



QRreport

47193 João Arcanjo, A47193@alunos.isel.pt

47256 Diogo Novo, A47256@alunos.isel.pt

Orientador: Pedro Miguens, pedro.miguens@isel.pt

Relatório de progresso realizado no âmbito de Projeto e Seminário

Licenciatura em Engenharia Informática e de Computadores

Semestre de verão 2021/2022

22 de julho de 2022

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA



QRreport

47193 João Arcanjo, A47193@alunos.isel.pt

47256 Diogo Novo, A47256@alunos.isel.pt

Orientador: Pedro Miguens, pedro.miguens@isel.pt

Relatório de progresso realizado no âmbito de Projeto e Seminário
Licenciatura em Engenharia Informática e de Computadores
Semestre de Verão - 2021/2022

22 de julho de 2022

Resumo

Hoje em dia as infraestruturas de escritórios, de serviços e de ensino, têm cada vez mais pontos de utilização comum, e nesses locais existem cada vez mais dispositivos, tais como torneiras, sistemas de descarga, interruptores, portas, entre muitos outros.

Com a grande diversidade e o número elevado de dispositivos, quando um destes em específico avaria ou está inoperacional, como por exemplo um problema elétrico, o procedimento para reportá-lo é não só muito trabalhoso e dispendioso em termos de tempo, como também muito confuso, na medida em que, normalmente, nunca se sabe ao certo os procedimentos para reportar o problema encontrado, desde a pessoa ao local onde reportar a anomalia detetada. Quando já se sabe o que fazer, eventualmente, será ainda necessário descrever num papel ou num e-mail o local, o dispositivo e a avaria, sem se saber posteriormente se a reparação já foi realizada.

Tendo em vista estes problemas desenvolvemos um sistema, que pretende mitigar todas estas dificuldades, com o objetivo de diminuir não só o tempo de comunicação de avarias existentes, como também o tempo associado à sua reparação, implementando uma infraestrutura para a comunicação de problemas em edifícios e a sua distribuição por funcionários habilitados a solucionar as avarias, tornando todo este processo muito mais célere para todos os envolvidos. O sistema é composto por uma base de dados manipulada através de um servidor API, que recebe pedidos via HTTP das aplicações cliente.

O ponto de entrada para iniciar o reportar de uma avaria é através de um *QR Code*, que estará localizado nas proximidades do dispositivo passível de ser reportado e redirecionar o dispositivo móvel para um formulário, de forma a que seja submetida a anomalia detetada.

Desta forma, todo este sistema contribui positivamente para a sustentabilidade do planeta, uma vez que não só são minimizados os gastos de papel para reportar anomalias, como também o ato de reportar problemas leva a que se possa tomar conhecimento mais rapidamente de eventuais desperdícios de água, energia e dinheiro.

Palavras-chave: sustentabilidade; servidor API; HTTP; aplicações cliente; *QR Code*.

Abstract

Nowadays, offices, services and education infrastructures have more and more points of common use, and in these places there are more and more devices, such as taps, flushing systems, switches, doors, among many others.

With the great diversity and high number of devices, when a specific or more general device breaks down or is inoperable, such as an electrical problem, the procedure to report it is not only very laborious and time expensive, but also very confusing, because, usually, a person never knows the procedure to report the problem, from whom to talk about it, to the place to report the anomaly detected. When already knows what to do, eventually, it will be necessary to describe on a paper or in an e-mail the localization, the device and the anomaly, without knowing afterwards if the repair is already done.

With these problems in mind, we have developed a system, in which all these difficulties will be centralized, with the objective of reducing not only the time to report existing malfunctions, but also the time associated with their arrangement, implementing an infrastructure for the reporting of problems in buildings and their distribution by employees qualified to solve the problems encountered, making this whole process much faster for everyone involved. The system is composed of a database handled through an API server, which receives requests via HTTP from client applications.

The entry point to start the report of a malfunction is through a QR Code, which will be near a reportable device and will redirect the mobile device to a form in order to submit the detected malfunction.

In this way, this entire system contributes positively to the sustainability of the planet, since not only are the paper waste to report anomalies minimized, but also the act of reporting problems makes it possible to become aware of possible waste of water, energy and money much faster.

Keywords: sustainability; API server; HTTP; client applications; report; QR Code.

Índice

1	Introdução	1
1.1	Objetivos	2
1.2	Organização do relatório	3
2	Arquitetura do sistema	5
2.1	Papéis dos utilizadores	5
2.2	Fluxo de trabalho	6
2.3	Arquitetura proposta	7
3	Implementação do <i>backend</i>	9
3.1	Modelo de dados conceptual	9
3.1.1	Empresas	9
3.1.2	Pessoas	10
3.1.3	<i>Tickets</i>	11
3.1.4	<i>QR Codes</i>	12
3.2	Base de dados	14
3.3	Servidor <i>Web</i>	14
3.3.1	Arquitetura da API	14
3.3.2	Acesso à base de dados	16
3.3.3	Representações das respostas	16
3.3.4	Requisitos funcionais disponibilizados	17
3.4	Servidor de autenticação	18
4	Implementação do <i>Front-end</i>	21
4.1	<i>Workflow</i>	21
4.1.1	<i>Report</i>	22
4.1.2	Processamento do <i>ticket</i> por <i>manager</i>	23
4.1.3	Processamento de reparação pelo funcionário	23
4.1.4	Avaliação por parte do utilizador que reportou	25
4.1.5	Alternativa: agrupamento de <i>ticket</i>	25
4.2	<i>User</i>	25

4.2.1	Autenticação	26
4.2.2	Página de perfil	26
4.2.3	Páginas para a alteração de <i>roles</i> e/ou <i>skills</i> de um utilizador	28
4.3	Empresas, edifícios e salas	29
4.3.1	Empresas	29
4.3.2	Edifícios	30
4.3.3	Salas	31
4.4	Dispositivos	33
4.5	Categorias	35
5	Caso de estudo - ISEL	39
5.1	Dados recolhidos e utilizados	39
5.2	Resultados	40
6	Conclusão e Trabalho futuro	43
6.1	Conclusão	43
6.2	Trabalho futuro	43
Referências		46

Listas de Figuras

2.1	Fluxo de trabalho do sistema	6
2.2	Visão geral da arquitetura do sistema proposto	7
3.1	Modelo de dados conceptual - Empresas	10
3.2	Modelo de dados conceptual - Pessoas	11
3.3	Modelo de dados conceptual - <i>Tickets</i>	12
3.4	Modelo de dados conceptual - <i>QR Codes</i>	12
3.5	Visão geral do modelo de dados conceptual (modelo ER)	13
3.6	Arquitetura da API	16
4.1	Página do <i>report</i> de uma anomalia	22
4.2	Páginas correspondentes à entrega de um <i>ticket</i> a um funcionário ou a sua rejeição	23
4.3	Página para alterar o estado de um <i>ticket</i>	24
4.4	Páginas correspondentes à criação, atualização ou remoção de um comentário num <i>ticket</i>	24
4.5	Páginas correspondentes à avaliação e ao agrupamento de um <i>ticket</i>	25
4.6	Página do menu quando não se está autenticado e página de <i>Login</i> e de <i>Signup</i>	26
4.7	Página do menu de um <i>admin</i> e páginas associadas ao perfil	27
4.8	Páginas para alterar <i>roles</i> , <i>skills</i> e associar uma pessoa a uma determinada empresa	28
4.9	Páginas correspondentes à criação e alteração de empresas	29
4.10	Página detalhada com as informações e ações de uma empresa	30
4.11	Páginas para criar e/ou alterar edifícios	31
4.12	Páginas associadas a uma sala	32
4.13	Páginas das ações associadas a uma sala	33
4.14	Páginas para alterar <i>roles</i> , <i>skills</i> e associar uma pessoa a uma determinada empresa	34
4.15	Páginas associadas às ações de uma anomalia	35
4.16	Páginas das ações associadas a uma sala	36
4.17	Página de perfil (com maior dimensão)	36

4.18 Página de <i>tickets</i> de um <i>manager</i> (com maior dimensão)	37
5.1 Páginas das ações associadas ao <i>ticket</i>	40
5.2 Página do agrupamento do <i>ticket</i> e o <i>ticket</i> agrupado a apresentar as carac- terísticas do <i>ticket</i> 'pai'	41
5.3 Página de da submissão de uma avaliação	41

Acrónimos

ER Entidade Relacionamento

URI Uniform Resource Identifier

SQL Structured Query Language

JSON JavaScript Object Notation

API Application Programming Interface

JDBC Java Database Connectivity

Jdbi Java database interface

DTO Data Transfer Object

DAO Data Access Object

RFC Request for Comment

HTTP Hypertext Transfer Protocol

Capítulo 1

Introdução

Hoje em dia os edifícios não são meras construções de betão, mas uma agregação de vários subsistemas, desde as redes de distribuição de água, eletricidade e climatização, dos elevadores às infraestruturas de dados, entre outros. Estes subsistemas, apesar de serem complexos, não estão isentos de anomalias, necessitando, por vezes, de intervenções de reparação.

A deteção de anomalias, a sua notificação e posterior adjudicação da intervenção é eventualmente um processo demorado e não centralizado, procrastinando a reparação. Esta demora provoca prejuízos, quer sejam por danos derivados da anomalia ou por inoperância do sistema.

Deste modo, tendo em vista o problema, pretende-se centralizar e diminuir o tempo de comunicação das avarias implementando uma infraestrutura para comunicação de anomalias de dispositivos presentes em edifícios. Para além do processo de comunicação, tenciona-se que o sistema permita o reencaminhamento destas por: localização; tipo de avaria; ou por tipo de intervenção.

Além da gestão de anomalias e respetivas reparações, pretende-se implementar também uma comunicação no sentido inverso, de modo a validar as reparações efetuadas e aumentar a satisfação dos utilizadores.

De forma a identificar os pontos de anomalia/intervenção, o sistema irá recorrer a identificadores do tipo *QR Code* [1].

Este sistema é concebido como um serviço que pode ser adquirido por diversas empresas, e não restrito apenas a uma, sendo, por isso, adaptável ao domínio de cada empresa. De momento o sistema irá utilizar como caso de estudo as infraestruturas do ISEL, tirando partido do conhecimento e facilidade de acesso aos seus edifícios, salas e dispositivos para ser possível testar e validar todas as funcionalidades implementadas.

1.1 Objetivos

Tendo em consideração o problema mencionado, e para cumprir com os objetivos estabelecidos foi idealizado um sistema que permitisse tornar todo o processo referido anteriormente mais fácil e eficiente. Para tal definiu-se um conjunto de objetivos/requisitos essenciais do sistema para alcançar o funcionamento pretendido, e outro conjunto de requisitos opcionais que irão complementar as funcionalidades essenciais.

Requisitos essenciais:

- Planeamento do modelo de dados e posteriormente transformação para o modelo físico numa base de dados relacional para armazenar todos os dados necessários inerentes ao nosso sistema, tais como funcionários, empresas, informações de avarias submetidas, entre outros;
- Criação de uma *RESTful API* [2] para manipular todos os recursos necessários da base de dados, implementar a maior parte da lógica da aplicação e receber os pedidos via *HTTP* por parte das aplicações cliente;
- Gerar *QR Codes* e suportar a sua leitura;
- Conceção e desenvolvimento de um servidor de autenticação com o objetivo de facilitar a utilização de meios de autenticação externos;
- Conceção e implementação de uma aplicação *Web* para os utilizadores interagirem com sistema concebido e desenvolvido.

Requisitos opcionais:

- Desenvolvimento de uma aplicação móvel, como alternativa à aplicação *Web*, cujo foco é facilitar o acesso ao sistema aos funcionários, os quais a partir desta aplicação poderão também receber notificações, por exemplo, acerca de avarias que lhe foram delegadas para solucionar;
- Conceção e desenvolvimento de um algoritmo, através de técnicas de Inteligência Artificial, para estabelecer medidas preventivas de manutenção de eventuais anomalias, evitando, assim, a necessidade de ser sempre reportado manualmente um certo problema periodicamente.

1.2 Organização do relatório

Este relatório está organizado em seis capítulos, sendo que neste primeiro realiza-se a introdução aos problemas e aos objetivos deste projeto.

No Capítulo 2 é retratada a arquitetura do sistema e são apresentados alguns de detalhes do sistema implementado, isto é, informações relevantes acerca *workflow* da aplicação e as funções/papéis que os utilizadores podem desempenhar.

No Capítulo 3 é apresentada a implementação da arquitetura proposta, sendo descritas as opções tomadas para cada um dos componentes envolvidos na implementação dos requisitos essenciais, relativamente ao *backend* do sistema.

No Capítulo 4 ao contrário do anterior, são descritas as funcionalidades e apresentadas as suas páginas da parte do *frontend* do sistema, dizendo respeito à aplicação *Web*.

No Capítulo 5 é apresentado o caso de estudo aplicado ao ISEL, de forma a serem validados tanto os requisitos arquitetónicos como os funcionais propostos.

No Capítulo 6 apresentam-se as conclusões sobre o trabalho desenvolvido nos módulos implementados e o trabalho futuro associado à evolução do sistema proposto.

Capítulo 2

Arquitetura do sistema

Neste capítulo são apresentados mais detalhes e características do sistema de forma a serem concretizados os objetivos anteriormente mencionados. Serão expostos os diferentes papéis (*roles*) que os utilizadores do sistema podem assumir, o fluxo de trabalho da aplicação e, por fim, os componentes da arquitetura que possibilitam a implementação de todos os recursos e funcionalidades necessárias. Todos estes módulos do nosso sistema encontram-se descritos na Secção 2.1, Secção 2.2 e Secção 2.3, respetivamente.

2.1 Papéis dos utilizadores

Os utilizadores da aplicação podem desempenhar um ou mais papéis dos cinco considerados:

1. **Convidado** (*guest*) pode submeter um *report* de alguma anomalia encontrada sem necessitar de ter conta no sistema. *Report* é o termo dado ao formulário, obtido através de um *QR Code*, que uma pessoa submete para o sistema;
2. **Utilizador** (*user*) corresponde a uma pessoa com conta no sistema que, para além de poder reportar anomalias como o convidado, poderá também consultar o seu histórico de *reports*, visualizar o estado do processo de reparação das anomalias que reportou e também atribuir uma pequena avaliação (0 a 5) relativamente ao *ticket*, termo dado dentro da aplicação a um problema submetido, que foi resolvido/finalizado;
3. **Funcionário** (*employee*) pode consultar os trabalhos/*tickets* que lhe foram atribuídos para resolver, realizar anotações sobre o estado da reparação e também progredir ou regredir os estados do *ticket*;
4. **Gestor** (*manager*) pode gerir um ou mais edifícios e tem como principal responsabilidade delegar trabalho aos seus colaboradores;
5. **Administrador** (*admin*) tem as permissões e os recursos para gerir todo o sistema, desde criar novos utilizadores, a novas empresas, edifícios, dispositivos, entre outros.

