Mestrado em Engenharia Informática - Computação Móvel

Iniciação do BackOffice do GlarAssist

Paulo Santos Pereira

Leiria, *setembro* de *2022*

Mestrado em Engenharia Informática - Computação Móvel

Iniciação do BackOffice do GlarAssist

Paulo Santos Pereira

Trabalho de Projeto realizado sob a orientação do Doutor Nuno Carlos Sousa Rodrigues, Professor Coordenador da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria.

Leiria, *setembro* de *2022*

Esta página foi intencionalmente deixada em branco

Agradecimentos

A realização do presente projeto de mestrado contou com o contributo e apoio de diferentes pessoas, às quais estou grato.

Ao Departamento de Engenharia Informática e Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria por me terem proporcionado as condições necessárias para realizar o mestrado.

Ao Doutor Nuno Carlos Sousa Rodrigues, pela orientação, disponibilidade e ajuda ao longo deste percurso.

A todos os meus colegas da Glartek com quem tenho o privilégio de partilhar diferentes ideias e desafios diariamente. Um agradecimento especial ao Gonçalo pela oportunidade de desenvolver este projeto tão promissor.

À minha mãe, pelo apoio incondicional, amor e resiliência. Sem ti nunca conseguiria realizar o meu percurso. Obrigada por nunca desistires de mim e da Inês, e por tornares todos os nossos sonhos possíveis. Os três passamos por fases difíceis, mas por tua causa conseguimos tudo o que queríamos.

À Inês, sem a tua ajuda e motivação seria tudo mais difícil. Obrigado por estares presente em todos os momentos e nunca me abandonares.

À Francisca, à minha companheira nas novas aventuras, que sempre acreditou em mim e com quem posso contar sempre. Obrigado por me apoiares e motivares, não só durante este percurso, mas pelos diferentes desafios que vamos experienciando na nossa vida.

Aos meus colegas do Instituto Politécnico de Leiria, que partilharem este percurso comigo, obrigado pela ajuda e partilha ao longo destes anos.

Aos meus amigos, que mesmo afastados, sentia-os de perto.

A todos os que contribuíram neste percurso.

Esta página foi intencionalmente deixada em branco

Resumo

A situação pandémica de Covid-19 aumentou a procura por soluções remotas, uma vez que a mesma obrigou a maioria das empresas a transitarem as suas operações para um ambiente remoto. Por consequência, as soluções remotas ganharam ainda mais relevância nos últimos anos. Para além disso, o trabalho remoto colaborativo tem sido aprofundado para diversos contextos. Uma das abordagens mais investigadas é a assistência remota, que possibilita a assistência de pessoas à distância.

A assistência remota (AR) permite auxiliar uma pessoa ou colaborador remotamente, evitando deslocações, e assim, reduzir o tempo e custos gastos nas mesmas. Um assistente pode observar o contexto do assistido, pela transmissão de vídeo e voz, em diversos dispositivos. Contudo, a assistência remota, apenas por vídeo e voz, não permite resolver a maioria das operações durante a assistência. Para auxiliar as AR é integrada a tecnologia de realidade aumentada (RA). A RA nas assistências remotas proporciona outras formas de realizar a assistência. Uma delas é a introdução de objetos 2D ou 3D ou de ferramentas de desenho no ambiente real do assistido, para indicar ao assistido pormenorizadamente as várias etapas a realizar para concluir as operações da assistência.

A aplicação GlarAssist, desenvolvida pela Glartek, é uma solução de assistência remota com RA que possibilita aos seus utilizadores e organizações realizarem assistências com diversas ferramentas. Contudo, apenas está desenvolvida a funcionalidade da assistência remota e não existe a possibilidade de organizações gerirem os recursos da aplicação, como colaboradores ou assistências. Desta forma, os seus clientes têm vindo a procurar novas funcionalidades.

No presente relatório é apresentado o trabalho realizado no início do desenvolvimento do BackOffice do GlarAssist. Para o desenvolvimento do mesmo foram implementadas funcionalidades que possibilitam aos diferentes utilizadores acederem à informação partilhada na aplicação, e ao mesmo tempo, criarem e gerirem as suas organizações. Assim, foram desenvolvidas onze funcionalidades que são abordadas no presente relatório. Durante o desenvolvimento das mesmas, foram realizados diferentes testes, para validar as funcionalidades antes de serem integradas no ambiente produtivo. Depois de realizados os testes, verifica-se que as funcionalidades foram incorporadas com sucesso no GlarAssist.

Palavras-chave: assistência remota, realidade aumentada, GlarAssist

Esta página foi intencionalmente deixada em branco

Abstract

The pandemic situation triggered by the Covid-19 pandemic has increased the demand for remote solutions, as it forced most companies to move their operations to a remote environment. Consequently, remote solutions have gained even more relevance in recent years. In addition, collaborative remote work has been investigated for different contexts. One of the most researched approaches is remote assistance, which makes it possible to assist people from a distance.

Remote assistance allows assisting a person or employee remotely, avoiding travel, and thus reducing the time and costs spent on it. An assistant can see the context of the assisted, through video and voice transmission, on several devices. However, remote assistance, only by video and voice, does not allow solving most of the operations during the assistance. To help remote assistance, augmented reality technology is integrated. Augmented Reality in remote assistance provides other ways of performing assistance. One of them is the introduction of 2D or 3D objects or drawing tools in the assisted person's real environment, to indicate to the assisted person in detail the various steps to be taken to complete the assistance operations.

The GlarAssist application, developed by Glartek, is an Augmented Reality remote assistance solution that enables its users and organizations to perform assistance with various tools. However, only the remote assistance functionality has been developed and there is no possibility for organizations to manage the application's resources, such as employees or assistances. Thus, its customers have been looking for new features.

