salaOrigem INTO
sala_dst_anterior , sala_ori_anterior
        FROM MedicaoPassagem
        WHERE Marsami = NEW.Marsami
            AND Status = 1
            AND salaDestino <> 0
            AND IDMedicao =
                (SELECT MAX(IDMedicao)
                FROM MedicaoPassagem
```



```
WHERE Marsami = NEW.Marsami
    AND Status = 1
    AND salaDestino <> 0
    AND IDMedicao < NEW.IDMedicao
);

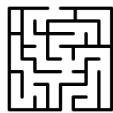
-- Passamos a ignorar a informação da sala de origem,
uma vez que a unica coisa que interessa é onde estava o
marsami na ultima medição
-- para o retirar de lá. (para o caso de perder
medições)

-- so insere salas <>0
IF sala_dst_anterior is not null and sala_dst_anterior
<> 0 THEN
    -- Verifica e insere sala anterior se necessário
    SELECT COUNT(*) INTO countSala
    FROM OcupacaoLabirinto
    WHERE Sala = sala_dst_anterior;

    IF countSala = 0 THEN
        INSERT INTO OcupacaoLabirinto (Sala,
        NumeroMarsamisOdd, NumeroMarsamisEven, IDJogo)
        VALUES (sala_dst_anterior, 0, 0, id_jogo);
    END IF;
end if;
-- Verifica e insere sala destino se necessário
IF NEW.SalaDestino <> 0 THEN
    SELECT COUNT(*) INTO countSala
    FROM OcupacaoLabirinto
    WHERE Sala = NEW.SalaDestino;

    IF countSala = 0 THEN
        INSERT INTO OcupacaoLabirinto (Sala,
        NumeroMarsamisOdd, NumeroMarsamisEven, IDJogo)
        VALUES (NEW.SalaDestino, 0, 0, id_jogo);
    END IF;
END IF;

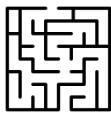
-- Atualiza a sala anterior (subtrai 1)
IF sala_dst_anterior is not null and
sala_dst_anterior <> 0 THEN
    UPDATE OcupacaoLabirinto
    SET
        NumeroMarsamisOdd = NumeroMarsamisOdd -
IF(marsami_paridade = 1, 1, 0),
        NumeroMarsamisEven = NumeroMarsamisEven -
IF(marsami_paridade = 0, 1, 0)
```



```
        WHERE Sala = sala_dst_anterior AND IDJogo =
id_jogo;
    END IF;

    -- Atualiza a nova sala (soma 1)
    IF NEW.saladestino <> 0 THEN
        UPDATE OcupacaoLabirinto
        SET
            NumeroMarsamisOdd = NumeroMarsamisOdd +
IF(marsami_paridade = 1, 1, 0),
            NumeroMarsamisEven = NumeroMarsamisEven +
IF(marsami_paridade = 0, 1, 0)
        WHERE Sala = NEW.saladestino AND IDJogo = id_jogo;
    END IF;
END IF;

END
```



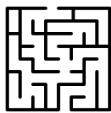
Código Stored Procedures implementados

1. Nome SP: CriarUtilizador

Esta SP recebe um JSON com os dados de um utilizador, valida os campos obrigatórios, insere o utilizador na tabela Utilizador, e cria o utilizador com as credenciais de acesso atribui-lhe o role apropriado com base no tipo (Jog -> Jogador; Adm -> AdminApp; Mig -> Migrador). Os pormenores das validações estão comentados ao longo do código.

CÓDIGO:

```
DELIMITER $$  
CREATE DEFINER=`root`@`localhost` PROCEDURE  
`CriarUtilizador`(IN `p_dados` JSON)  
BEGIN  
    DECLARE v_email VARCHAR(50);  
    DECLARE v_nome VARCHAR(100);  
    DECLARE v_telemovel VARCHAR(12);  
    DECLARE v_tipo VARCHAR(3);  
    DECLARE v_pass VARCHAR(50);  
  