2.2 Fluxo de trabalho

De forma a clarificar todo o processo, abaixo está descrito o fluxo de trabalho do sistema, cuja numeração corresponde à mesma utilizada na Figura 2.1 de forma a ilustrar melhor o funcionamento do sistema desde o *report* inicial até o problema estar resolvido.

1. Um utilizador (*guest* ou *user*) irá realizar um *report* através de um *QR Code* referente a um problema encontrado.
2. O *report* será enviado para o servidor que disponibiliza a API do sistema, e será criado/aberto um *ticket*.
3. Posteriormente, um *manager* responsável pelo edifício, sobre o qual foi realizado o *report*, irá analisar os *tickets* submetidos e atribuí-los aos funcionários com as competências necessárias para a resolução dos problemas submetidos.
4. O funcionário irá, então, começar a solucionar o primeiro problema que foi atribuído. À medida que existirem progressos ou problemas poderá, como já foi referido, progredir ou regredir o estado do *ticket* e também realizar anotações relativamente ao estado da reparação, existindo, uma via de comunicação direta e rápida com o *manager*.
5. Quando a reparação estiver concluída, isto é, quando o funcionário sinalizar a sua conclusão através do estado, o utilizador irá ser notificado e poderá ir até esse *ticket* e atribuir uma avaliação simbólica.

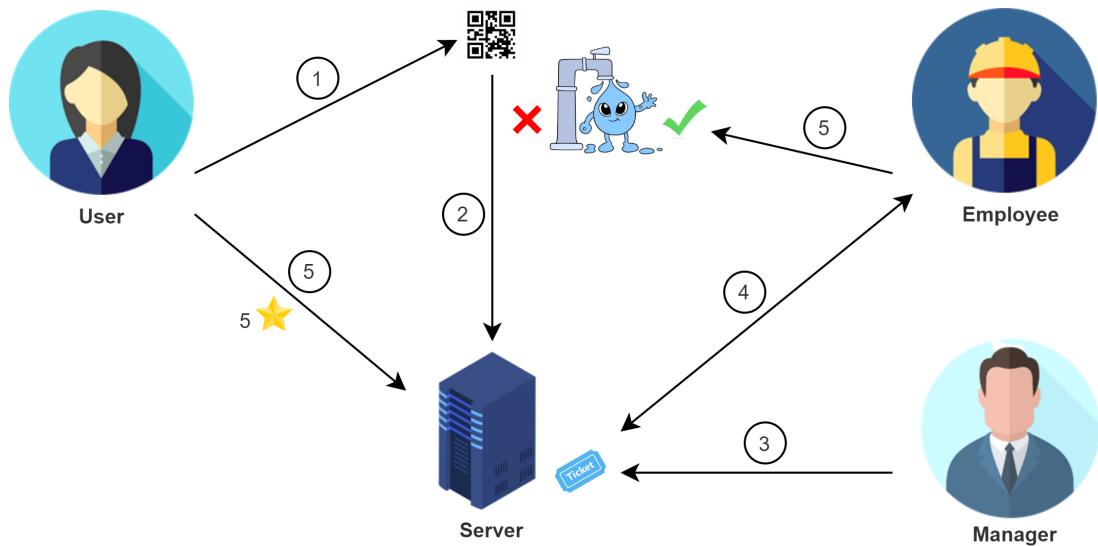


Figura 2.1: Fluxo de trabalho do sistema

2.3 Arquitetura proposta

A arquitetura do sistema proposto tem como módulos essenciais: *i*) um servidor *Web*; *ii*) um servidor de autenticação; *iii*) uma base de dados; e *iv*) uma aplicação *Web*. Os componentes acinzentados correspondem aos não essenciais para o funcionamento do sistema, mas que poderão ser desenvolvidos e facilmente adaptados aos restantes módulos, isto é, a criação de uma aplicação móvel e a ligação a um servidor externo de autenticação. Na Figura 2.2 encontra-se representada a visão geral da arquitetura proposta.

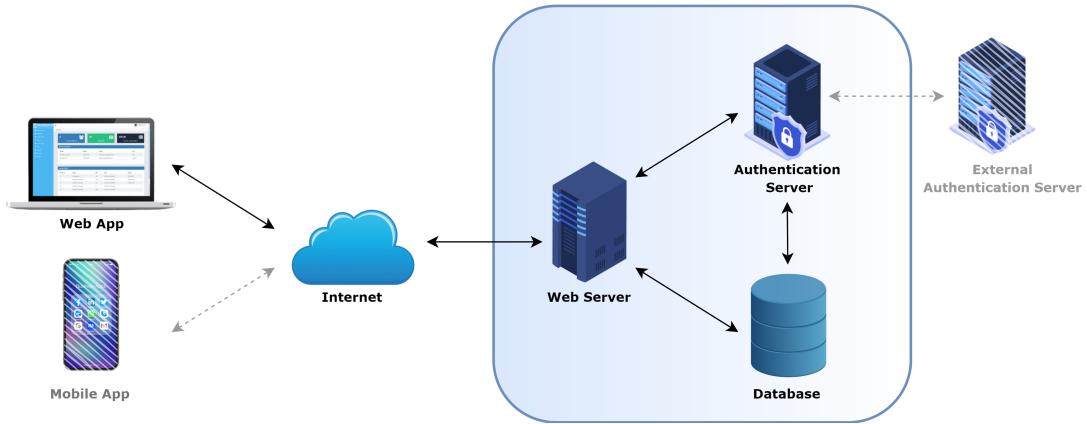


Figura 2.2: Visão geral da arquitetura do sistema proposto

O servidor *Web* implementa uma *RESTful API* e é o intermediário de todas as comunicações no sistema. Este irá não só receber pedidos por parte dos clientes, isto é, aplicação *Web* e eventual aplicação móvel, mas também interagir com os pedidos de autenticação e aceder aos dados persistidos na base de dados.

A aplicação móvel irá, tal como já foi referido anteriormente, facilitar o acesso ao sistema pelos funcionários e disponibilizar a ocorrência de notificações relativamente a, por exemplo, *tickets* que tenham sido delegados.

O servidor de autenticação externo irá permitir que, caso o seu funcionamento tenha um comportamento idêntico ao serviço de autenticação local, o serviço de autenticação privado de uma empresa possa ser facilmente adicionado ao nosso sistema. Visto que o ISEL será utilizado como caso de estudo, pretende-se implementar o seu serviço de autenticação.

Capítulo 3

Implementação do *backend*

Apresentada toda a arquitetura, todos os requisitos e objetivos do sistema, neste capítulo descrevem-se os detalhes e as decisões tomadas na implementação dos componentes referidos no capítulo anterior. Na Secção 3.1 são descritos os pressupostos associados ao desenho do modelo de dados conceptual e na Secção 3.2 é descrita a implementação do modelo de dados físico, isto é, a estrutura da base de dados. Na Secção 3.3 e na Secção 3.4 são apresentadas as decisões tomadas relativamente ao servidor *Web* e ao servidor de autenticação, respetivamente.

3.1 Modelo de dados conceptual

Para que fosse possível armazenar todos os dados necessários associados a todo o domínio do sistema, e de forma a que toda a lógica desejada pudesse ser respeitada, foi desenhado um modelo Entidade-Relacionamento. Com o objetivo de explicitar com detalhe todos os aspectos do modelo desenvolvido, este foi fragmentado em quatro partes, as quais são descritas nas próximas quatro subsecções: *i*) empresas; *ii*) pessoas; *iii*) *tickets*; e *iv*) *QR Codes*. No final desta secção é apresentada ainda, na Figura 3.5, a visão geral de todo o modelo conceptual, que reúne todos os fragmentos mencionados.

3.1.1 Empresas

Dado que o nosso sistema tem como objetivo ser um serviço, que pode ser adquirido por diversas empresas, teve de ser desenvolvido um modelo que possibilitasse existir divisão de dados de empresa para empresa. Com isto em mente, foram idealizadas três entidades, **COMPANY**, **BUILDING** e **ROOM**, ilustradas na Figura 3.1. A primeira para ficarem todas as empresas que adquiriram o nosso serviço e as restantes para serem armazenados os edifícios e salas de cada uma destas, respetivamente. Para além do nome dado a cada elemento armazenado em cada uma destas entidades, a entidade relativa aos edifícios (**BUILDING**) conta ainda uma coluna para o número de pisos de cada edifício, e cada sala (**ROOM**) tem associado o piso dentro do seu edifício. Todas estas entidades têm também um estado

associado que pode assumir dois valores, *ativo* ou *inativo* (na implementação do sistema usamos sempre a terminologia em inglês e por isso, neste caso será *active* ou *inactive*). Ao estado duma entidade está associado um *timestamp*, ou seja, um marco temporal do momento em que o estado foi alterado, tanto para ativo como para inativo. A existência deste estado tem como fim, por exemplo, quando alguma empresa pretende não estar mais associada ao nosso sistema, que os seus dados permaneçam no mesmo mas com o estado *inactive*, utilizando a técnica de *soft delete*. Este processo de *soft-delete* será aplicado em todas as entidades que possuam o campo *state*.

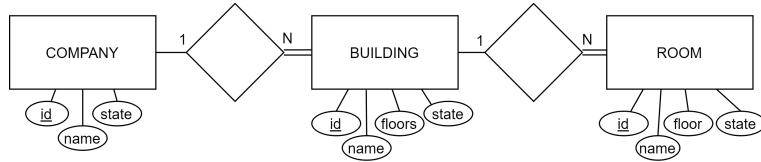


Figura 3.1: Modelo de dados conceptual - Empresas

3.1.2 Pessoas

Todos as pessoas (**PERSON**) que utilizam a aplicação podem desempenhar um ou mais papéis (**ROLE**) dentro da mesma, como foi referido e descrito anteriormente, no Capítulo 2 (*guest*, *user*, *employee*, *manager* e/ou *admin*). A relação entre as pessoas e os seus papéis é realizada através da relação **PERSON_ROLE**, como representada na Figura 3.2. Embora possa ter um conjunto de papéis (*roles*), cada pessoa poderá apenas ter um destes ativo (relação **ACTIVE_ROLE**).

A cada *employee* estão associadas também uma ou mais competências (registadas em **PERSON_SKILL**), as quais serão usadas pelos *managers*, para ao atribuírem a tarefa existente, escolherem os funcionários com melhor capacidades para solucionar o problema em questão. Estes serão também associados a, no mínimo, uma empresa, e como tal, só poderão solucionar os problemas das empresas a que pertencem (relação **PERSON_COMPANY**). Os *managers* para além de também terem, pelo menos, uma empresa associada, serão responsáveis por gerir um ou mais edifícios relativos às empresas a que estão associados.

De forma a não eliminar permanentemente uma pessoa do sistema caso, por exemplo, já não desempenhe nenhuma função, tal como referido para as empresas, existe um campo para alterar o estado das mesmas para inativo. Neste caso, para além dos estados *active* e *inactive*, existe ainda o estado *banned*, uma vez que, caso um *guest* ou *user* não esteja a utilizar a aplicação com responsabilidade, seja possível impedi-lo de utilizar o sistema. Existe ainda um campo *reason* para que a pessoa (*manager* ou *admin*) que realizou a restrição possa indicar a razão pela qual o fez.

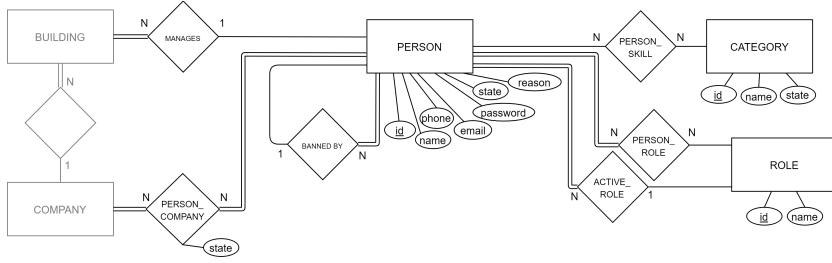


Figura 3.2: Modelo de dados conceptual - Pessoas

3.1.3 Tickets

No sistema, a um *ticket* (**TICKET**) está associado não só as informações relativas ao problema notificado e a pessoa que o realizou, mas também o local e o dispositivo anómalo. Tal como já foi mencionado anteriormente, o *manager* irá analisar os *tickets* abertos e delegá-los a um colaborador com as capacidades necessárias para a resolução do problema em questão. O *manager* será capaz de realizar esta associação, uma vez que a cada dispositivo está associado uma categoria (**CATEGORY**), que tem correspondência direta com as competências de cada colaborador. Caso já exista um *ticket* a ser resolvido com as mesmas características, o novo poderá herdar as informações de um *ticket* 'pai', realizando assim um agrupamento.

Os *tickets* têm associado dois estados, o primeiro para ser visualizado pelos colaboradores (tanto funcionários como gestores) e pelo administrador, o qual apresenta com precisão o atual estado do processo de resolução do *ticket* (**EMPLOYEE_STATE**), e o segundo para ser observado apenas pelos utilizadores que reportaram o problema em questão, que irá apresentar um estado, menos pormenorizado, acerca da situação em que a reparação se encontra (**USER_STATE**).

Cada *ticket*, à medida que é processado, suporta também a criação de anotações, com o objetivo de existir um canal de comunicação mais fácil, direto e rápido entre o funcionário que está a resolver o problema e o gestor, existindo, assim, a possibilidade de reportar algum impedimento na resolução ou qualquer outra informação que ambas as partes considerem relevantes (**COMMENT**). Após o estado de um *ticket* ter sido alterado para arquivado, não será possível realizar mais alterações sobre o mesmo.