In this report the GlarAssist BackOffice is initiated. For its development, functionalities were implemented to enable the different users to access the information shared in the application, and at the same time, create and manage their organizations. Consequently, eleven functionalities were developed that will be addressed in this report. During their development, different tests were carried out to validate the functionalities before being integrated into the production environment. Once performed, it was verified that the functionalities were successfully incorporated into GlarAssist.

Keywords: remote assistance, augmented reality, GlarAssist

Esta página foi intencionalmente deixada em branco

Lista de figuras

Esta página foi intencionalmente deixada em branco

Lista de tabelas

Esta página foi intencionalmente deixada em branco

Lista de apêndices

Esta página foi intencionalmente deixada em branco

Lista de siglas

API	Application Programming Interface
AR	Assistência Remota
AWS	Amazon Web Services
CSS	Cascading Style Sheets
DOM	Document Object Model
HMD	Head-Mounted Displays
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Identity Provider
JSON	Javascript Object Notation
MR	Merge Request
MUI	Material UI
RA	Realidade Aumentada
REST	Representational State Transfer
RM	Realidade Mista
RTC	Real-time Engagement Core
RTM	Real-time Messaging
RV	Realidade Virtual
SAML	Security Assertion Markup Language
SDK	Software Development Kits
SOAP	Simple Object Access Protocol
SP	Service Provider
SQL	Structured Query Language
SSO	Single Sign-On
UI	User Interface
URI	Uniform Resource Identifier
WBS	Websocket

Esta página foi intencionalmente deixada em branco

Índice

1. Introdução

Devido à situação pandémica Covid-19, o trabalho remoto tornou-se importante para que as empresas e os seus colaboradores continuem os seus processos e operações, durante a pandemia. Apesar da colaboração remota ter ganho relevância durante a mesma, esta abordagem já estava a ser explorada para diversas áreas. A comunicação remota pode ser utilizada de muitas e diferentes formas, entre elas a assistência remota (AR). O objetivo da assistência remota é auxiliar uma ou várias pessoas na resolução de problemas, de um lugar remoto, e, assim, evitar que a assistência tenha de ser realizada pessoalmente por especialistas/assistentes. As assistências remotas podem ser utilizadas em diferentes contextos. Atualmente uma das áreas onde a AR é mais explorada é na indústria para a realização de manutenção, apoio especializado no terreno, formação de colaboradores, entre outras atividades. Contudo, a assistência remota tradicional, por vídeo e voz, por vezes, não possibilita a resolução dos problemas, devido às suas limitações, como por exemplo a dificuldade em identificar objetos ou recursos no ambiente pelos utilizadores. Por esse motivo, para ultrapassar algumas dessas barreiras, surgem as assistências remotas através de realidade aumentada (RA). Estas assistências combinam a transmissão de vídeo e voz, com tecnologia de RA, para oferecer diferentes ferramentas para realizarem a assistência remota. A realidade aumentada enriquece a assistência, uma vez que, permite adicionar objetos ou recursos no ambiente real (como por exemplo, setas, texto, entre outros), para auxiliar na resolução do problema. As assistências remotas de RA são uma área cada vez mais explorada, pois permitem a empresas e organizações resolver diversos problemas que surjam nas suas operações, por técnicos, de uma forma rápida e especializada, e ao mesmo tempo, reduzindo custos e tempo gasto em deslocações [1]–[8].

A Glartek, fundada em 2017, é uma empresa com aplicações de realidade aumentada, concentradas na indústria. O seu objetivo é aumentar a eficiência e a segurança dos processos de produção e manutenção. As suas soluções de software combinam a RA e Internet das Coisas (IoT – Internet of Things) para proporcionar visibilidade em todas as operações da organização, otimizar os seus processos e aumentar a segurança nos mesmos. A Glartek tem dois produtos: GlarVision e GlarAssist. O GlarVision utiliza a RA para otimizar o dia-a-dia dos colaboradores, aumentando a produtividade, mobilidade e a segurança durante a realização dos diferentes processos. O GlarAssist é um produto de assistência remota, que utiliza a RA e a videoconferência, para realizar assistências remotas. Contudo, atualmente,

o GlarAssist permite apenas realizar as AR, e por consequência, não permite aos utilizadores armazenar e visualizar as assistências que realizaram anteriormente, assim como gerirem os recursos das suas organizações, entre outras funcionalidades essenciais.

Desta forma, o presente projeto propõe o desenvolvimento de onze funcionalidades, com o intuito de iniciar o desenvolvimento do BackOffice do GlarAssist. Estas funcionalidades irão disponibilizar aos colaboradores e gestores das organizações, ferramentas para gerir utilizadores, assistências remotas efetuadas e toda a informação partilhada na plataforma do GlarAssist. Assim, estas funcionalidades são o começo do BackOffice, permitindo que novas funcionalidades possam ser integradas no futuro. O desenvolvimento das mesmas será realizado em duas aplicações do GlarAssist: aplicação Go[1] (*backend*) e aplicação React[2] (*frontend web*). Para além do desenvolvimento das aplicações, também são abordadas todas as alterações realizadas durante a implementação das mesmas, na base de dados e nos restantes serviços e estruturas do GlarAssist. As funcionalidades propostas são: *chat*, histórico das sessões do utilizador, envio e visualização de ficheiros, organizações, histórico das sessões da organização, filtros nas tabelas de histórico de sessões, alteração do volume do utilizador assistido, gestão de conta dos utilizadores, *feature flags*, lista de contactos dos utilizadores e autenticação centralizada.