    -- Extrair os campos diretamente  
    SET v_email = JSON_UNQUOTE(JSON_EXTRACT(p_dados,  
'$.Email'));  
    SET v_nome = JSON_UNQUOTE(JSON_EXTRACT(p_dados,  
'$.Nome'));  
    SET v_telemovel = JSON_UNQUOTE(JSON_EXTRACT(p_dados,  
'$.Telemovel'));  
    SET v_tipo = JSON_UNQUOTE(JSON_EXTRACT(p_dados,  
'$.Tipo'));  
    SET v_pass = JSON_UNQUOTE(JSON_EXTRACT(p_dados,  
'$.Password'));  
  
    -- Validações  
    IF v_email IS NULL OR TRIM(v_email) = '' THEN  
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Campo  
"Email" é obrigatório';  
    END IF;  
  
    IF v_email NOT LIKE '%@%' THEN  
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Email  
inválido';  
    END IF;
```



```
IF v_pass IS NULL OR TRIM(v_pass) = '' THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT =
'Password não pode ser vazia';
END IF;

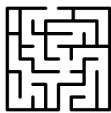
IF v_tipo IS NULL OR NOT v_tipo IN ('Jog', 'Adm',
'Mig') THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Tipo
inválido. Valores válidos: "Jog", "Adm", "Mig"';
END IF;

-- Inserção com grupo fixo 16
INSERT INTO Utilizador>Email, Nome, Telemovel, Grupo,
Tipo)
VALUES (
    v_email,
    v_nome,
    v_telemovel,
    16,
    v_tipo
);

-- Criação do utilizador MySQL
SET @Create = CONCAT('CREATE USER ''', v_email, ''
IDENTIFIED BY '', TRIM(v_pass), '';');
PREPARE stmt FROM @Create;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

-- Atribuição de roles
IF v_tipo = 'Jog' THEN
    SET @q1 = CONCAT('GRANT Jogador TO ''', v_email,
'';');
    SET @q2 = CONCAT('SET DEFAULT ROLE Jogador FOR
'', v_email, '';');
ELSEIF v_tipo = 'Adm' THEN
    SET @q1 = CONCAT('GRANT AdminApp TO ''', v_email,
'';');
    SET @q2 = CONCAT('SET DEFAULT ROLE AdminApp FOR
'', v_email, '';');
ELSE
    SET @q1 = CONCAT('GRANT Migrador TO ''', v_email,
'';');
    SET @q2 = CONCAT('SET DEFAULT ROLE Migrador FOR
'', v_email, '';');
END IF;

-- Executar atribuições
```



```
PREPARE stmt FROM @q1; EXECUTE stmt; DEALLOCATE
PREPARE stmt;
PREPARE stmt FROM @q2; EXECUTE stmt; DEALLOCATE
PREPARE stmt;