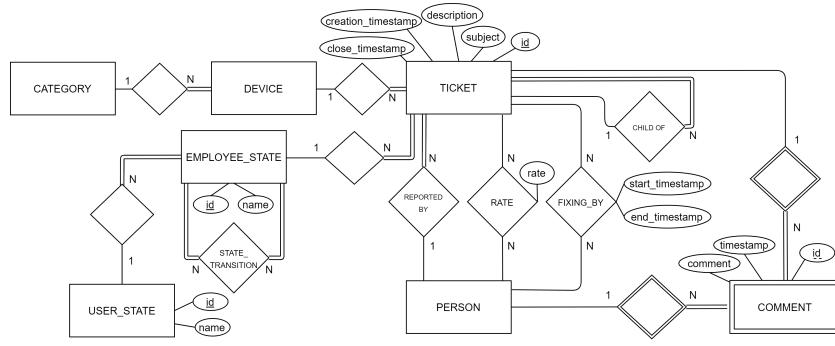


Figura 3.3: Modelo de dados conceptual - *Tickets*

3.1.4 QR Codes

Cada sala contém um conjunto de dispositivos, e a cada um destes está associado um *QR Code*, o qual serve para redirecionar a pessoa que está a reportar para uma página *Web* com um formulário simples, de forma a que possa ser submetido o *report* relativo à anomalia encontrada. Como medida preventiva, o URI que estará representado em cada *QR Code* é composto por um *hash*, com o objetivo de que não sejam expostos os identificadores dos recursos que estão a ser acedidos, isto é, o identificador da empresa, do edifício, da sala e do dispositivo, uma vez que estes iriam estar expostos a qualquer pessoa, potencialmente com más intenções, e desta forma, caso haja algum problema, pode ser facilmente gerado outro *QR Code*, e por sua vez outro *hash*, enquanto se fossem os identificadores seria mais complexo e com custos mais elevados alterá-los. O *hash* estará associado a uma sala e a um dispositivo, sendo esta relação **ROOM_DEVICE**, apresentada na Figura 3.4. Caso o problema a ser reportado não tenha um dispositivo em específico, cada sala poderá ter também *QR Codes* genéricos, para problemas mais gerais, como algum problema elétrico, ar condicionado, rede de computadores, entre outros.

Cada dispositivo (**DEVICE**) tem ainda agregado a si um conjunto de anomalias mais comuns (**ANOMALY**), as quais são sugeridas ao utilizador que estiver a realizar um *report* de forma a que o preenchimento possa ser mais célere, mas caso não identifique nenhuma das recomendações com o problema que encontrou, poderá também preencher os campos com as informações que considerar relevantes.

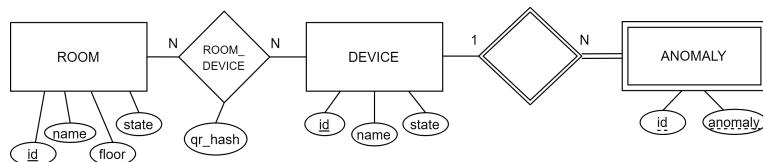
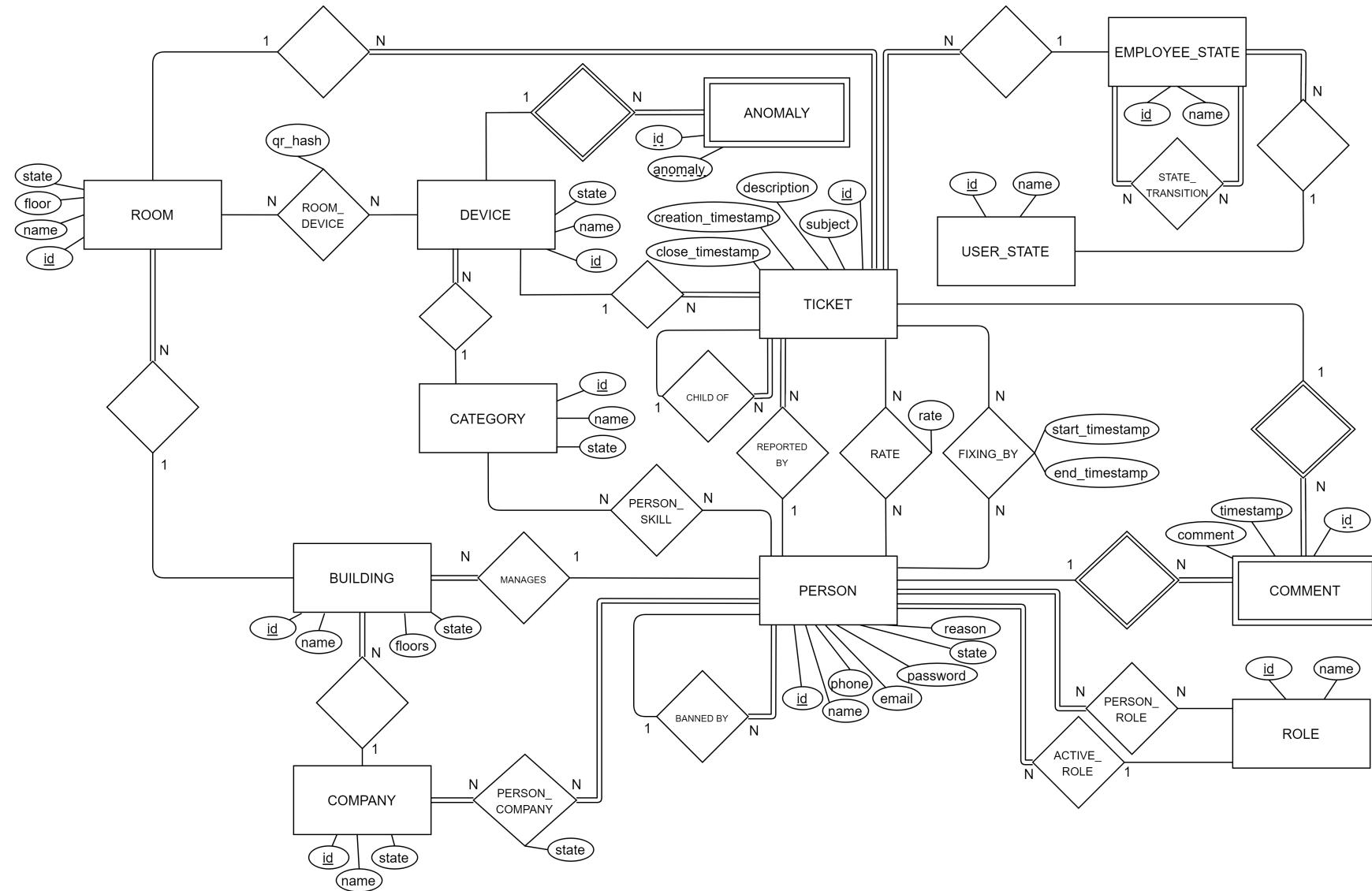


Figura 3.4: Modelo de dados conceptual - *QR Codes*



RI: All entities with the 'state' column have also a column called 'timestamp'.

Figura 3.5: Visão geral do modelo de dados conceptual (modelo ER)

3.2 Base de dados

Após a definição do modelo ER, foi construído o modelo físico da base de dados utilizando PostgreSQL [3]. Foi selecionada esta base de dados relacional, uma vez que, para além de ser *open source* e de apresentar as características comuns de qualquer base de dados relacional SQL, dá ainda a possibilidade de serem utilizados objetos não relacionais, isto é, objetos ou representações no formato JSON [4].

Para além da base de dados, foi também implementada uma parte da lógica do sistema, através da utilização de procedimentos armazenados (*procedures*), funções, gatilhos (*triggers*) e vistas. Os procedimentos são utilizados apenas quando se pretende realizar alguma escrita, os quais para retornar têm de recorrer aos parâmetros de *output* (existem dois tipos de parâmetros, de *input*, o caso normal das funções/métodos, ou de *output*). A partir dos de *output*, é retornada sempre uma representação em JSON para o servidor com o resultado referente ao pedido que foi efetuado. Optou-se por esta abordagem não só por facilitar o mapeamento direto para um objeto em memória, mas por principalmente evitar fazer mais acessos desnecessários à base de dados, uma vez que os dados provenientes de entidades diferentes, podem ser agregados numa única representação no formato JSON.

Caso exista algum conflito, em termos da lógica da aplicação, irá ocorrer uma exceção na base de dados, que irá retornar um identificador único e estável do problema que ocorreu, para que no lado do servidor este possa ser recolhido e interpretado, de forma a que seja possível enviar para o cliente uma mensagem personalizada com informação do erro ocorrido.

3.3 Servidor Web

O servidor *Web* é o hospedeiro da API do nosso sistema e nesta secção são descritas as tecnologias usadas e a função que desempenham na solução, tal como o tipo de respostas que o servidor envia para os clientes, tanto em respostas de sucesso como de insucesso.

Para implementar a *Web API* foi utilizada a *framework Spring Boot* [5], desenvolvida em linguagem *Kotlin*. A escolha para ser utilizada esta linguagem comparativamente com *Java* foi maioritariamente uma questão de preferência, mas está também associada à diminuição de, por exemplo, código *boilerplate* [6], isto é, a repetição de código com pouca variação em diversos locais, tal como, a criação de um tipo em *Java* que teria de conter um construtor, *setters* e *getters*, e com o *Kotlin* é tão simples como usar a *syntax data class* [7] para definir um tipo.

3.3.1 Arquitetura da API

Para a concretização da API, foram utilizadas as dependências *Spring Security* [8], descrita na Secção 3.4, e *Spring Web MVC* [9]. Esta última é a dependência responsável pela

interação com os pedidos *HTTP*, a qual disponibiliza um conjunto de componentes essenciais (ilustrados na Figura 3.6):

- *Filters* - Responsáveis por estabelecerem a primeira interação com o pedido *HTTP*, podendo ser filtrado de acordo com configurações, e também muito úteis para realizar o registo (*logging*) dos pedidos e das respostas. Este componente ao não se encontrar dentro do ambiente da *framework Spring*, não é possível, por exemplo, apanhar possíveis exceções e ser retornada uma mensagem de erro com um formato específico definido no *Spring*.
- *Interceptors* - Desempenham basicamente as mesmas funções que os filtros, mas já se encontram dentro da *framework Spring* e, como tal, já é possível retornar as mensagens de erro com o formato determinado para as mesmas, sendo, por isso, o componente ideal para realizar verificações antes dos pedidos chegarem aos *Controllers*.
- *Argument Resolvers* - Têm o objetivo de interpretarem os dados presentes nos pedidos *HTTP* e, caso não haja à priori nenhum erro, transformá-los num objeto de domínio, que será fornecido ao *Controller* correspondente. Sendo assim realizada uma primeira verificação e entregue ao *Controller* apenas um objeto com os dados recebidos, ao invés deste receber cada uma das propriedades por parâmetro, isto é, por exemplo, em vez da função do *Controller* receber através de parâmetros distintos *page* e *limit* referentes à paginação, irá receber apenas um objeto, construído pelos *Argument Resolvers*, contendo estas propriedades.
- *Controllers* - A sua principal função é lidar tanto com os pedidos *HTTP*, interpretando a informação enviada e encaminhá-la para a camada seguinte, ou seja, os serviços, como a construção das respostas e o seu envio para o cliente, isto é, receber os resultados do pedido feito a partir do *service*, e não só definir todos os *headers* como também construir as respostas *HTTP* com o formato definido, apresentado na Subsecção 3.3.3.
- *Services* - É a camada que implementa a maior parte da lógica de negócio do sistema, sendo que, como já foi referido anteriormente, a outra parte está definida na base de dados. A partir dos argumentos recebidos, são construídos objetos de domínio (ou DTOs, *Data Transfer Objects*), para que se todos os dados estiverem válidos, sejam fornecidos à camada de acesso a dados.
- *Data Access Objects* (DAOs) - Correspondem à camada de acesso aos dados persistidos na base de dados, cuja implementação realizada é referida na próxima Subsecção 3.3.2.

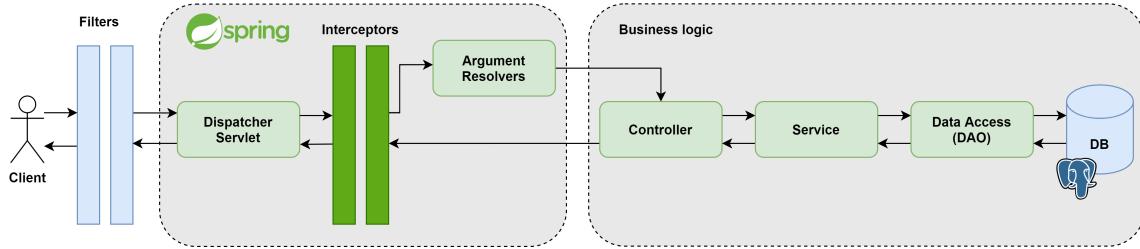


Figura 3.6: Arquitetura da API

3.3.2 Acesso à base de dados

Para realizar o acesso à base de dados relacional PostgreSQL através dos *DAOs*, foi escolhida a API declarativa da Jdbi [10]. Tomou-se esta decisão devido aos seguintes fatores:

- É construída sobre a JDBC [11], fornecendo uma API com uma abordagem/conceito semelhante, tendo, assim, uma curva de aprendizagem mais curta;
- Tem suporte para a linguagem Kotlin;
- Fornece uma API mais simples e natural comparativamente com a disponibilizada pela JDBC.

Através de anotações, a API declarativa da Jdbi acede aos recursos desejados. No caso dos resultados serem provenientes dos parâmetros de *output* de procedimentos armazenados, será retornado um objeto *OutParameters* com o resultado e posteriormente será mapeado para um objeto de domínio, caso contrário o resultado poderá ser mapeado diretamente tanto para um objeto como para uma *string*.

3.3.3 Representações das respostas

Em todas as respostas o servidor envia para o cliente uma representação, de sucesso ou insucesso/erro, referente aos resultados obtidos a partir do pedido realizado.

Para as **respostas de sucesso** optou-se por uma abordagem similar à especificação *hypermedia* Siren [12]. Apesar de ser muito versátil, foi constatado que as *actions*, que são uma forma de informar o cliente de comportamento que a entidade em questão possibilita, tais como apagar um recurso, eram limitadas ao *media-type application/x-www-form-urlencoded*, e por sua vez, nos seus campos só podia ser inserido valores com os tipos de *input* descritos no HTML5 [13]. Como tal, foi adicionada a possibilidade de serem adicionados valores de *input* através do *media-type application/json*, no qual os tipos das suas propriedades são aquelas que o formato JSON [4] suporta, visto que também não era possível indicar qual dos campos a serem inseridos eram obrigatórios ou opcionais, foi adicionada a propriedade booleana *required* para esse efeito, sendo apenas necessário ser inserida com o valor *false* quando se trata de uma propriedade opcional, caso contrário pode ser omitida, considerando

o valor por omissão como *true*, ou seja, obrigatório. Para permitir o envio de eventuais valores que possam ser inseridos nas respetivas propriedades, foi também adicionada a propriedade *possibleValues* que contém um URI para obter estes possíveis valores.

Tendo em conta que também não existia possibilidade de indicar ao cliente quando estava presente um *templated* URI, foi adicionada uma propriedade booleana *templated* para esse efeito, sendo inspirada na especificação HAL [14]. O seu formato foi construído de acordo com a especificação *URI Template* descrita no RFC 6570 [15].

Uma vez que a especificação Siren foi alterada, foi decidido alterar a designação do *media-type* da representação de sucesso para *application/vnd.qrreport+json*.