Para além do presente capítulo, o relatório conta com mais cinco capítulos para apresentar todo o processo de desenvolvimento da iniciação do BackOffice do GlarAssist. No capítulo *2 - Contextualização das assistências remotas de* RA é realizada uma contextualização das assistências remotas de RA, onde são abordadas as AR, as assistências remotas com RA, a apresentação da aplicação GlarAssist e é realizada uma análise competitiva, onde são comparadas diferentes aplicações de assistência remota com RA. De seguinte, no capítulo *3 - Processo de desenvolvimento*, é exibido todo processo de desenvolvimento das funcionalidades. No mesmo são apresentados os requisitos definidos para a implementação das funcionalidades, a metodologia de desenvolvimento utilizada, a arquitetura do GlarAssist e as tecnologias e serviços utilizados na aplicação. No capítulo *4 - Desenvolvimento das funcionalidades* são exploradas as funcionalidades desenvolvidas no presente projeto, onde são apresentadas todas as alterações realizadas na aplicação *web* e no servidor, durante a implementação. No capítulo seguinte, *5 - Testes e resultados*, são

[1] Go é uma linguagem de programação tipada, compilada estaticamente [32].

[2] React é uma biblioteca JavaScript de *frontend cross-plataform* e *open-source* para construir *user interfaces* baseadas em componentes de UI [44].

abordados os testes realizados em cada funcionalidade, assim como os obstáculos encontrados durante o processo de desenvolvimento. Por último, no capítulo *6 - Conclusão* é efetuada uma conclusão onde é discutido o resultado das funcionalidades desenvolvidas, são exploradas as dificuldades que ocorreram durante a implementação das mesmas, como também, são apresentadas algumas sugestões de trabalho futuro.

2. Contextualização das assistências remotas de RA

No presente capítulo é apresentada uma contextualização das assistências remotas de RA. No mesmo é abordada a tecnologia de realidade aumentada, como também os seus benefícios nas assistências remotas. Para além da contextualização da tecnologia e das AR, é apresentada uma análise competitiva de aplicações de assistência remota com RA, com o foco nas funcionalidades desenvolvidas no projeto, e posteriormente, a aplicação de assistência remota GlarAssist.

2.1. Realidade aumentada nas assistências remotas

A tecnologia de Realidade Virtual (RV) cria um ambiente virtual, em que o utilizador "sente" e experiencia o ambiente como se estivesse realmente presente [9], [10]. Por outro lado, a Realidade Aumentada (RA) refere-se à incorporação de objetos virtuais num ambiente real [1], [5], [7], [9]–[12], criando assim uma ligação entre o ambiente real e o ambiente virtual [2]. Esta ligação entre os dois ambientes pode ser considerada uma Realidade Mista (RM) [7], [10]. Para criar essa ligação a RA deve atender a três requisitos [9]: combinar objetos virtuais e reais num ambiente real; correr interactivamente; e integrar objetos reais e virtuais uns com os outros, em tempo real. A mobilidade na RA torna-se também relevante, uma vez que, é importante interagir com o ambiente e receber informações do mesmo, em qualquer altura e em qualquer lugar [10]. Os objetos ou conteúdos digitais são produzidos por diversos dispositivos móveis e podem ser: imagens, sons, texto, objetos 3D, entre outros [10], [11]. Porém, os objetos virtuais incorporados no ambiente real não herdam todos os atributos do ambiente real (como por exemplo, a sombra) e o tamanho dos objetos virtuais podem não corresponder completamente às características do ambiente real [10]. Com a integração da RA, os utilizadores adquirem a consciência necessária do seu ambiente real, o que aumenta os seus conhecimentos durante a sua atividade, ao exibir informação adicional [10]. Os dispositivos mais utilizados para implementações de RA são os *Head-Mounted Displays* (HMD) e dispositivos portáteis,

como os *smartphones* [7], [10], [13]. Uma das principais características destes dispositivos é a sua portabilidade não ser intrusiva, que pode ser relevante nos diferentes cenários (como por exemplo durante o processo de uma manutenção) [7]. Atualmente, a realidade aumentada atingiu um nível de maturidade que permite a sua utilização em diversos contextos, como por exemplo: industrial [3], [5], [14], [15], agrícola [16], médica [7], [16], [17], aeroespacial e defesa [17], entre outros.

Uma das vantagens da RA é possibilitar a colaboração entre dois ou mais utilizadores [14], [16], permitindo diversas tarefas de colaboração que não eram possíveis com outras tecnologias, como por exemplo, os utilizadores podem orientar remotamente outros utilizadores através do ambiente partilhado, criado pelos dispositivos digitais [14]–[16]. O utilizador perito/assistente pode identificar objetos visualmente no ambiente partilhado pelo utilizador/operador assistido e ao mesmo tempo, fornecer instruções por outros meios, como por exemplo, áudio [4]. Consequentemente, a experiência dos utilizadores torna-se mais personalizada, eficaz e imersiva [4], [16], [18]. Nesta colaboração surgem as assistências remotas com RA.

Quando há uma avaria ou problema, e é necessária a manutenção de algum recurso, geralmente, é preciso o auxílio de um técnico especializado no local [6], [11]. As videochamadas podem ser utilizadas para evitar a deslocação do técnico, realizando assim a assistência remotamente. Contudo, surgem alguns problemas quando um perito quer localizar um determinado pormenor numa máquina ou peça [6], [11], [19]. Nestes casos, a assistência remota com RA proporciona diferentes formas de auxiliar os colaboradores e as empresas no dia-a-dia, quando é necessária manutenção ou reparação especializada [11], [15]. Com a assistência remota é possível realizar assistências em tempo real, proporcionando uma assistência especializada em diferentes locais e organizações [2], [5], [10], [13]. Com diversos dispositivos, o campo de visão de um utilizador/operador, que necessite de assistência, é partilhado a um perito, permitindo-lhe adquirir conhecimentos sobre as condições locais e fornecer instruções verbais e visuais [2], [4], [5], [13], [18]–[20]. Com a assistência remota através de RA, os operadores ou peritos conseguem colocar anotações (recursos de RA) no ambiente real e as mesmas são exibidas no campo de visão do operador e no ecrã do perito, proporcionando assim uma comunicação bidirecional [6], [11], [19], [20]. Com estas anotações, o perito consegue auxiliar por voz ou visualmente com outras anotações, no ambiente real, durante a realização de um procedimento [19]. Existe uma correspondência entre um operador e os peritos remotos, ou seja, um operador pode