END$$
DELIMITER ;
```

2. Nome SP: EditarUtilizador

// Esta SP permite a edição dos dados de um utilizador a partir de um JSON. Administradores podem editar qualquer utilizador, enquanto jogadores só podem editar os seus próprios dados. Pode atualizar o nome, o telemóvel e a password de acesso à base de dados. Os pormenores das validações estão comentados ao longo do código.

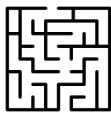
CÓDIGO:

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`EditarUtilizador`(IN `p_dados` JSON)
BEGIN
    DECLARE v_session_user VARCHAR(50);
    DECLARE v_session_tipo VARCHAR(10);
    DECLARE v_email_alvo VARCHAR(50);
    DECLARE v_novo_nome VARCHAR(100);
    DECLARE v_novo_telemovel VARCHAR(20);
    DECLARE v_pass_nova VARCHAR(50);

    -- Obter email do utilizador da sessão
    SET v_session_user := REPLACE(SESSION_USER(),
    '@localhost', '');

    -- Obter tipo de utilizador da sessão
    SELECT Tipo INTO v_session_tipo
    FROM Utilizador
    WHERE Email = v_session_user;

    -- Extrair campos do JSON
    SET v_email_alvo = JSON_UNQUOTE(JSON_EXTRACT(p_dados,
    '$.Email'));
    SET v_novo_nome = JSON_UNQUOTE(JSON_EXTRACT(p_dados,
    '$.Nome'));
    SET v_novo_telemovel =
    JSON_UNQUOTE(JSON_EXTRACT(p_dados, '$.Telemovel'));
```



```
SET v_pass_nova = JSON_UNQUOTE(JSON_EXTRACT(p_dados,
'$>Password'));

-- Verificações de permissão
IF v_session_tipo = 'Adm' THEN
    -- Admin tem de indicar o email do utilizador a
editar
    IF v_email_alvo IS NULL OR TRIM(v_email_alvo) =
'' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Tem de indicar o
utilizador que pretende editar.';
    END IF;

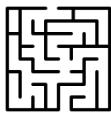
    -- Verificar se o email existe
    IF NOT EXISTS (SELECT 1 FROM Utilizador WHERE
Email = v_email_alvo) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'O utilizador indicado não
existe.';
    END IF; END IF;

IF v_session_tipo = 'Jog' THEN
    -- Jogador só pode editar a si próprio
    IF v_email_alvo IS NOT NULL AND v_email_alvo <>
v_session_user THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Só pode alterar os seus
próprios dados (não é preciso introduzir o email).';
    END IF;
END IF;

-- Forçar edição apenas do próprio
-- Se o email não for fornecido, assumimos que o
jogador está a tentar editar os próprios dados
IF v_email_alvo IS NULL THEN
    SET v_email_alvo = v_session_user;
END IF;

-- Atualizar nome e telemóvel se forem fornecidos
UPDATE Utilizador
SET
    Nome = IFNULL(v_novo_nome, Nome),
    Telemovel = IFNULL(v_novo_telemovel, Telemovel)
WHERE Email = v_email_alvo;

-- Alterar a password se válida
IF v_pass_nova IS NOT NULL AND TRIM(v_pass_nova) <>
'' THEN
```



```
        SET @q = CONCAT('ALTER USER ''', v_email_alvo, ''
IDENTIFIED BY '', TRIM(v_pass_nova), '';');
        PREPARE stmt FROM @q;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;
    END IF;

END$$
DELIMITER ;
```

3. Nome SP: RemoverUtilizador

Esta SP remove um utilizador da base de dados e do MySQL, desde que o seu tipo seja 'Jog' ou 'Mig' (Administradores não se removem uns aos outros).

CÓDIGO:

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`RemoverUtilizador`(IN p_email VARCHAR(50))
begin
    declare tipoVal varchar(20);

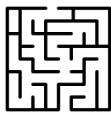
    -- Verifica se o utilizador existe
    SELECT Tipo INTO tipoVal FROM utilizador WHERE email=
p_email;
    IF tipoVal IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Utilizador não existe';

    END IF;
    -- Verifica se o tipo de utilizador é 'Jog' ou 'Mig'
    (admin n pode remover outro admin)
    IF (tipoVal IN ('Jog', 'Mig')) THEN

        -- Remove o utilizador do MySQL
        SET @query = CONCAT("DROP USER '", p_email,
"';");
        PREPARE stmt FROM @query;
        EXECUTE stmt;

        -- Remove o utilizador da tabela 'utilizador'
        DELETE FROM Utilizador WHERE Email = p_email;

    ELSE
```



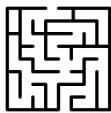
```
SIGNAL SQLSTATE '45000'  
SET MESSAGE_TEXT = 'Nao tem permissoes para  
remover este utilizador';  
END IF;  
END$$  
DELIMITER ;
```

4. Nome SP: CriarJogo

Esta SP cria um novo jogo para um utilizador do tipo jogador (pode ser criado pelo próprio jogador ser atribuído por um Administrador), com base num JSON recebido. Valida permissões e os dados com que está a ser criada e regista o jogo na base de dados com o estado "Agendado". Esta SP não inicia nenhum jogo, apenas os regista e devolve o Id do jogo registado para que possam ser iniciados pela SP IniciarJogo. Os pormenores das validações estão comentados ao longo do código.

CÓDIGO:

```
DELIMITER $$  
CREATE DEFINER=`root`@`localhost` PROCEDURE  
`CriarJogo`(IN p_dados JSON, OUT id_jogo_criado INT)  
BEGIN  
    DECLARE v_session_user VARCHAR(50);  
    DECLARE v_session_tipo VARCHAR(10);  
    DECLARE p_jog_alvo VARCHAR(50);  
    DECLARE p_start_time DATETIME;  
    DECLARE p_descricao VARCHAR(100);  
    DECLARE v_now DATETIME;  
    SET v_now = DATE_SUB(NOW(), INTERVAL 1 SECOND); --  
para tolerar atrasos (atraso de um segundo)  
  
    -- Obter email do utilizador da sessão  
    SET v_session_user := REPLACE(SESSION_USER(),  
'@localhost', '');  
  
    -- Obter tipo de utilizador da sessão  
    SELECT Tipo INTO v_session_tipo  
    FROM Utilizador  
    WHERE Email = v_session_user;  
  
    -- Extrair campos do JSON  
    SET p_jog_alvo = JSON_UNQUOTE(JSON_EXTRACT(p_dados,  
'$>Email'));
```



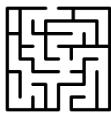
```
SET p_start_time = JSON_UNQUOTE(JSON_EXTRACT(p_dados,
'$._StartTime'));
-- SET p_som = JSON_UNQUOTE(JSON_EXTRACT(p_dados,
'$._Som'));
SET p_descricao = JSON_UNQUOTE(JSON_EXTRACT(p_dados,
'$._Descricao'));

-- Verificação se o email foi fornecido e se existe
um jogador com esse email
IF p_jog_alvo IS NOT NULL AND TRIM(p_jog_alvo) <> ''
THEN
    -- Verificar se o email fornecido existe e é do
    tipo 'Jog'
    IF NOT EXISTS (SELECT 1 FROM Utilizador WHERE
Email = p_jog_alvo AND Tipo = 'Jog') THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Não existe nenhum jogador
com esse email';
    END IF;
END IF;

-- Verificações de permissão
IF v_session_tipo = 'Adm' THEN
    -- Admin tem de indicar o email do utilizador a
editar
    IF p_jog_alvo IS NULL OR TRIM(p_jog_alvo) = ''
THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Tem de indicar o mail de
um utilizador do tipo Jogador para o qual quer criar o
jogo。';
    END IF;
END IF;

IF v_session_tipo = 'Jog' THEN
    -- Jogador só pode criar o jogo para si próprio
    IF p_jog_alvo IS NOT NULL AND p_jog_alvo <>
v_session_user THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Só pode criar um jogo
para si próprio (não é preciso introduzir o email).';
    END IF;
END IF;

-- Forçar edição apenas do próprio
-- Se o email não for fornecido, assumimos que o
jogador está a tentar criar o jogo para si mesmo
IF p_jog_alvo IS NULL THEN
    SET p_jog_alvo = v_session_user;
```



```
END IF;

IF p_start_time IS NULL THEN
    SET p_start_time = NOW();
ELSEIF p_start_time = '' THEN
    SET p_start_time = NOW();
END IF;

-- Validar e ajustar a data/hora de início
IF p_start_time < v_now THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'A data e hora de início devem
ser posteriores à atual';
END IF;

-- Definir descrição padrão se não fornecida
IF p_descricao IS NULL THEN
    SET p_descricao = 'Sem descrição';
END IF;

-- Inserir o novo jogo na tabela Jogo (o som é null
porque só é registado pelo migrador com a informação da
nuvem)
INSERT INTO Jogo(Descricao, Utilizador,
DataHoraInicio, SomMaximo, Estado)
VALUES(p_descricao, p_jog_alvo, p_start_time, null,
'Agendado');

-- Obter ID do jogo recém-criado ()
SET id_jogo_criado = LAST_INSERT_ID();