Já para as **respostas de erro ou insucesso** foi escolhida a especificação referente ao *media-type application/problem+json* descrito no RFC 7807 - *Problem Details for HTTP APIs* [16]. À mesma foram adicionadas duas propriedades com o objetivo de ser possível fornecer ao cliente mais informação sobre o erro que ocorreu quando for necessário:

- *invalidParameters* - Adicionada apenas para erros relativos a *Bad Requests*, que conta com um corpo específico, isto é, um objeto com as propriedades, *name*, *local* e *reason*, que servem para, respetivamente, indicar o nome do campo onde ocorreu o erro, o local onde ocorreu e a razão pela qual aconteceu;
- *data* - Adicionada para quaisquer outros erros HTTP, não tendo um corpo específico, pode conter qualquer informação adicional que seja importante transmitir ao cliente.

3.3.4 Requisitos funcionais disponibilizados

O servidor API através das representações nas respostas disponibiliza um conjunto de funcionalidades essenciais ao sistema, as quais estão enumeradas abaixo e mais à frente no relatório, nomeadamente no Capítulo 4, serão demonstradas as vistas a partir da aplicação *Web* que disponibilizam a sua interação com o utilizador.

Funcionalidades disponibilizadas:

- Submissão de *reports* de anomalias e, por sua vez, a criação de *tickets*;
- Entrega/associação de *tickets* a funcionários (*managers*);
- Alteração dos estados dos *tickets* (*employee/manager*);
- Realizar comentários sobre *tickets* (*employee/manager*);
- Consultar tickets correspondentes (*user, employee, manager*);
- Criação e atualização de empresas, dispositivos, anomalias e categorias, tal como a sua mudança de estado (*admin*);
- Criação e atualização de edifícios, salas e *QR Codes* (*manager*);

- Consulta dos funcionários de uma empresa e possibilidade de despedir e recontratar (*manager*);
- Consulta de todas as pessoas, categorias, dispositivos e anomalias (*admin*);
- Criação de funcionários e a sua atribuição a uma empresa (*manager*);
- Atribuição/remoção de um perfil(*role*) a uma pessoa (*admin*);
- Atribuição/remoção de uma competência(*skill*) a um funcionário (*admin*);
- Associação de dispositivos a salas e a possibilidade de removê-los (*manager*);
- Consulta de informações pessoas e possibilidade para atualizá-las (*user, employee, manager, admin*);
- Mudança de perfil(*role*) ativo, no caso em que a pessoa possui mais do que um.

Apesar de não ser referido em cada uma das funcionalidades, os administradores têm as permissões necessárias para aceder a quase todos os recursos.

3.4 Servidor de autenticação

O servidor de autenticação foi implementado em conjunto com o servidor *Web*, utilizando os componentes *filters* e *interceptors* que a plataforma *Spring MVC* disponibiliza (Figura 3.6). O mecanismo de autenticação foi desenvolvido através de *JSON Web Tokens* (JWT) [17], que permitem evitar que seja necessário confirmar as credenciais do utilizador todas as vezes que é realizado um pedido ao servidor, depois de este já se encontrar autenticado.

Ao início, quando um utilizador realiza um pedido de autenticação, o servidor ao receber as suas credenciais irá, através do algoritmo de *hash BCrypt* [18], disponibilizada pelo *Spring Security* [8], codificar a palavra-passe fornecida em conjunção com um *salt* [19], que é automaticamente gerado pelo algoritmo e armazenado em conjunto com a palavra-passe codificada, de forma a que quando seja necessário comparar duas palavras-passe, o algoritmo consiga e saiba como obter o *salt* anterior respetivo. Basicamente, o *salt* permite que duas palavras-passe iguais não fiquem armazenadas na base de dados com a mesma codificação, ou seja, irá tornar cada palavra-passe única no sistema, mesmo que no fundo sejam iguais. Já em relação ao algoritmo *BCrypt*, optou-se por este em detrimento de *SHA256*, uma vez que este realiza o *hash* mais complexo devido ao facto de realizar várias iterações de *hashing* utilizando um *salt* aleatório, sendo desta forma muito mais seguro e forte para proteger palavras-passe.

Após verificadas as credenciais, a base de dados irá retornar uma representação das informações essenciais do utilizador, tais como, nome, email, papéis, competências e/ou empresas a que pertence. Estas informações irão ser utilizadas na construção do JWT com o objetivo de não ser necessário nos próximos pedidos obter estes dados associados ao utilizador

autenticado novamente. Desta forma, sempre que chegar um novo pedido e o servidor declarar o *token* recebido como válido, irá mapear os dados que este contém para um objeto de domínio, *AuthPerson*, sendo a partir desse momento apenas necessário recorrer a esse objeto para realizar um conjunto de primeiras verificações relativas à lógica de negócio, como por exemplo, verificar se a pessoa autenticada tem autorização para aceder ao recurso pretendido.

Sempre que uma autenticação não tiver sucesso ou quando tem mas o utilizador não possui as permissões necessárias para aceder ao recurso, serão emitidas as mensagens de erro HTTP 401 - *Unauthorized* ou 403 - *Forbidden*, respetivamente.

Pensando no futuro, e no eventual desenvolvimento de aplicação externa a um *browser*, tal como uma aplicação móvel, foi tomada a decisão de ser adicionado um novo *header* a todos os pedidos HTTP, designado por *Request-Origin*, com o objetivo de ser possível obter facilmente no servidor a informação de que tipo de dispositivo realizou o pedido, tendo por enquanto, dois valores possíveis, *i) WebApp*, quando é proveniente da aplicação *Web*, e *ii) Mobile*, quando teve origem na aplicação *mobile*. Foi feita esta distinção de forma a que para aplicação *Web* o JWT seja fornecido via *Cookies* [20], com os atributos *HTTP-Only* ativo e *Same-Site Strict*, de forma a que não seja possível manipular o *token* através de aplicações *JavaScript*, reduzindo assim a existência de possíveis acessos malignos à API. E no caso de uma aplicação móvel, de forma a facilitar o envio do *token* num pedido, este será enviado e recebido através do *header Authorization*.

Capítulo 4

Implementação do *Front-end*

Neste capítulo serão apresentados os detalhes da implementação da aplicação *Web*, sendo apresentadas as principais funcionalidades das páginas que constituem a aplicação construída.

A aplicação foi desenvolvida através da *framework React* [21] e foi utilizada a linguagem *TypeScript* [22] por ser uma linguagem fortemente tipificada, reduzindo possíveis erros em tempo de compilação. Relativamente ao grafismo das páginas *Web*, foi utilizada uma mistura entre *CSS* [23], *TailwindCSS* [24] e outras *frameworks* que já forneciam alguns componentes prontos a utilizar, tal como *React Icons* [25].

A aplicação contém apenas uma página *HTML*, e através do *React* vão sendo renderizados um conjunto de componentes de forma a ilustrar a existência de diferentes páginas na aplicação utilizando-se a biblioteca *React Router* [26] para permitir a ilusão da existência de diferentes páginas.

A existência de *Hypermedia* nas respostas do servidor API, permite que a aplicação *Web* não esteja totalmente dependente da lógica do servidor API. Como por exemplo, alterar algum tipo de comportamento através da propriedade *actions* das representações, tal como uma ação para apagar um recurso, pode ser controlada a partir do servidor.

Após a autenticação, serão armazenadas algumas informações referentes ao utilizador no *Session Storage* [27] do *browser*, e na secção das *Cookies* estará armazenado o *token* (*JWT*) que será fornecido ao servidor em todos os pedidos realizados.

4.1 *Workflow*

Nesta secção serão apresentadas todas as páginas que pertencem ao *workflow* do sistema, ou seja, as páginas que estarão presentes desde a submissão de um *ticket* por parte de um utilizador, autenticado ou não, até à resolução do mesmo, tendo o utilizador, a partir desse momento, a possibilidade de atribuir uma pequena avaliação (1 a 5 estrelas) ao trabalho que foi concluído. O utilizador só conseguirá realizar esta avaliação caso se encontre autenticado.

4.1.1 Report

Uma pessoa ao encontrar uma anomalia num dispositivo e realizar o *scan* do *QR Code* correspondente, irá ser redirecionado para a página para preencher o formulário do *report*, representada na Figura 4.1. Nesta, é apresentada uma pequena descrição, através da qual o utilizador pode verificar qual o dispositivo que está a reportar, bem como a localização do mesmo dentro do edifício, de maneira a possibilitar o utilizador de confirmar se o *QR Code* lido corresponde, realmente, ao dispositivo que pretende reportar. Na página apresenta-se ainda um formulário para o utilizador explicar qual o problema que encontrou, selecionando uma das anomalias que estão previamente associadas ao dispositivo, ou, caso o utilizador considere que nenhuma dessas descreva o problema encontrado, ao selecionar a opção "*Other Problem...*", aparecerá um novo *input*, designado por *Subject*, no qual o utilizador poderá inserir manualmente a anomalia que considere mais adequada que as demais. Seguidamente, o utilizador deverá inserir uma descrição, na qual terá de explicar sucintamente e detalhadamente o problema encontrado, caso não tenha escolhido nenhuma anomalia o seu preenchimento é obrigatório, caso contrário é opcional. Posteriormente, caso não se encontre autenticado, terá de introduzir algumas informações pessoais, nomeadamente o seu nome, o seu email e caso pretenda, o seu contacto telefónico. Por fim, antes de ser submetido o *report*, o utilizador terá de validar a sua submissão através de um *reCAPTCHA* [28], incluído com o objetivo de proteger o nosso serviço, limitando o acesso e tráfego criado por *bots*. Após esta validação, o utilizador ao pressionar o botão *Submit*, irá enviar o *report* para o servidor API, gerando a criação de um *ticket*.

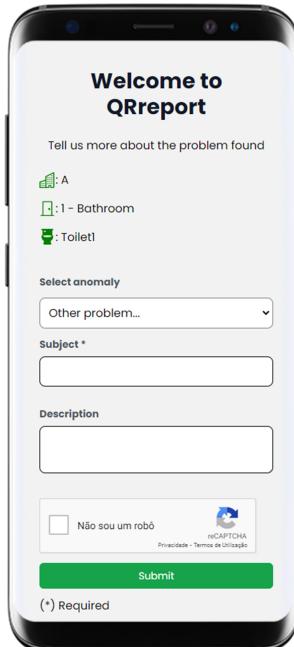


Figura 4.1: Página do *report* de uma anomalia

4.1.2 Processamento do *ticket* por *manager*

Após a criação do *ticket* por parte do utilizador, os *managers*, da empresa associada ao *ticket*, poderão aceder ao mesmo através da página destinada a listar os *tickets*, ilustrada na Figura 4.2 (a).

Caso considerem que o *ticket* é válido e que deva ser solucionado, podem entregá-lo a um funcionário que possua a competência (*skill*) necessária para resolver o problema encontrado, Figura 4.2 (b). Caso contrário, pode ser efetuada a atualização do seu estado para *Refused*, tal como se pode constatar pela representação na Figura 4.2 (c).

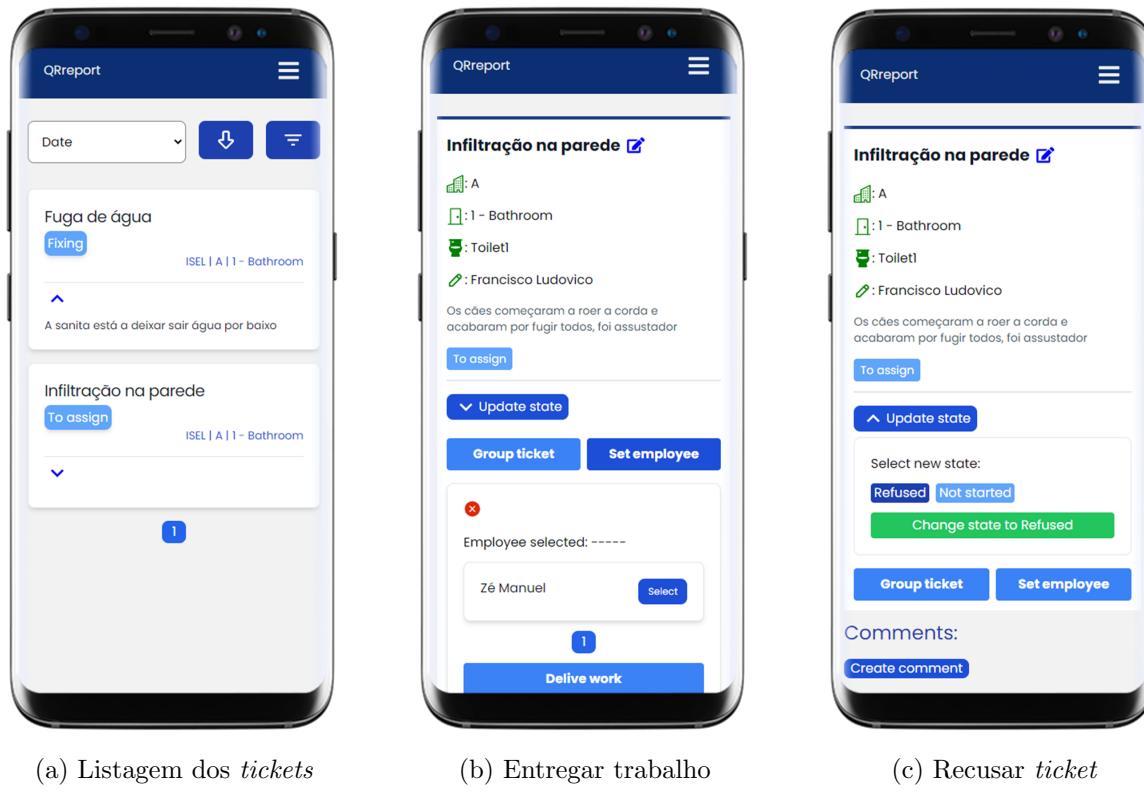


Figura 4.2: Páginas correspondentes à entrega de um *ticket* a um funcionário ou a sua rejeição

4.1.3 Processamento de reparação pelo funcionário

O funcionário ao aceder aos detalhes dos *tickets* que possui, poderá alterar o seu estado, possibilitando, assim, os restantes utilizadores do sistema que têm acesso ao *ticket* de visualizarem o estado da resolução em que o mesmo se encontra, ilustrada na Figura 4.3.