solicitar vários peritos remotos que podem fornecer simultaneamente serviços de assistência remota [4]. A AR torna-se assim numa ferramenta interativa, não tradicional, que permite aos participantes criar salas de reunião e juntar-se às mesmas [18]. As aplicações de AR são representadas por três partes [4]: a gestão e comunicação da assistência, é a parte principal da aplicação e é utilizada principalmente para estabelecer o canal de colaboração e gerir os comportamentos de todas as partes; a chamada de vídeo/áudio, é utilizada principalmente para permitir aos peritos remotos observar o local do operador e proporcionar uma comunicação de voz; e a visualização da informação da assistência do operador, que utiliza um algoritmo de registo de rastreio de alvos, para construir um modelo dinâmico de RA, baseado nos dados de assistência fornecidos pelos peritos remotos. Esta tecnologia é cada vez mais relevante [1], [21], visto que é uma forma diferente e mais eficiente de partilhar informação, sem tornar os métodos tradicionais obsoletos [1]. A assistência remota apresenta inovações disruptivas, tais como a eliminação da necessidade de peritos ou assistentes no local e, consequentemente, reduzir os custos e o tempo de viagens e de intervenções [1]–[8]. Para além dos custos reduzidos, o investimento necessário para implementar estas aplicações é baixo, uma vez que, as mesmas podem ser realizadas por dispositivos comuns (*smartphones* ou *tablets*) [3], [20].

Durante o estudo, foram identificados diferentes problemas e funcionalidades essenciais na realização de assistências remotas através de RA. De acordo com a análise realizada [22], foram definidos padrões de melhores práticas e pontos a ter conta, para o desenvolvimento de aplicações de assistência remota de RA. De seguida, são apresentados todos os padrões definidos [22], com os problemas e funcionalidades previamente identificados, correspondentes ao padrão. Desta forma, as práticas e pontos definidos foram os seguintes [22]:

- Configurabilidade [23]: a aplicação tem de estar preparada para diferentes restrições e limitações consoante os contextos e as atividades em que estiver a ser utilizada.
- Largura de banda [7]: possibilitar alterar a qualidade de vídeo e voz da assistência, se não for garantida uma boa conexão à Internet.
- *Delay* de transmissão: ter *delay* baixo é essencial para os utilizadores terem uma boa experiência durante a assistência.
- Segurança [23]: toda a informação partilhada durante a assistência (áudio, vídeo, documentos, entre outros) tem de ser transmitida de forma segura (encriptação dos

dados, autenticação de utilizadores, entre outros), uma vez que, a maioria dos contextos são empresas e os dados são internos.

- Desenho no espaço (2D) [22], [24]: nem todos os dispositivos tem acesso a tecnologias de RA, consequentemente, é importante ter um modo de assistência 2D para a aplicação ser suportada em diferentes dispositivos; transferência de 2D para 3D durante a assistência.
- Permanência das anotações: possibilitar aos utilizadores diferentes modos de desenho de anotações (permanente, onde todas as anotações eram mantidas e ao vivo, onde as anotações tem um intervalo de tempo até serem automaticamente removidas).
- Desempenho [3], [7], [23]: as tecnologias de RA têm um desempenho intensivo nos dispositivos, por consequência, é importante gerir as mesmas para não sobreaquecer ou reduzir rapidamente as baterias dos dispositivos.
- Funcionalidade "mãos livres" [11], [14], [15], [18], [21]: possibilitar que o(s) perito(s) controle(m) toda a assistência e o dispositivo do assistido, uma vez que, existem atividades em que o assistido não consegue utilizar o seu dispositivo.
- Estabilidade na introdução de âncoras no ambiente virtual.
- Restrições de *hardware* [15], [23]: o *hardware* tem de ser robusto e estar preparado para a rotina dos utilizadores.
- Condições ambientais [3], [11], [23]: o ambiente do assistido pode não possibilitar realizar uma assistência eficaz (como por exemplo, quando a iluminação é reduzida, possibilitar utilizar a lanterna dos dispositivos ou quando há muito ruído, ter diferentes formas de comunicar, por exemplo por *chat*).

De seguida, são abordadas as principais conclusões da análise efetuada. A maioria das investigações analisadas foca-se no desenvolvimento ou utilização de uma aplicação de assistência remota com RA, com o objetivo de compreender a experiência de utilizadores, em diferentes contextos. Apesar de ser importante na avaliação de diferentes aplicações, o presente projeto, tem como objetivo, o desenvolvimento de funcionalidades para a iniciação do BackOffice de uma aplicação de assistência remota com RA. Por consequência, os resultados referentes à qualidade das assistências ou experiência dos utilizadores durante a assistência, não está relacionado com o BackOffice das aplicações. No entanto, foram identificados alguns resultados direcionados ao BackOffice e não para as funcionalidades da própria assistência. Desta forma, as conclusões identificadas estão divididas em duas

categorias: resultados relacionados com a experiência de utilizadores e a qualidade da assistência e resultados relacionados com a plataforma na generalidade.

Resultados referentes à experiência dos utilizadores e à qualidade das assistências

A investigação [11] observou que as condições atmosféricas adversas, como o ruído, sujidade e regulamentos de segurança (tais como capacetes, luvas e óculos de segurança), desempenham um papel importante na rotina de trabalho. No entanto, os atuais HMD (mesmo os com foco industrial, como a série Epson ou os Vuzix) não cumprem estes requisitos, uma vez que, não são adequados para a utilização no exterior em condições climatéricas adversas [11]. Também as luvas complicam o controlo dos dispositivos (especialmente os com *touchscreen*) [11].