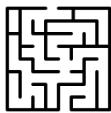
END$$
DELIMITER ;
```

5. Nome SP: IniciarJogo

Esta SP tenta iniciar um jogo com o ID indicado. Se o utilizador for um jogador, só pode iniciar jogos seus. O jogo só é iniciado se não houver outro com estado "Ativo"; caso contrário, é marcado como "Cancelado" e é emitido um erro. Os pormenores das validações estão comentados ao longo do código.

CÓDIGO:

```
DELIMITER $$
```



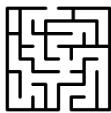
```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`IniciarJogo`(IN `p_idJogo` INT)
BEGIN
    DECLARE v_email_sessao VARCHAR(50);
    DECLARE v_tipo_utilizador VARCHAR(10);

    -- Obter email da sessão
    SET v_email_sessao := REPLACE(SESSION_USER(),
    '@localhost', '');

    -- Obter tipo de utilizador
    SELECT Tipo INTO v_tipo_utilizador
    FROM Utilizador
    WHERE Email = v_email_sessao;

    -- Se for jogador, tem de ser dono do jogo
    IF v_tipo_utilizador = 'Jog' THEN
        IF NOT EXISTS (
            SELECT 1
            FROM Jogo
            WHERE IdJogo = p_idJogo AND Utilizador =
v_email_sessao
        ) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Jogador não tem
permissões para iniciar este jogo.';
        END IF;
    END IF;

    -- Verificar se já existe um jogo a decorrer
    IF ((SELECT COUNT(*) FROM Jogo WHERE Estado =
'Ativo') = 0) THEN
        -- Iniciar o jogo
        UPDATE Jogo
        SET DataHoraInicio = CURRENT_TIMESTAMP(), Estado
= 'Ativo'
        WHERE IdJogo = p_idJogo;
    ELSE
        -- Alterar o estado para Cancelado
        UPDATE Jogo
        SET DataHoraInicio = CURRENT_TIMESTAMP(), Estado
= 'Cancelado'
        WHERE IdJogo = p_idJogo;
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Não é possível começar um
jogo, já está um a decorrer neste momento.';
    END IF;
END$$
DELIMITER ;
```

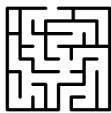


6. Nome SP: EditarJogo

Esta SP permite editar um jogo existente, validando permissões conforme o tipo de utilizador. Jogadores só podem alterar jogos seus, não podem editar jogos ativos, só podem alterar a descrição de jogos terminados e não podem mudar o estado. Administradores têm permissões completas. Os pormenores das validações estão comentados ao longo do código.

CÓDIGO:

```
DELIMITER $$  
CREATE DEFINER='root'@'localhost' PROCEDURE `EditarJogo` (  
    IN p_id_jogo INT,  
    IN p_dados JSON  
)  
BEGIN  
    DECLARE v_session_user VARCHAR(50);  
    DECLARE v_session_tipo VARCHAR(10);  
    DECLARE v_jogAssoc_jogo VARCHAR(50);  
    DECLARE v_estado_jogo VARCHAR(20);  
  
    DECLARE p_start_time DATETIME;  
    DECLARE p_descricao VARCHAR(100);  
    DECLARE p_estado VARCHAR(20);  
  
    -- Verificar se o jogo existe  
    IF NOT EXISTS (SELECT 1 FROM Jogo WHERE IdJogo =  
p_id_jogo) THEN  
        SIGNAL SQLSTATE '45000'  
        SET MESSAGE_TEXT = 'O jogo indicado não existe.';  
    END IF;  
  
    -- Obter utilizador da sessão  
    SET v_session_user := REPLACE(SESSION_USER(),  
    '@localhost', '');  
  
    -- Obter tipo de utilizador  
    SELECT Tipo INTO v_session_tipo  
    FROM Utilizador  
    WHERE Email = v_session_user;
```



```
-- Obter o utilizador associado ao jogo e o estado do
jogo
    SELECT Utilizador, Estado INTO v_jogAssoc_jogo,
v_estado_jogo
        FROM Jogo
    WHERE IdJogo = p_id_jogo;