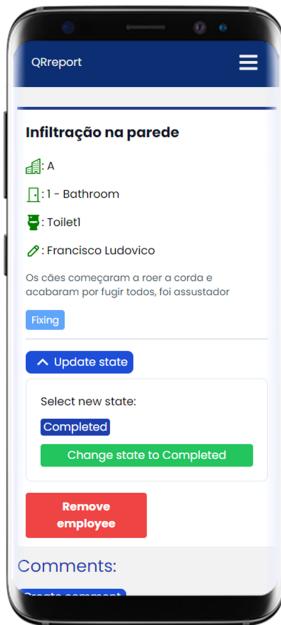
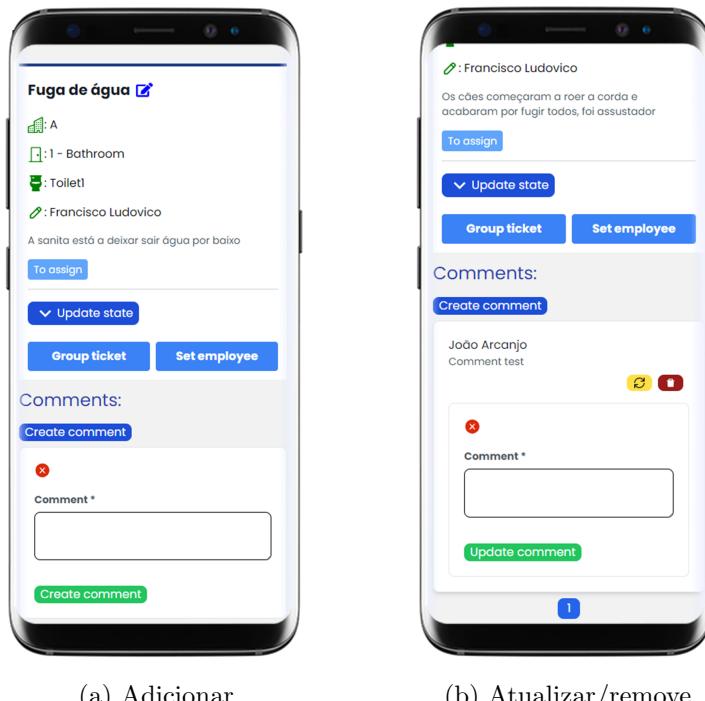


Figura 4.3: Página para alterar o estado de um *ticket*

Enquanto o *ticket* ainda não se encontra no estado de *Archived* ou *Refused*, o funcionário e os *managers* que possuem acesso ao mesmo, podem realizar comentários na sua página, tal como se pode verificar na Figura 4.4 (a). Caso a pessoa considere que deve reformular o comentário, pode atualizá-lo ou até mesmo apagá-lo definitivamente, Figura 4.4 (b).



(a) Adicionar

(b) Atualizar/remove

Figura 4.4: Páginas correspondentes à criação, atualização ou remoção de um comentário num *ticket*

4.1.4 Avaliação por parte do utilizador que reportou

O *employee* ou o *manager*, ao alterar o estado de um determinado *ticket* para *Archived* significa que deu o trabalho como concluído. Esta alteração de estado levará a que do lado do utilizador que submeteu o *ticket*, apareça uma *box* a partir da qual poderá submeter uma avaliação de 0-5 do trabalho realizado por parte do funcionário, como representado na Figura 4.5 (a).

4.1.5 Alternativa: agrupamento de *ticket*

De maneira a evitar que diferentes *tickets* que correspondam ao mesmo problema sejam entregues a funcionários diferentes ou que um determinado funcionário possua diversos trabalhos que representem o mesmo problema, o sistema possibilita a agregação de *tickets*, possibilitando o *manager*, na altura de entregar o *ticket* a um funcionário, de associá-lo a um que já tenha sido previamente submetido, herdando do mesmo o seu estado e todos os comentários que forem submetidos no mesmo, esta operação é representada na Figura 4.5 (b).

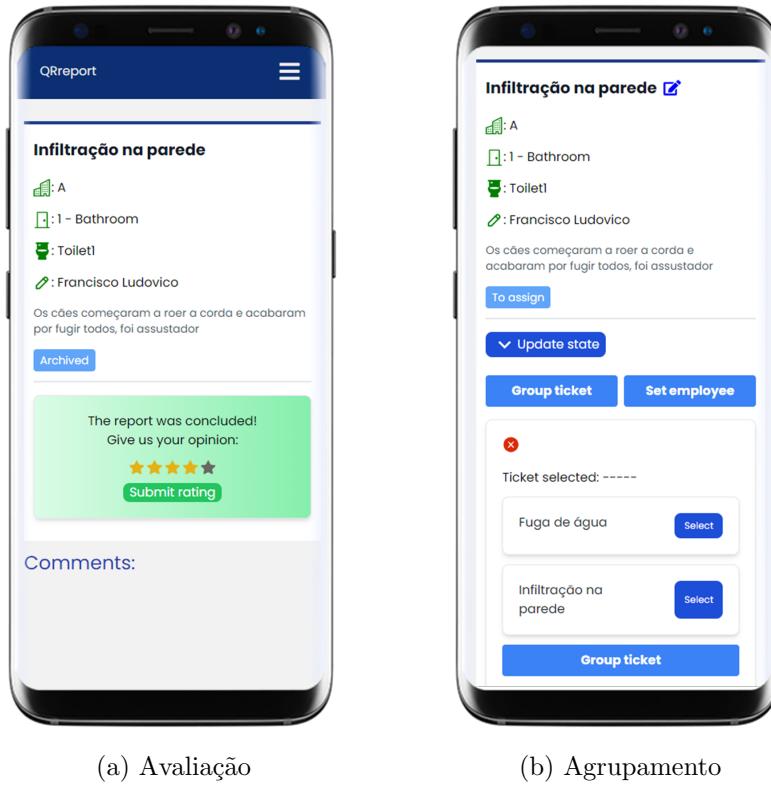


Figura 4.5: Páginas correspondentes à avaliação e ao agrupamento de um *ticket*

4.2 User

Os utilizadores têm a possibilidade de realizar algumas alterações sobre as informações que possuem e outras que apenas podem ser alteradas a partir de outras pessoas com mais

permisões, nomeadamente um *manager* ou um *admin* do sistema. Nesta secção serão apresentadas todas as vistas envolvidas com o perfil do utilizador.

4.2.1 Autenticação

Os utilizadores para usufruírem das funcionalidades do serviço necessitam de se autenticarem, para tal, quando qualquer utilizador não autenticado acede ao nosso sistema, irá ser redirecionado para a página de *login* (Figura 4.6 (a)) e ao acederem ao menu é apresentada apenas a opção de *login* tal como se pode verificar na Figura 4.6 (b).

Caso o utilizador ainda não possua conta no sistema, ao selecionar o botão de *signup*, apresentado na página de *login*, será obtida a página ilustrada na Figura 4.6 (c), na qual terão de ser inseridas todas as informações necessárias para criar uma nova conta, nomeadamente o seu nome, email, número de telefone (opcional) e a password para conseguir aceder à conta.

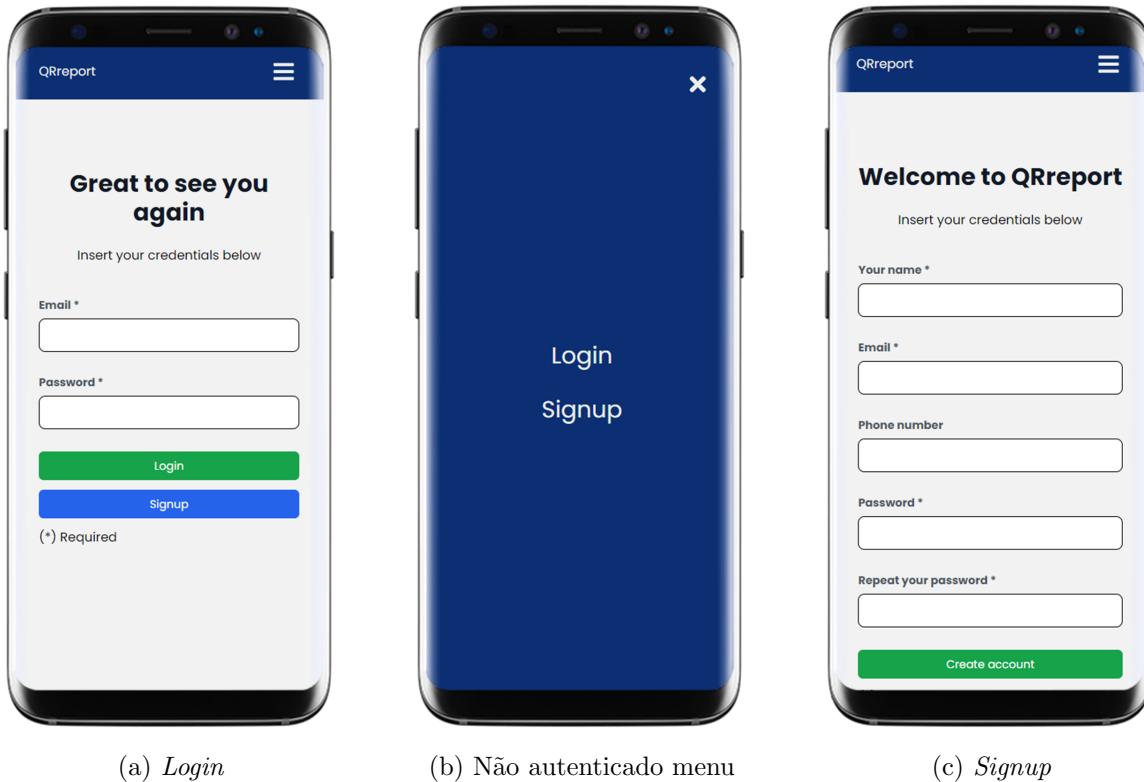


Figura 4.6: Página do menu quando não se está autenticado e página de *Login* e de *Signup*

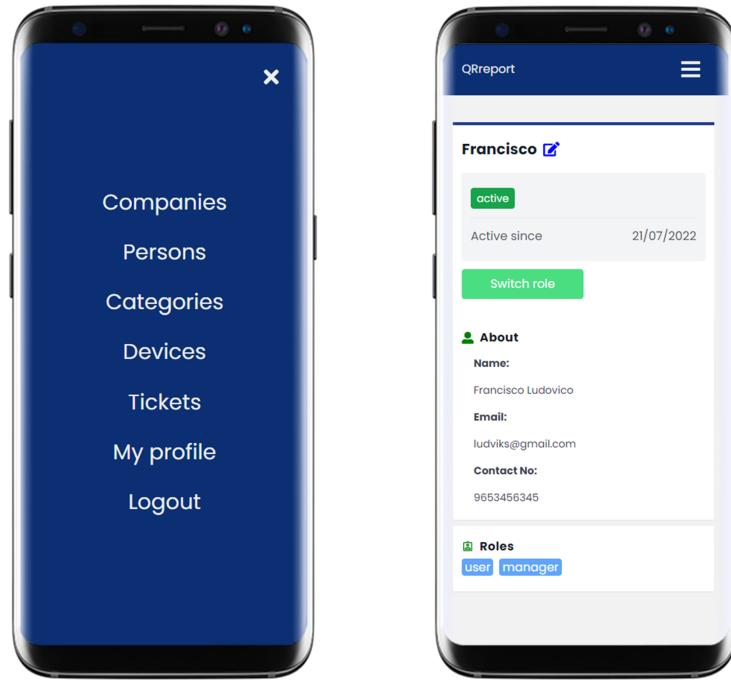
4.2.2 Página de perfil

Uma vez realizada a autenticação, o menu será atualizado, apresentando ao utilizador todos os recursos do sistema que este pode aceder, páginas essas que variam conforme a *role* possuída pelo utilizador autenticado. Na Figura 4.7 (a) é possível verificar o menu que é apresentado ao utilizador que possui a *role* de *admin*.

O utilizador ao seleccionar a opção *My Profile* do menu principal, é lhe apresentado a sua

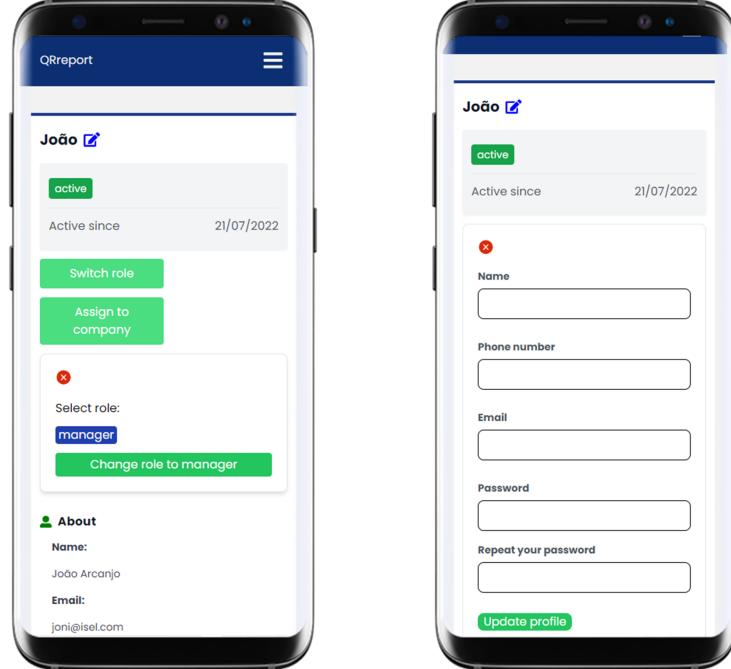
página de perfil, Figura 4.7 (b).

No seu perfil poderá ainda trocar a *role* que tem ativa, para uma outra, no caso de estar associado a mais do que um papel no sistema (Figura 4.7 (c)), tal como atualizar as suas informações pessoais, através da opção *update* (Figura 4.7 (d)).



(a) Admin menu

(b) Perfil



(c) Trocar perfil

(d) Atualizar perfil

Figura 4.7: Página do menu de um *admin* e páginas associadas ao perfil

4.2.3 Páginas para a alteração de *roles* e/ou *skills* de um utilizador

Caso o utilizador possua a *role* de *manager* ou de *admin* consegue aceder à página de dados dos utilizadores, as quais apresentam todas as suas informações, bem como as ações que podem ser efetuadas, tais como adicionar e/ou remover *roles* ao utilizador (ilustradas na Figura 4.8 (a) e (b)), caso possua a *role* *employee*, a possibilidade de adicionar e/ou remover competências (*skills*) do mesmo (ilustradas na Figura 4.8 (c) e (d)) e também a possibilidade de associar o mesmo a uma nova empresa (Figura 4.8 (e)).