Por outro lado, os resultados da investigação [7], mostram que a maioria das avaliações recolhidas, de uma escala de 1 a 5, são Bom (4) ou Muito Bom (5), em relação à utilização de assistências remotas com RA. Um dos problemas técnicos identificados foi a impossibilidade de utilizar a câmara frontal do *smartphone* durante a assistência [7].

Foi observado que a ligação remota em conjunto com uma aplicação com compatibilidade extensa, aumenta a mobilidade do assistido no chão de fábrica num ambiente industrial [13]. Um impacto significativo identificado na investigação [3], é que a aplicação de assistência remota com RA melhorou a comunicação com os clientes ao fornecer instruções à distância, isto é, os clientes executaram as tarefas remotas com maior eficiência, diminuindo o tempo necessário para diversas operações e reduzindo assim os custos de intervenção.

O protótipo apresentado, na investigação [14], inclui um conjunto de características, com base nos requisitos definidos pelos seus parceiros do sector industrial, sendo eles: a captação do contexto do mundo real; apoio à comunicação através de áudio, texto, imagem ou partilha de vídeo; utilização de várias características de anotação para aumentar a qualidade das imagens; criação da funcionalidade, passo a passo, para auxiliar um utilizador num problema específico; reutilização de anotações, se o mesmo problema ocorrer noutro equipamento ou com outro membro da equipa; visualização de anotações em ambiente 2D; e integração de notificações para aumentar a consciencialização entre os membros da equipa. Os resultados consideram que o desenho, notas, notificações e a funcionalidade, passo a passo, como as características mais importantes, pela ordem apresentada [14]. Os participantes também identificaram que observar as anotações de RA, alinhadas com o ambiente do mundo real,

era relevante e reconheceram que contribuía para uma melhor compreensão na realização de um determinado procedimento [14]. A aplicação de assistência remota através de RA foi relevante, uma vez que, possibilitou aos peritos remotos colaborarem com os profissionais no terreno, independentemente da sua localização e tempo [14].

A investigação [4], desenvolve uma solução para a realização de uma assistência remota com RA. A mesma reduziu com eficácia as barreiras de comunicação entre as partes e melhorou a eficiência da assistência remota [4]. Os resultados da investigação [6], mostraram que a assistência remota com RA realizada com um *smartphone*, tem potencial para reduzir erros nas atividades de manutenção, uma vez que, é significativamente mais rápida na identificação de componentes, tipos ou variantes de máquinas e tornou mais fácil resolver tarefas mais complexas.

As anotações visuais criadas na investigação [21], foram importantes ao auxiliar a identificação das características uniformizadas, que podem levar tempo a serem descritas verbalmente. A investigação [8] propõe uma abordagem para reduzir o tempo investido por um especialista remoto, ao mesmo tempo que presta apoio aos operadores no terreno através de ferramentas de assistência remota com RA. Esta abordagem teve resultados positivos, uma vez que, a aplicação reduziu significativamente o tempo de intervenção do perito nas diferentes tarefas realizadas [8]. A possibilidade de ter diferentes sessões ligadas umas às outras e restauradas quando necessário, é essencial nestas aplicações [8].

O suporte por assistência remota com RA reduziu o número de erros durante a assistência [22]. As aplicações de AR sem realidade aumentada são uma extensão interessante às aplicações com RA, uma vez que, solucionam diferentes problemas identificados nas aplicações com RA [22]. Por consequência, a empresa parceira do projeto desenvolvido na investigação [22], mostrou interesse em utilizar a assistência remota durante as principais atividades.

As entrevistas realizadas na investigação [23], foram realizadas para identificar efeitos positivos e negativos, riscos e possíveis melhorias utilizando HMD durante tarefas de assistência remota com RA. Na generalidade, todos os colaboradores de manutenção demonstraram ter visto melhorias na atividade através da utilização de HMD [23]. Porém, foram identificados impedimentos [23], como riscos de lesões, um alcance de visão reduzido, perda de orientação espacial e algum desconforto [20]. Comparando os dois meios de comunicação testados, a investigação anteriormente referida revelou que a assistência

com HMD foi classificada como mais eficiente do que com o *smartphone* [20], [23]. Os colaboradores de manutenção que realizaram os testes, demonstraram sentirem-se mais seguros durante o teste com HMD, devido ao feedback recebido e ao controlo especializado fornecido pelo perito [23]. O feedback recolhido dos participantes, mostra que a assistência remota de RA por HMD pode ser utilizada para melhorar o desempenho das tarefas, especialmente por trabalhadores inexperientes [23].

Resultados referentes ao BackOffice das aplicações

Na investigação [8], para cada sessão, a plataforma regista a informação recolhida na fase de resolução de problemas, bem como as instruções fornecidas pelo perito, utilizando as ferramentas RA disponíveis. Por consequência, o conceito de histórico ou documentação de sessão tornou-se importante no desenvolvimento de uma plataforma de assistência remota com RA [8], [22]. Desta forma um utilizador consegue recuperar as instruções de uma sessão anterior para executar um procedimento para o qual já tinha sido assistido, sem necessitar de mais apoio e, assim, reduzir o tempo despendido pelos peritos [8], [22].

Em jeito de síntese, é possível aferir que as assistências remotas de RA melhoraram os procedimentos nas empresas e/ou organizações onde foram utilizadas e, ao mesmo tempo, reduziram os custos das mesmas, devido à diminuição de assistências presenciais. Contudo, há um problema recorrente na maioria das investigações, os operadores reportaram algum desconforto e preocupações a nível de segurança na utilização destes equipamentos durante as suas atividades do dia-a-dia. De seguida é apresentada a análise competitiva realizada, onde são analisadas aplicações, que podem ser consideradas concorrentes ao GlarAssist.