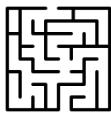
-- Validar permissões do jogador
    IF v_session_tipo = 'Jog' AND v_jogAssoc_jogo <>
v_session_user THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Não tem permissão para editar
este jogo.';
    END IF;

-- Extrair campos do JSON
    SET p_start_time = JSON_UNQUOTE(JSON_EXTRACT(p_dados,
'$StartTime'));
    -- SET p_som = JSON_UNQUOTE(JSON_EXTRACT(p_dados,
'$Som'));
    SET p_descricao = JSON_UNQUOTE(JSON_EXTRACT(p_dados,
'$Descricao'));
    SET p_estado = JSON_UNQUOTE(JSON_EXTRACT(p_dados,
'$Estado'));

-- Validações para jogadores
    IF v_session_tipo = 'Jog' THEN

        IF v_estado_jogo = 'Ativo' THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Jogadores não podem
editar jogos a decorrer.';
        END IF;

        IF v_estado_jogo = 'Terminado' THEN
            -- Só pode editar a descrição
            IF p_start_time IS NOT NULL THEN
                SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Jogadores só podem
editar a descrição de jogos terminados.';
            END IF;
        ELSE
            -- Só pode alterar a hora se for futura
            IF p_start_time IS NOT NULL AND p_start_time
<= NOW() THEN
                SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'DataHoraInicio deve
ser posterior à hora atual.';
            END IF;
        END IF;
    END IF;
```



```
END IF;

    IF p_estado IS NOT NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Jogadores não podem
alterar o estado do jogo.';
    END IF;

END IF;

-- Atualizar o jogo
UPDATE Jogo
SET
    DataHoraInicio = IFNULL(p_start_time,
DataHoraInicio),
    Descricao = IFNULL(p_descricao, Descricao),
    Estado = IFNULL(p_estado, Estado)
WHERE IdJogo = p_id_jogo;

END$$
DELIMITER ;
```

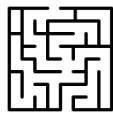
7. Nome SP: CancelarJogo

Esta SP cancela um jogo existente com base no seu ID, desde que esteja no estado "Agendado". Se o utilizador for um jogador, só pode cancelar jogos seus e impede o cancelamento de jogos que já estão em execução ou finalizados. Se a condição não for cumprida, lança um erro apropriado. Os pormenores das validações estão comentados ao longo do código.

```
DELIMITER $$
CREATE DEFINER=root@localhost PROCEDURE CancelarJogo(IN
p_idJogo INT)
BEGIN
    DECLARE v_email_sessao VARCHAR(50);
    DECLARE v_tipo_utilizador VARCHAR(10);

    -- Obter email da sessão
    SET v_email_sessao := REPLACE(SESSION_USER(),
'@localhost', '');

    -- Obter tipo de utilizador
    SELECT Tipo INTO v_tipo_utilizador
    FROM Utilizador
    WHERE Email = v_email_sessao;
```



```
-- Se for jogador, tem de ser dono do jogo
IF v_tipo_utilizador = 'Jog' THEN
    IF NOT EXISTS (
        SELECT 1
        FROM Jogo
        WHERE IdJogo = p_idJogo AND Utilizador =
v_email_sessao
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Jogador não tem
permissões para iniciar este jogo.';
    END IF;
END IF;

-- Só podemos cancelar jogos em estado agendado. não
faz sentido cancelar jogos no passado. E o sjogos a
decorre não podem sofrer qualquer alteração de acordo com
os pressupostos do sistema
IF ((SELECT COUNT(*) FROM Jogo WHERE Estado =
'Agendado' and IdJogo = p_idJogo) = 1) THEN
    -- Cancelar o jogo
    UPDATE Jogo
    SET Estado = 'Cancelado'
    WHERE IdJogo = p_idJogo;
ELSE
    -- Não existe nenhuim jogo em estado que possa ser
cancelado. i.e não há nenhum jogoa gendado
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Não foi encontrado nenhum
jogo em estado que permita o cancelamento para o id de
joiigo indicado. ';
END IF;
END$$
DELIMITER ;
```