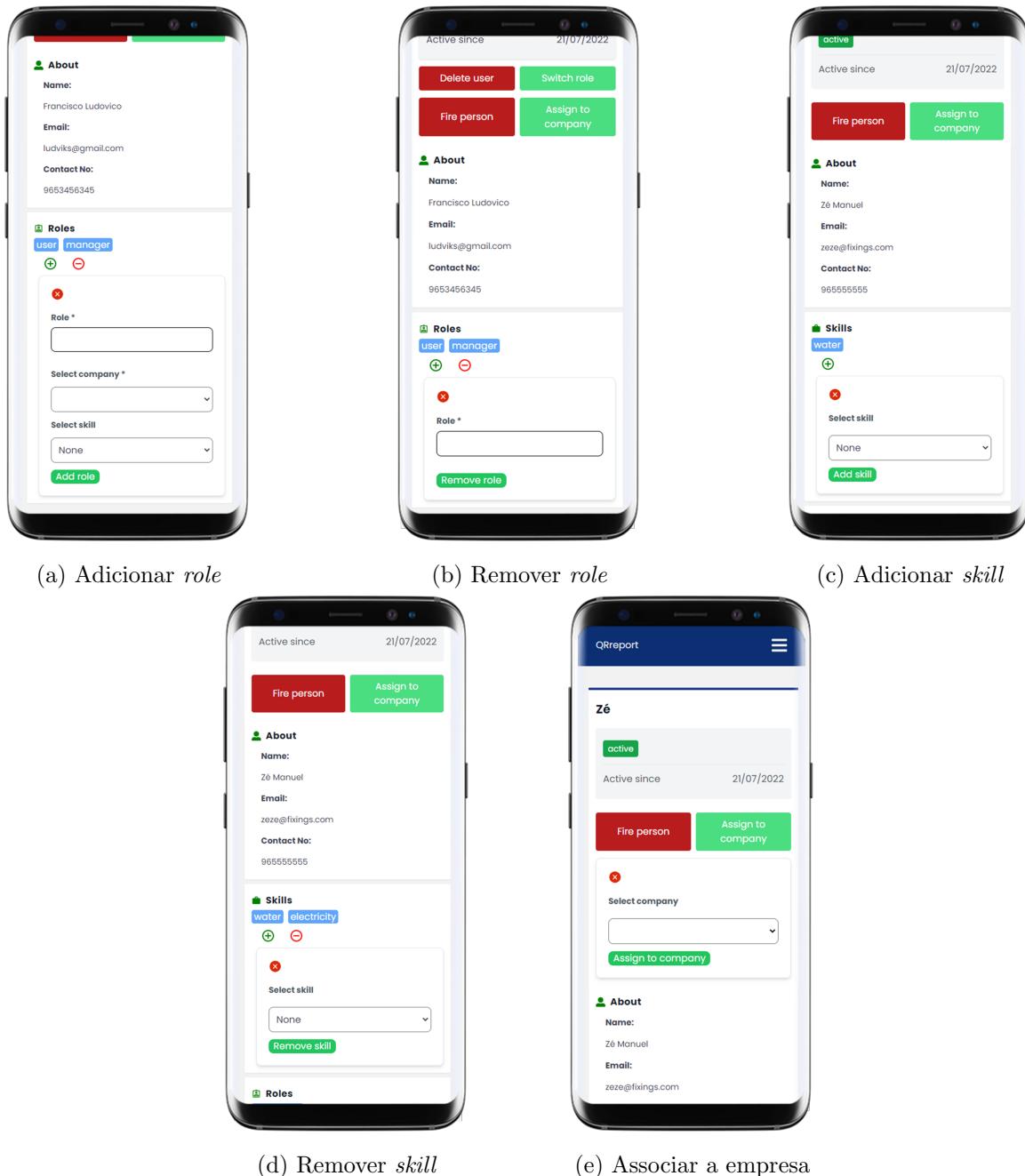


Figura 4.8: Páginas para alterar *roles*, *skills* e associar uma pessoa a uma determinada empresa

4.3 Empresas, edifícios e salas

As páginas apresentadas nesta secção são acessíveis apenas aos *managers* e *admins* do sistema. Sendo que apenas os administradores é que possuem as autorizações necessárias para criar uma nova empresa no sistema, sendo por isso necessário que quando uma empresa pretende utilizar o nosso serviço, contacte a administração.

4.3.1 Empresas

Apesar de tanto os *managers* como os *admins* conseguirem aceder à página onde ocorre a listagem das empresas, os *managers* apenas conseguem aceder àquelas a que pertencem, enquanto os *admins* têm acesso privilegiado a todas as registadas no sistema (Figura 4.9 (a)), sendo estes também os únicos que conseguem realizar a criação de novas empresas no sistema (Figura 4.9 (b)) ou alterar as existentes (Figura 4.9 (c)).

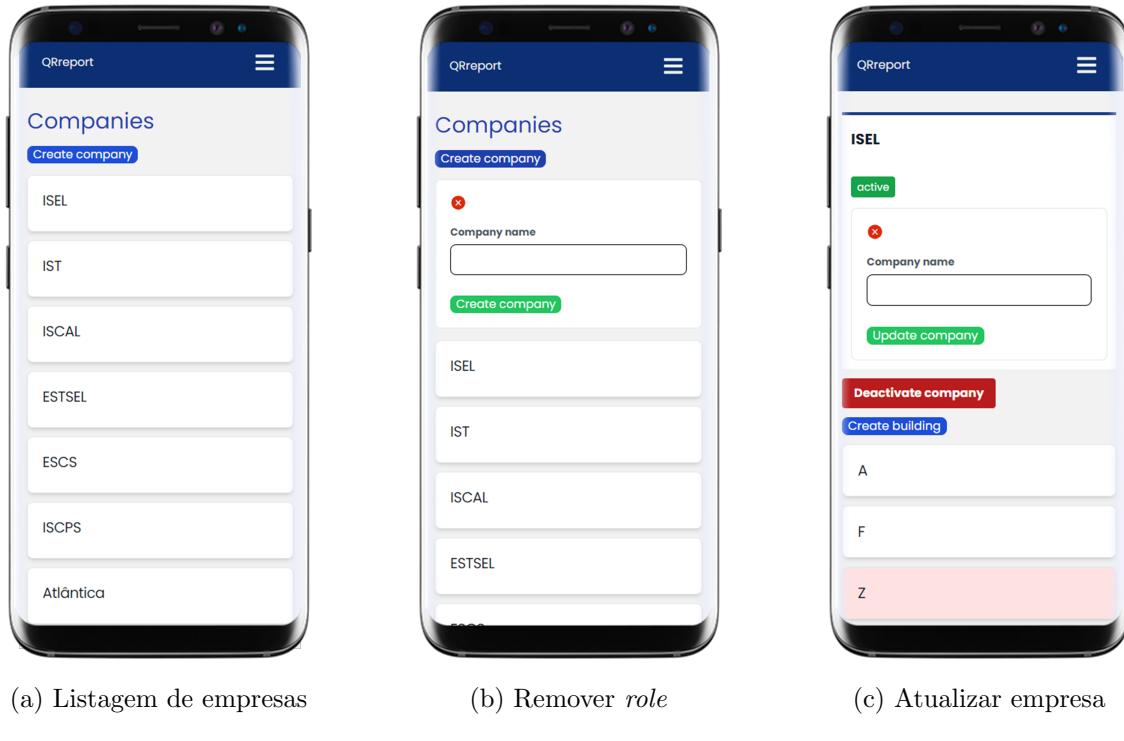


Figura 4.9: Páginas correspondentes à criação e alteração de empresas

Para que se acceda a toda a informação de uma determinada empresa, nome, edifícios associados e ações, é necessário aceder à página que apresenta os detalhes sobre a mesma, página essa que é representada na Figura 4.10.

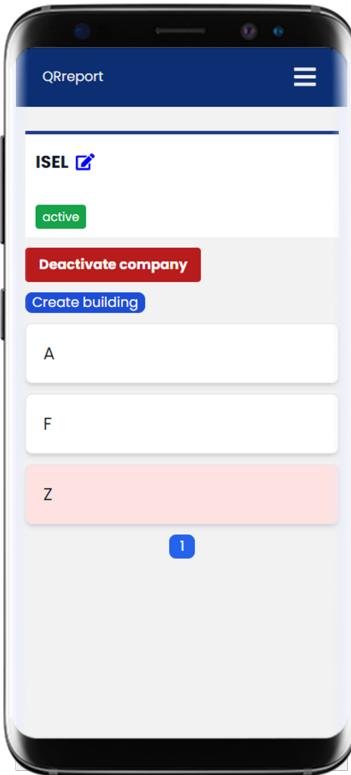


Figura 4.10: Página detalhada com as informações e ações de uma empresa

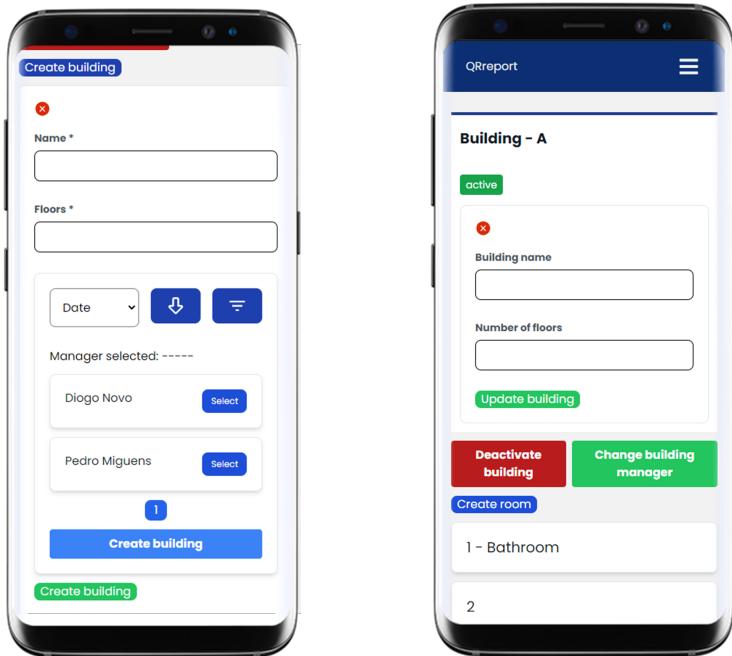
4.3.2 Edifícios

A funcionalidade de criação de um novo edifício está presente na página referente à empresa a que se quer associar o novo elemento. Esta opção é desempenhada pelos *managers* da empresa em questão, e na sua criação, para além de inserir os dados do novo edifício, nomeadamente o nome e o número de pisos, tem de selecionar um *manager* para ficar responsável por esse edifício, esta funcionalidade é ilustrada na Figura 4.11 (a).

Tal como já foi referido, a listagem dos edifícios de uma determinada empresa está presente na página da mesma, tal como se pode verificar na Figura 4.10. A alteração dos dados de um determinado edifício pode ser realizado, ao contrário das empresas, tanto pelos *admins* do sistema, como pelo *manager* do edifício, Figura 4.11 (b).

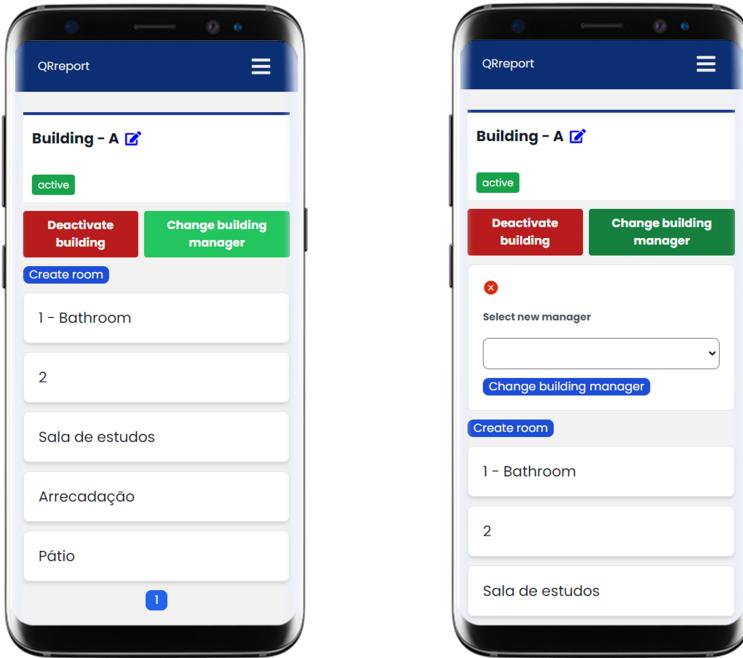
Ao acedermos a uma página que representa um determinado edifício, será apresentada ao utilizador uma página com as suas informações detalhadas, tal como apresenta a Figura 4.11 (c).

A página do edifício possui três funcionalidades, *i)* desativar o edifício, caso esse se encontre ativo, *ii)* ativar o edifício, caso o seu estado seja inativo e *iii)* alterar o *manager* responsável pelo edifício, Figura 4.11 (d).



(a) Criar edifício

(b) Atualizar edifício



(c) Detalhes de edifício

(d) Alterar *manager*

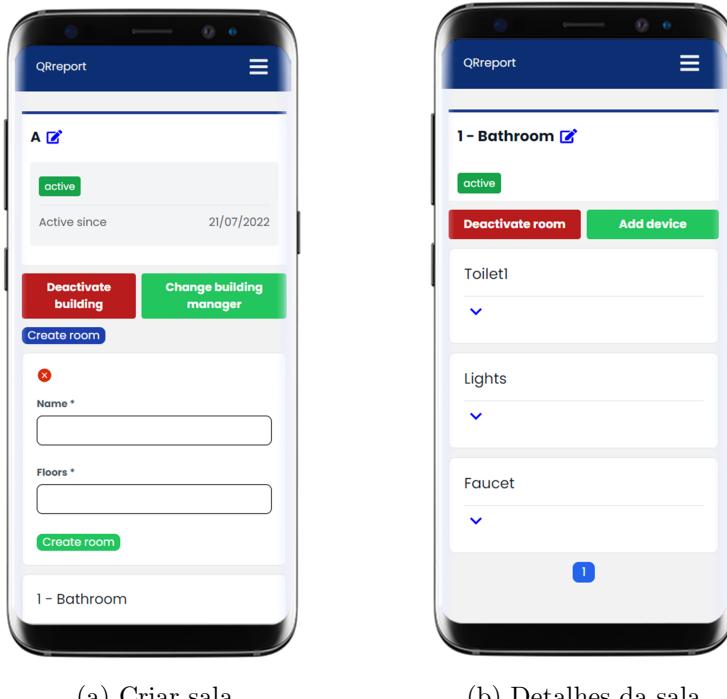
Figura 4.11: Páginas para criar e/ou alterar edifícios

4.3.3 Salas

Para a criação de uma nova sala é necessário que o utilizador seja o *manager* do edifício ao qual a sala irá ser adicionada, Figura 4.12 (a)

A listagem das salas encontra-se presente na página do edifício ao qual a sala pertence, tal como se pode verificar na Figura 4.11 (c).

A página que se destina a apresentar toda a informação relativamente a uma determinada sala é apresentada na Figura 4.12 (b), nesta o *manager* e os *admins* podem ativar ou desativar a sala, dependendo do estado atual da mesma.