2.2. Análise competitiva

No presente subcapítulo são analisadas aplicações de assistência remota atuais que utilizam realidade aumentada para realizar as assistências. Estas aplicações podem ser consideradas concorrentes à aplicação GlarAssist, uma vez que se baseiam no uso da realidade aumentada para auxiliar e melhorar a experiência dos utilizadores durante a assistência remota. Como são apenas abordadas funcionalidades do BackOffice, no presente relatório, as funcionalidades relacionadas com a qualidade da assistência (como por exemplo: transmissão de vídeo e voz, qualidade da chamada, realidade aumentada, entre

outros) não são consideradas na análise. Contudo, foram definidos alguns requisitos na seleção das aplicações, tais como a realidade aumentada auxilia a assistência remota, a assistência é realizada por um ou mais assistentes e um assistido, existe a autenticação de utilizadores na aplicação e, por fim, o utilizador assistido utiliza um dispositivo móvel, *smart glasses*, HoloLens, entre outros. Ainda assim, foi necessário criar critérios para avaliação das aplicações, sendo eles: se tem uma aplicação *web*; se tem uma aplicação *desktop*; se a assistência suporta múltiplos utilizadores; se possibilita autenticação centralizada; se tem *chat* embutido na aplicação; se permite a partilha de ficheiros; se armazena o histórico das sessões; se possibilita adicionar filtros avançados ao histórico das sessões; se permite a gestão de equipas/organizações; e se possibilita a criação de lista de contactos dos utilizadores. Os critérios *chat* e múltiplos utilizadores apesar de serem funcionalidades relacionadas com a assistência, tornam-se relevantes na apresentação de um histórico das assistências.

Depois da pesquisa realizada, com base nos requisitos definidos, foram analisadas treze aplicações. A pesquisa é apresentada em formato de tabela no Apêndice A. Na Figura 1 é apresentado um gráfico de barras com o número de funcionalidades suportadas pelas aplicações.

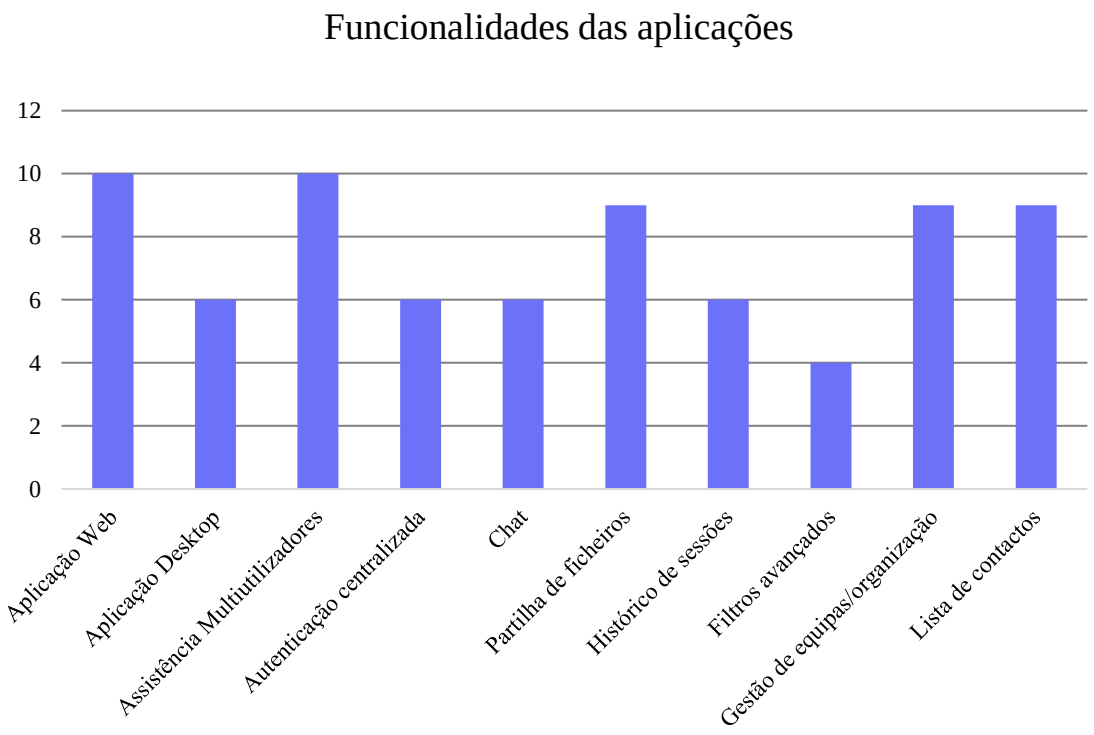


Figura 1 - Gráfico das funcionalidades das aplicações analisadas

Na maioria das aplicações analisadas, as aplicações *web* ou *desktop* são utilizadas pelos assistentes. As mesmas funcionam como aplicações para a função assistente, onde administradores ou técnicos conseguem gerir e realizar as suas assistências. Como é possível observar na Figura 1 a aplicação *web* é a mais comum entre as aplicações. Da mesma forma as funcionalidades de assistência com multiutilizadores, partilha de ficheiros, gestão de equipas/organização e lista de contactos são as funcionalidades mais frequentes nas aplicações. Uma funcionalidade essencial na gestão de organizações é o histórico de sessões, como abordado anteriormente. Como é possível observar a maioria das aplicações não armazena e não apresenta o histórico aos utilizadores. Da mesma forma, o histórico na maioria das aplicações é utilizado como uma ferramenta para os administradores, assim, os restantes utilizadores não conseguem obter informações sobre as assistências que realizaram. Para além do histórico não estar disponível a todos os utilizadores, há históricos de aplicações onde não são apresentadas todas as informações partilhadas durante a assistência, como por exemplo, os ficheiros partilhados, as mensagens do *chat*, entre outras. Uma vez que, o histórico de sessões é uma das funcionalidades menos comuns nas aplicações, é esperado que os filtros avançados seja a funcionalidade menos desenvolvida, visto que, está dependente da funcionalidade do histórico.