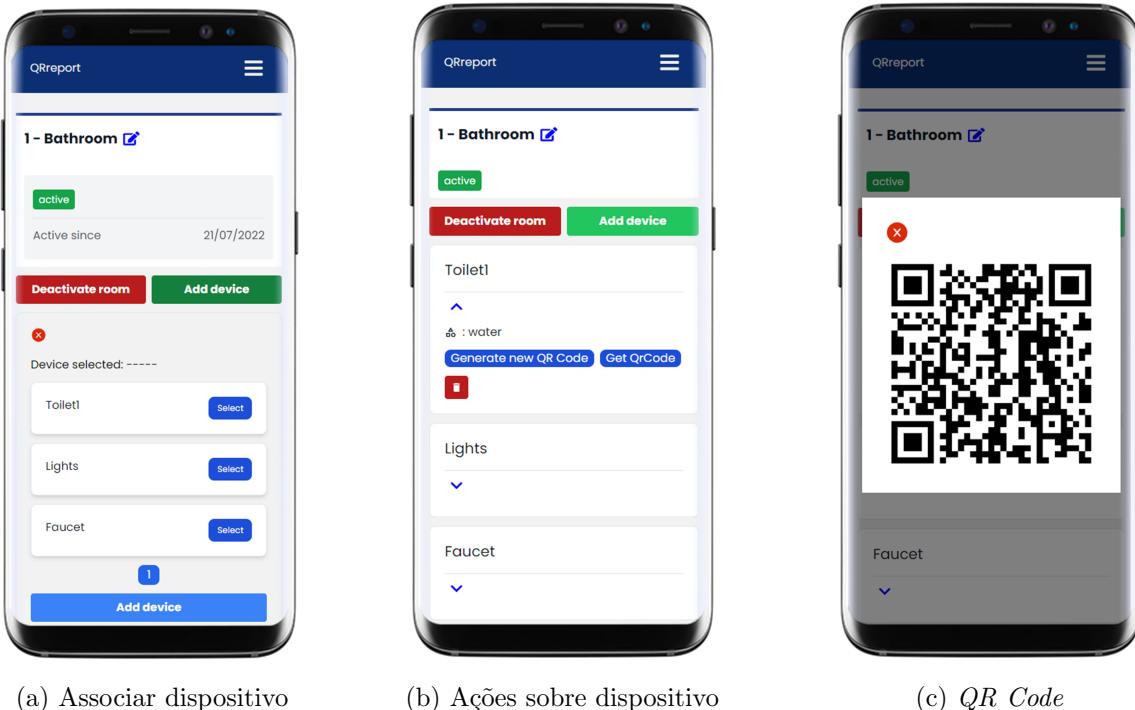
(a) Criar sala

(b) Detalhes da sala

Figura 4.12: Páginas associadas a uma sala

Na página de uma determinada sala, também é possível adicionar um novo dispositivo à mesma. Esta funcionalidade apresenta uma lista ao utilizador com todos os dispositivos que são possíveis adicionar, Figura 4.13 (a).

Uma sala ao possuir um dispositivo, disponibiliza algumas funcionalidades associadas ao mesmo, tais como removê-lo da sala, obter o seu *QR Code*, ou gerar um *QR Code* novo. Na Figura 4.13 (b) são apresentadas estas funcionalidades e na Figura 4.13 (c) o *popup* gerado quando é obtido o *QR Code*.



(a) Associar dispositivo

(b) Ações sobre dispositivo

(c) QR Code

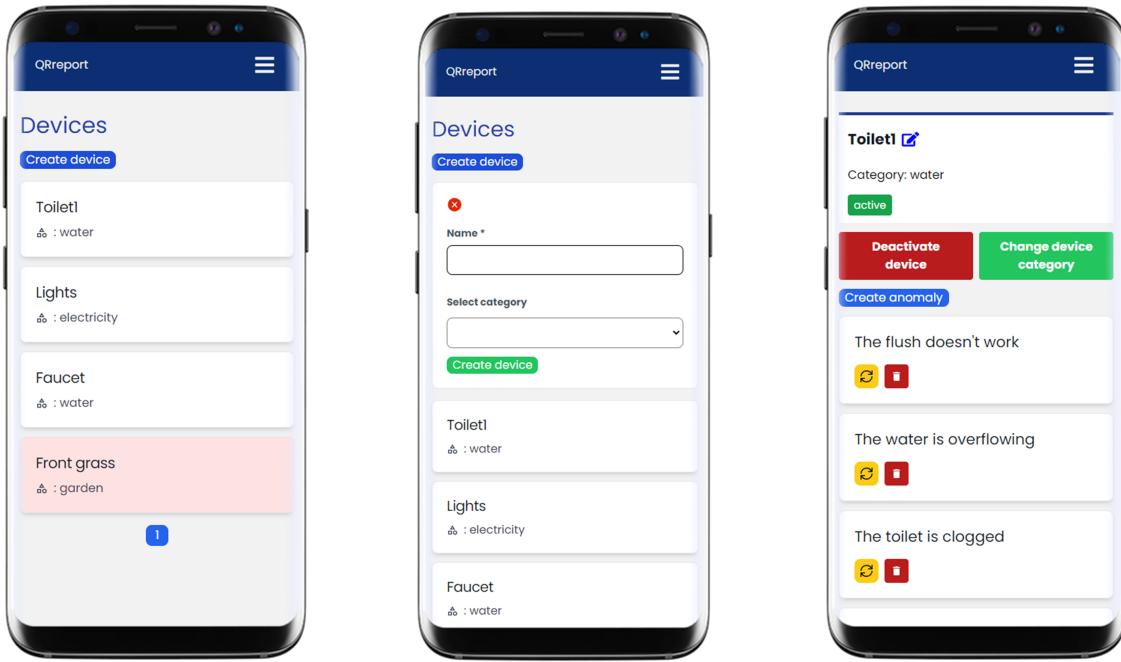
Figura 4.13: Páginas das ações associadas a uma sala

4.4 Dispositivos

A manipulação dos dispositivos presentes no sistema apenas é acessível pelos *admins*. A página dos dispositivos é composta por uma lista que contém todos os dispositivos que se encontram registados no sistema, tal como se pode verificar na Figura 4.14 (a). Na sua página também se encontra a opção para adicionar um novo dispositivo, tal como se pode verificar na Figura 4.14 (b).

A página que tem como intuito apresentar a informação específica sobre determinado dispositivo, apresenta as anomalias associadas ao mesmo, tal como é possível observar na Figura 4.14 (c). Nesta página, os *admins* podem efetuar alterações sobre os dispositivos, tais como atualizar o seu nome (Figura 4.14 (d)) e alterar a categoria ao qual pertence (Figura 4.14 (e)).

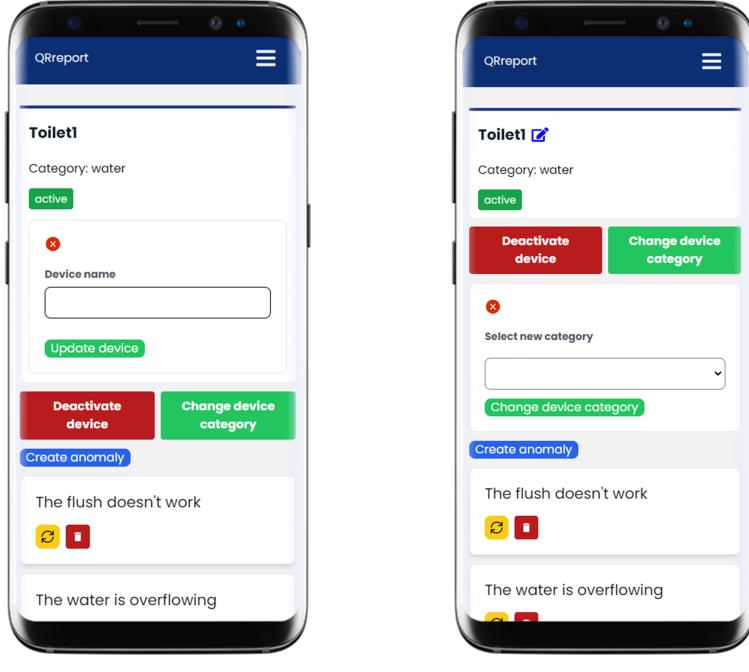
Tal como foi referido, na página de um dispositivo também se encontra a lista das suas anomalias, as quais o utilizador do serviço pode utilizar no formulário do *report* para especificar mais facilmente a anomalia encontrada. Nesta página encontra-se também a opção de adicionar uma nova anomalia ao dispositivo, observável na Figura 4.15 (a).



(a) Listagem dos dispositivos

(b) Criar dispositivo

(c) Detalhes do dispositivo

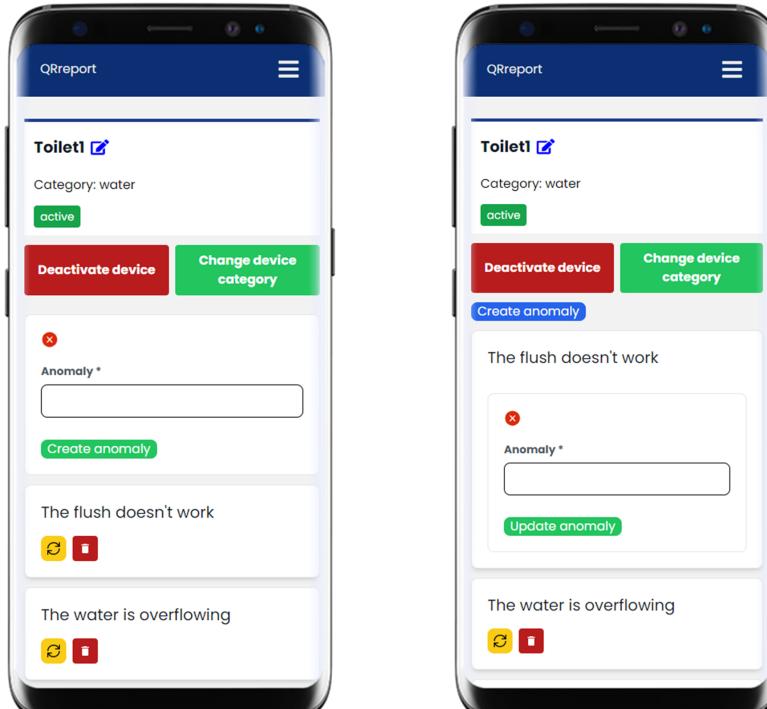


(d) Atualizar dispositivo

(e) Trocar categoria

Figura 4.14: Páginas para alterar *roles*, *skills* e associar uma pessoa a uma determinada empresa

Cada anomalia presente na lista, tem associada as ações que podem ser realizadas sobre a mesma, ou seja, removê-la permanentemente do dispositivo ou atualizá-la, como se pode verificar na Figura 4.15 (b).



(a) Criar anomalia

(b) Atualizar anomalia

Figura 4.15: Páginas associadas às ações de uma anomalia

4.5 Categorias

Para que seja possível atualizar as categorias às quais os dispositivos são associados ou as competências possuídas por determinado funcionário, é indispensável uma página na qual os *admins* do sistema possam manipular as mesmas, página essa representada na Figura 4.16 (a).

Na página das categorias, os *admins* podem proceder à criação de uma nova categoria, tal como é apresentado na Figura 4.16 (b), bem como atualizar as que já se encontram presentes no sistema (Figura 4.16 (c)).

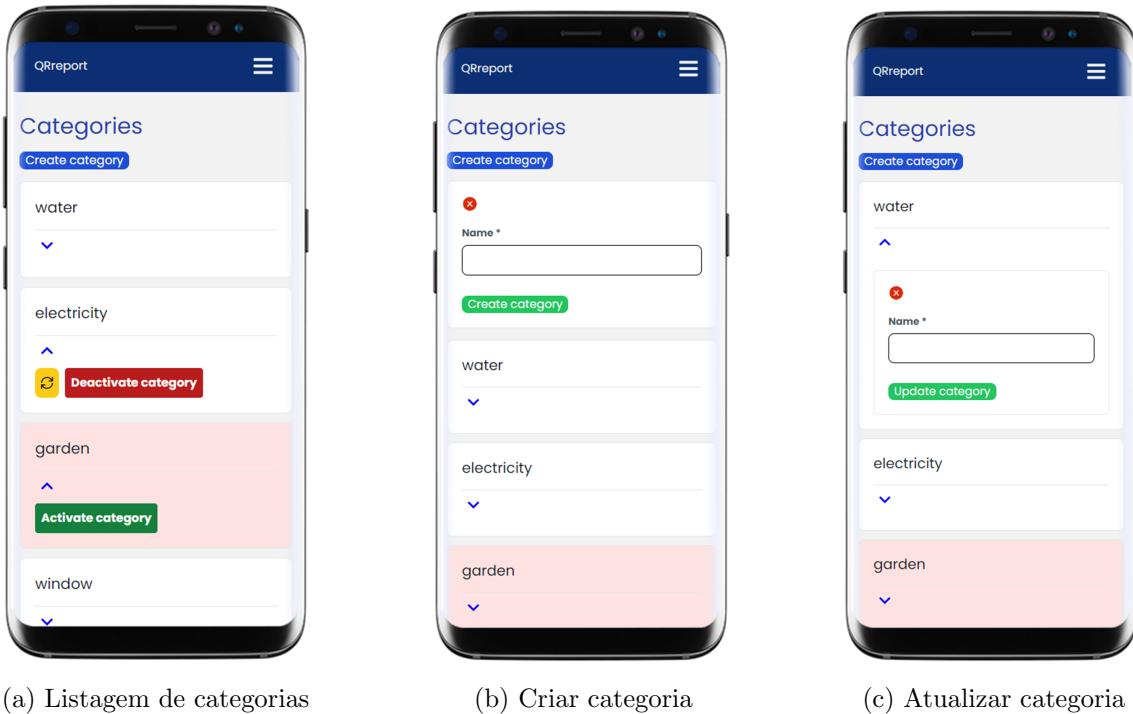


Figura 4.16: Páginas das ações associadas a uma sala

Apesar de terem sido, anteriormente, apenas demonstradas vistas no formato de um telemóvel, é de realçar o facto de todas estas páginas não estarem limitadas à sua utilização nestes dispositivos, podendo o sistema ser também utilizado num *browser* com páginas de maior dimensão, tal como é possível observar nas Figuras 4.17 e 4.18, onde é apresentado o exemplo da página de perfil e da página da listagem dos *tickets* de um *manager*, respetivamente.