Analisando a tabela do Apêndice A é possível observar que apenas a aplicação Dynamics 365 suporta todas as funcionalidades abordadas.

Para além das funcionalidades analisadas existem outras relevantes que não foram abordadas, como por exemplo: chamadas diretas entre utilizadores, convite de utilizadores para as assistências via Email ou SMS, *white label*, entre outras.

Com os resultados abordados nas investigações anteriores [8], [22], e devido à importância da funcionalidade de BackOffice para administradores/gestores, é possível aferir que o histórico das sessões realizadas pelos utilizadores das organizações, com todas as informações partilhadas, torna-se uma funcionalidade fundamental para os contextos empresariais. Segundo a análise competitiva, esta funcionalidade, não é a mais comum nas aplicações, o que diferencia as aplicações em relação à sua utilização em ambientes empresariais e ao mesmo tempo dá relevância às funcionalidades desenvolvidas no presente projeto.

A contextualização e análise competitiva realizada é importante para enquadrar o GlarAssist e as funcionalidades a serem desenvolvidas, com as investigações já realizadas

acerca das assistências remotas de RA e com as atuais aplicações de assistência remota com RA.

2.3. GlarAssist – Aplicação de assistência remota

O GlarAssist é uma aplicação de assistência remota que utiliza a RA nas chamadas de videoconferência para auxiliar a assistência remota. Geralmente, a assistência é constituída por um assistido e por um ou vários assistentes. O assistido é conhecido por cliente, emissor ou utilizador e o assistente por perito, operador remoto ou gestor. Para a realização de uma assistência, o assistido necessita de um dispositivo Android ou *smart glasses* e o assistente necessita de um dispositivo com acesso a um *browser*. O GlarAssist está disponível para *desktops* (através de *browsers*), *smartphones*, HoloLens, *smart glasses*, entre outros.

Para a realização de uma assistência remota, um assistido e um ou vários assistentes, necessitam de se juntar a uma sala. Os utilizadores que podem realizar as assistências podem ser autenticados ou não autenticados (*guest*). Contudo, a criação de salas só pode ser realizada por utilizadores autenticados, enquanto os utilizadores *guest* apenas poderão juntar-se a salas já criadas. As salas são identificadas por um código de nove dígitos numéricos, para facilitar a partilha das mesmas. Durante a assistência só pode haver um utilizador a partilhar vídeo de cada vez (assistido). No entanto, na mesma sala, podem entrar diferentes assistidos. Durante a assistência os utilizadores podem utilizar diversas funcionalidades para resolver os diferentes problemas.

Atualmente, o GlarAssist possibilita aos utilizadores efetuar chamadas por voz e vídeo, a gravação de chamadas, o envio de ficheiros de assistente para assistido, o desenho no plano (2D) e no ambiente virtual criado pela tecnologia de RA, por parte do assistido ou assistente, para auxiliar a assistência, adicionar texto no ambiente virtual, entre outras funcionalidades. Porém, tem algumas limitações. Em primeiro lugar, não é possível armazenar e visualizar métricas sobre as assistências (como por exemplo, o histórico de chamadas, os ficheiros partilhados, as gravações efetuadas, entre outros). Por outro lado, não é possível comunicar por mensagens de texto, assim como não é possível a troca de ficheiros de assistido para assistente (apenas assistente para assistido) e ainda, não é possível gerir a

organização/empresa (visualizar e gerir os utilizadores e as assistências da organização), entre outras funcionalidades essenciais para os utilizadores.

O *backend* do GlarAssist é composto por uma aplicação Go e o *frontend* por duas aplicações, uma aplicação *web* que utiliza a biblioteca React e uma aplicação Android, que não será abordada no presente projeto, uma vez que, o desenvolvimento das aplicações foi apenas realizado no servidor e na aplicação *web*. Para a realização das assistências remotas são utilizadas duas tecnologias em paralelo, o Agora, que possibilita a comunicação de voz e vídeo durante as chamadas, e o Websocket (WBS) para a comunicação entre *backend* e *frontend(s)* durante as chamadas. Ambas as tecnologias são exploradas de forma detalhada mais em diante (3.4).

Relacionando o GlarAssist com os padrões de melhores práticas analisados [22], é possível afirmar que o mesmo cumpre a maioria delas. Contudo, as investigações [8], [22] identificam que um histórico ou documentação é relevante em aplicações de assistência remota com RA e o GlarAssist não possibilita armazenar as informações das assistências. Por este motivo, o desenvolvimento das funcionalidades propostas torna-se ainda mais importante, uma vez que, soluciona este problema identificado, assim como, melhora a aplicação, devido à importância deste tipo de funcionalidades. De seguida é abordado o processo de desenvolvimento utilizado durante a implementação das funcionalidades definidas.

3. Processo de desenvolvimento

No presente capítulo é apresentado o processo de desenvolvimento utilizado para a implementação das funcionalidades requeridas. O mesmo está dividido em quatro partes, sendo elas: a análise de requisitos, a metodologia de desenvolvimento, a arquitetura da aplicação GlarAssist, e por fim, as tecnologias e funcionalidades utilizadas no GlarAssist. De seguida é abordada a análise de requisitos.

3.1. Requisitos

Devido à procura de novas funcionalidades pelos clientes, foram definidos requisitos para o desenvolvimento das mesmas. Os mesmos permitem cumprir todas as necessidades dos clientes requeridas para a iniciação do BackOffce GlarAssist. Posteriormente, estes requisitos foram associados a diferentes funcionalidades. Os requisitos definidos são:

1. Possibilidade de comunicar por mensagens de texto, entre utilizadores, da mesma sala de assistência.
2. Partilha de ficheiros na sala da assistência (seja qual for a função ou permissão do utilizador).
3. Visualizar todas as assistências realizadas.
4. Armazenar e visualizar toda a informação partilhada na assistência.
5. Gestão da conta de utilizador, de forma a adicionar mais informação acerca dos mesmos.
6. Possibilidade de criar e gerir organizações/empresas (utilizadores, assistências, entre outros recursos).
7. Permitir utilizar autenticação centralizada para as contas das organizações.