Figura 4.17: Página de perfil (com maior dimensão)

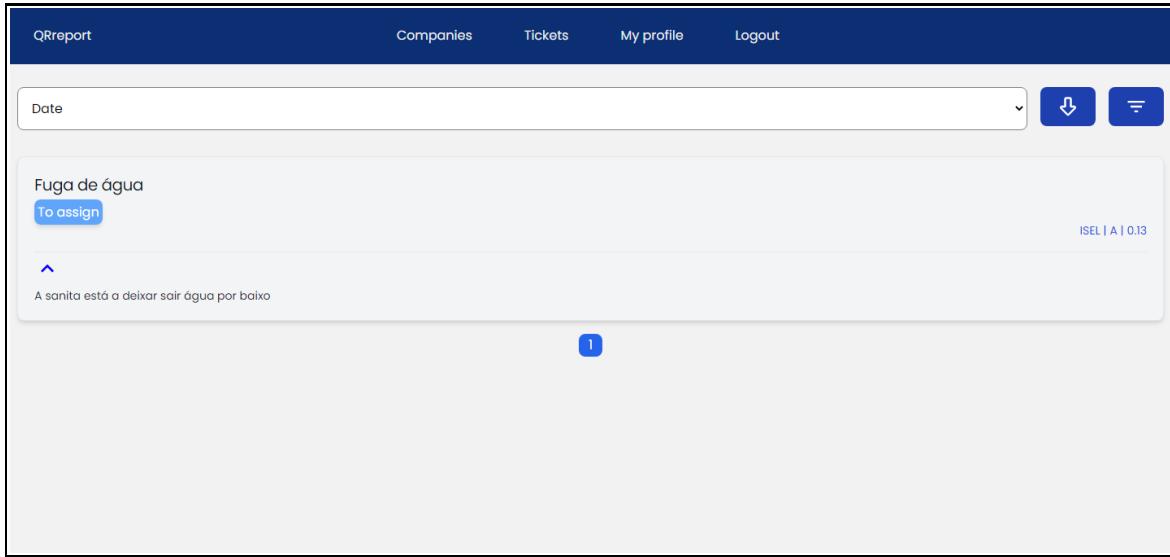


Figura 4.18: Página de *tickets* de um *manager* (com maior dimensão)

Após apresentadas todas as vistas essenciais com dados mais genéricos, de forma a serem demonstradas a grande maioria das funcionalidades disponíveis, no próximo capítulo irá ser descrito o caso de estudo realizado no ISEL, com o objetivo de aplicarmos e testarmos o sistema num caso real.

Capítulo 5

Caso de estudo - ISEL

Neste capítulo será apresentado o caso de estudo realizado no ISEL, com o objetivo de verificar e aplicar o sistema num contexto real, com dados verídicos, para que as funcionalidades e objetivos propostos possam ser validados. Serão referidos os detalhes do que foi realizado para concluir o caso de estudo, em conjugação com as figuras relativas aos processos mais relevantes para demonstrar o correto funcionamento do sistema.

5.1 Dados recolhidos e utilizados

Durante o desenvolvimento do projeto, foi-nos fornecido um ficheiro *Excel* para o desenvolvimento deste caso de estudo onde, num esforço conjunto de algumas professoras do ISEL, averiguaram e registaram um conjunto de anomalias para diferentes dispositivos, sobretudo para casas de banho, uma vez que é lá que se registam os maiores desperdícios de água.

Tendo em posse este documento, foi elaborado um *script* com as informações fornecidas, para popular a base de dados com as mesmas. Foram registados inicialmente os seguintes dados:

- Uma empresa denominada ISEL.
- Dois edifícios, o edifício A e C.
- Três salas correspondentes a casas de banho, salas 0.13 e 0.15 do edifício A e a sala 1.2(1) do edifício C.
- Seis dispositivos.
- Vinte e três anomalias no total, distribuídas pelo número de dispositivos referido.

Foi adicionado um *manager* responsável a cada edifício e foram registados dois funcionários no sistema com as competências necessárias para solucionar as anomalias inseridas.

5.2 Resultados

Após os dados inseridos, foi dado início aos testes das funcionalidades que foram descritas anteriormente. Inicialmente foi feito o *report* de uma anomalia e como tal, um *manager* delegou a sua resolução ao funcionário com as competências para resolvê-lo (Figura 5.1 (a)). O funcionário, à medida que foi reparando o problema foi avançando nos estados possíveis (Figura 5.1 (b)), realizando, por vezes, um comentário no *ticket* para informar o *manager* sobre algum acontecimento (Figura 5.1 (c)).

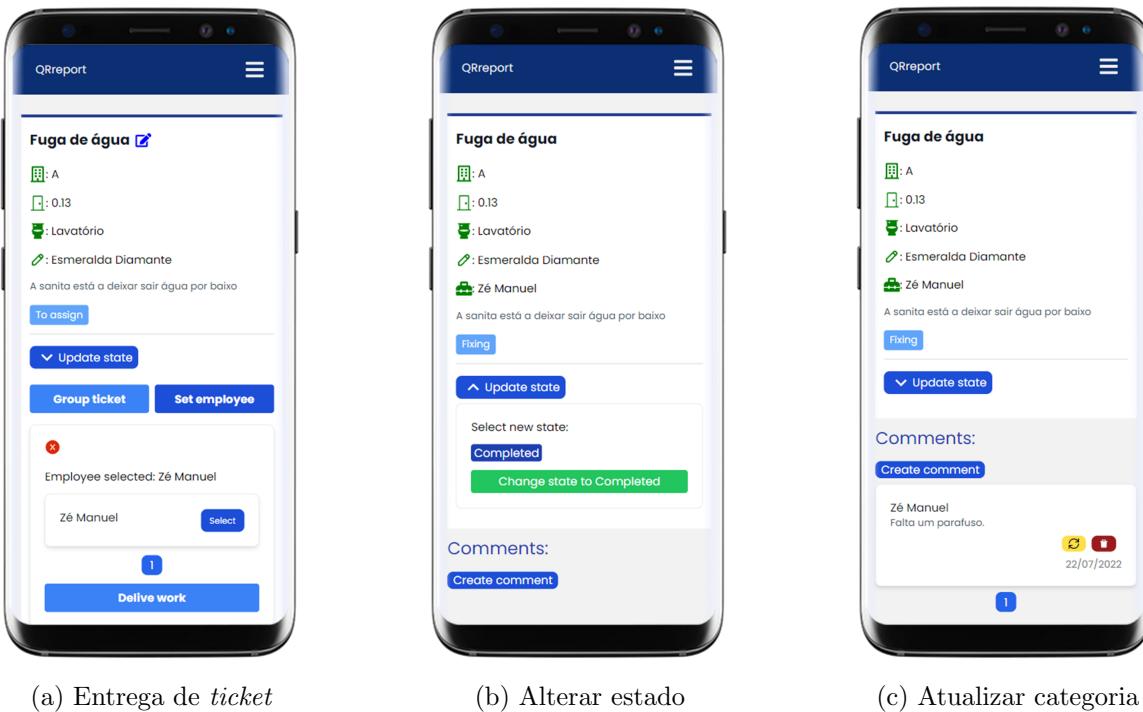


Figura 5.1: Páginas das ações associadas ao *ticket*

Enquanto se estava a desenvolver a reparação, foi submetido um *report* com uma anomalia semelhante à que está a ser reparada, e por isso, o *manager* realizou o seu agrupamento (Figura 5.2 (a)), herdando o *ticket* recente as informações dinâmicas associadas ao seu 'pai', isto é, estados e comentários, sendo assim visíveis em ambos mas apenas atualizáveis no primeiro (Figura 5.2 (b)).

O *ticket* ao ser concluído e arquivado, permite que o utilizador que reportou possa realizar a sua avaliação (Figura 5.3 (a)).

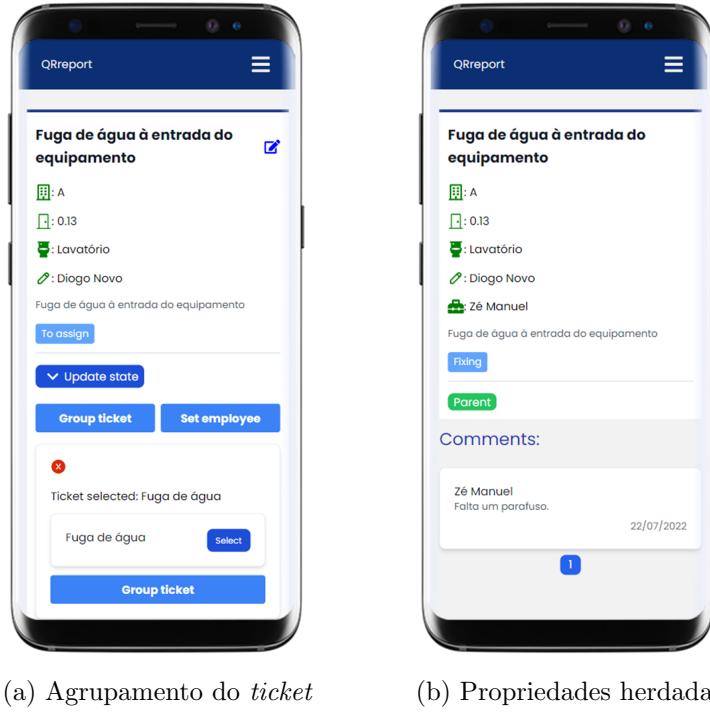


Figura 5.2: Página do agrupamento do *ticket* e o *ticket* agrupado a apresentar as características do *ticket* 'pai'

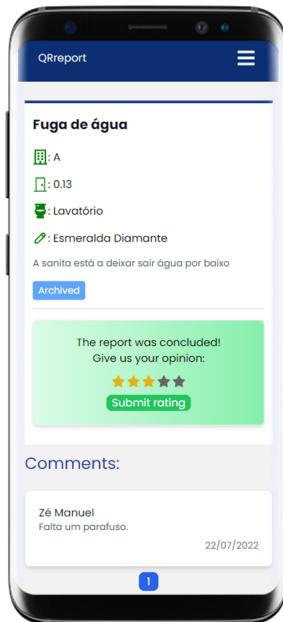


Figura 5.3: Página de submissão de uma avaliação

Chegando ao fim deste caso de estudo, é possível concluir que todas as funcionalidades fundamentais para o correto funcionamento do sistema num contexto real, realizam as suas operações corretamente, sendo as mesmas facilmente utilizadas, tendo, por isso, todo o sistema uma curta curva de aprendizagem relativamente à sua utilização.

Capítulo 6

Conclusão e Trabalho futuro

6.1 Conclusão

O QRreport centraliza todos os problemas inerentes à deteção de um problema e a sua eventual comunicação. Tendo como principais objetivos a facilidade e rapidez para a submissão de um *report* e contribuirativamente para a sustentabilidade do planeta, reduzindo desperdícios comuns a qualquer empresa, tais como a energia, a água e o dinheiro, que corresponde ao gasto económico associado aos restantes desperdícios. Mas, para serem atingidos tais objetivos, é fundamental contar com a ajuda de todas as pessoas que frequentam os espaços para que ativamente reportem os problemas assim que se deparam com eles. Pois só assim, com a responsabilidade social, é que será possível contribuir para um mundo mais sustentável e tirar o máximo do sistema desenhado.

A solução desenvolvida engloba os componentes: *Web API*, sendo o motor principal do sistema, desenvolvida recorrendo à *framework Spring Boot*, agregando também o sistema de autenticação; base de dados relacional PostgreSQL; e uma aplicação *Web*, onde é possível interagir com todas as funcionalidades disponíveis.

6.2 Trabalho futuro

Finalizado todos os requisitos essenciais propostos inicialmente, como trabalho futuro poderia ser implementado os requisitos opcionais que complementariam o sistema, isto é, uma aplicação móvel focada para os funcionários, com o objetivo de não terem de aceder à aplicação *Web*, o desenvolvimento de um algoritmo, recorrendo a técnicas de Inteligência Artificial, de forma a estabelecer medidas preventivas de manutenção de eventuais avarias, evitando, assim, a necessidade de ser sempre reportado manualmente um certo problema periodicamente, como por exemplo, a manutenção anual de um ar condicionado, e também a agregação automática de *tickets* que fossem detetados como fazendo referência à mesma anomalia.

Referências

- [1] DENSO WAVE INC. QR Code. <https://www.qrcode.com/>. Consultado a 02/06/2022.
- [2] What is a REST API? <https://www.redhat.com/en/topics/api/what-is-a-rest-api>. Consultado a 02/06/2022.
- [3] PostgreSQL. <https://www.postgresql.org/>. Consultado a 02/06/2022.
- [4] JSON Standard. <https://www.json.org/>. Consultado a 02/06/2022.
- [5] Framework Spring Boot. <https://spring.io/projects/spring-boot>. Consultado a 02/06/2022.
- [6] Boilerplate code. <https://www.freecodecamp.org/news/whats-boilerplate-and-why-do-we-use-it-lets-check-out-the-coding-style-guide-ac2b6c8>. Consultado a 02/06/2022.
- [7] Kotlin data class. <https://kotlinlang.org/docs/data-classes.html>. Consultado a 02/06/2022.
- [8] Dependência Spring Security. <https://spring.io/projects/spring-security>. Consultado a 02/06/2022.
- [9] Dependência Spring Web MVC. <https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html>. Consultado a 02/06/2022.
- [10] Jdbi API declarativa. https://jdbi.org/#_declarative_api. Consultado a 02/06/2022.
- [11] JDBC. <https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/>. Consultado a 02/06/2022.
- [12] Siren. <https://github.com/kevinswiber/siren>. Consultado a 02/06/2022.
- [13] HTML5 input types. <https://html.spec.whatwg.org/#the-input-element>. Consultado a 02/06/2022.
- [14] HAL. https://stateless.group/hal_specification.html. Consultado a 02/06/2022.

- [15] RFC 6570 URI Template. <https://datatracker.ietf.org/doc/html/rfc6570>. Consultado a 02/06/2022.
- [16] RFC 7807 Problem Details for HTTP APIs. <https://datatracker.ietf.org/doc/html/rfc7807>. Consultado a 02/06/2022.
- [17] JSON Web Token (JWT). <https://jwt.io/>. Consultado a 10/07/2022.
- [18] BCrypt. <https://auth0.com/blog/hashing-in-action-understanding-bcrypt/>. Consultado a 10/07/2022.
- [19] Salt to Hashing. <https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords>. Consultado a 10/07/2022.
- [20] Cookies e syntax. <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers/Set-Cookie>. Consultado a 10/07/2022.
- [21] ReactJS framework. <https://reactjs.org/>. Consultado a 10/07/2022.
- [22] Typescript. <https://www.typescriptlang.org/>. Consultado a 10/07/2022.
- [23] CSS. <https://developer.mozilla.org/pt-BR/docs/Web/CSS>. Consultado a 16/07/2022.
- [24] TailwindCSS. <https://tailwindcss.com/>. Consultado a 16/07/2022.
- [25] React Icons. <https://react-icons.github.io/react-icons/>. Consultado a 16/07/2022.
- [26] React Router. <https://reactrouter.com>. Consultado a 10/07/2022.
- [27] Session Storage no Browser. <https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage>. Consultado a 10/07/2022.
- [28] reCAPTCHA. <https://www.google.com/recaptcha/about/>. Consultado a 16/07/2022.