Com os requisitos propostos foram criadas as funcionalidades necessárias para os solucionar e ao mesmo tempo preparar futuras funcionalidades. No subcapítulo seguinte, é abordada a metodologia de desenvolvimento utilizada pela equipa, para a implementação das novas funcionalidades do GlarAssist.

3.2. Metodologia de desenvolvimento

Durante a integração na Glartek, foi apresentada a equipa de desenvolvimento da aplicação GlarAssist e todo o processo de desenvolvimento. Inicialmente a equipa era composta por, apenas, três elementos, mas durante o desenvolvimento do projeto foi variando entre três e cinco elementos. A equipa durante o processo de desenvolvimento de software segue os valores e os princípios do Agile Manifest, adotando um processo de desenvolvimento Scrum e utilizando um quadro Kanban para apresentar todas as tarefas definidas. De forma a apresentar a metodologia de desenvolvimento o presente subcapítulo foi divido em duas partes. Na primeira parte são apresentados os conceitos e as tecnologias utilizados durante o desenvolvimento das funcionalidades. Na parte seguinte é apresentado o processo de desenvolvimento da equipa.

3.2.1. Conceitos e tecnologias

Agile é uma *framework* conceptual para engenharia de software que começa com uma fase de planeamento inicial e depois é transferida para uma fase de implementação com interações iterativas e incrementais durante o ciclo do projeto [25]. O Agile tem como objetivo reduzir a sobrecarga durante o processo de desenvolvimento de software, dando a possibilidade de alterar o mesmo, sem que o processo esteja em risco ou que necessite de trabalho adicional [25]. Em 2001, dezassete "*software practitioners"* publicaram o "Agile Software Development Manifesto", no qual foram descritos um conjunto de valores e princípios em software e *system agility* [25], [26]. No mesmo, foram definidos quatro valores e doze princípios que sustentam e constituem o Agile [25]. Os valores e os princípios representam uma base para orientar o processo de desenvolvimento de software [25]. Os quatro valores definidos são [25]–[27]: indivíduos e interações sobre processos e ferramentas, software em funcionamento sobre documentação abrangente, colaboração do cliente por cima da negociação de contratos e responder mais à mudança do que seguir um plano.

Scrum é uma *framework* para gestão de projetos, muitas vezes relacionados a software, que possui curtos prazos e requisitos específicos [27]. Os projetos são desenvolvidos por iterações que são chamadas de *sprints* [27]. Um *sprint* geralmente pode durar duas semanas,

durante as quais, devem ser concluídas as atividades definidas [27]. Cada *sprint* começa com uma reunião de planeamento diária onde o Scrum Master e os membros da equipa de desenvolvimento definem o que deve ser alcançado. O Scrum tem diferentes terminologias, sendo elas [27]:

- Equipa Scrum - Não existem funções especificas nas equipas Scrum, visto que os *developers* trabalham em conjunto para concluir a atividade planeada e definida para o *sprint*.
- Product Owner - é o principal *stakeholder* e pode representar os utilizadores, clientes e outras pessoas envolvidas na definição de requisitos, necessidades e recursos.
- Scrum Master - é a pessoa responsável por liderar o esforço geral, monitorizar e liderar a equipa durante todo o processo Scrum. O mesmo deve gerir, recolher os possíveis riscos e evitar possíveis problemas que possam pôr em causa o desenvolvimento da atividade.
- Product Backlog - *user stories*, prioridades e potencias tarefas estão contidas no Product Backlog. Os mesmos podem ser recursos ou requisitos de curto e longo prazo, para uma funcionalidade do produto, que foram definidos em conjunto pelo Product Owner e pela equipa Scrum.
- Reunião de planeamento do *sprint* - A primeira atividade num *sprint* é fazer o planeamento adequado. Nesta reunião, o Product Owner apresenta o Product Backlog à equipa, para permitir que cada membro selecione o trabalho que deseja concluir durante o *sprint*. Esta reunião, geralmente, é a mais longa, em termos de duração, comparando com as reuniões de *sprint*.
- Reunião *stand-up* diária - habitualmente não dura mais de 15 minutos e os membros da equipa podem prestar assistência uns aos outros, se um *developer* estiver preso ou for incapaz de concluir uma tarefa. Todos os membros da equipa, incluindo o Product Owner, são convidados a participar na reunião diária e é recomendado que seja a primeira atividade do dia para resolver rapidamente potenciais problemas.

O quadro Kanban é utilizado para visualizar o fluxo de trabalho e monitorizar o desenvolvimento do projeto, mostrando as atividades do processo de desenvolvimento e mantendo o controlo no trabalho que está a ser realizado [28]. Normalmente, o quadro Kanban é dividido verticalmente em diferentes colunas ou fases. Cada fase refere-se ao estado da tarefa, enquanto cada tarefa é representada, geralmente, por um "cartão" no quadro

na fase correspondente, representando assim o estado atual das tarefas [28]. Os cartões são movidos da esquerda para a direita com base nas mudanças do estado da tarefa [28]. O quadro Kanban tem três fases, sendo elas: *Open*, *In progress* e *Review*. O mesmo facilita o processo de desenvolvimento dos projetos de software, uma vez que, é mais fácil de entender e acompanhar o estado do projeto [28]. No entanto, determinar os limites para cada fase, no quadro Kanban simples, torna-se desafiante para os gestores do projeto. A visualização do quadro limita-se a apresentar as atividades do processo de desenvolvimento, em vez de fornecer informações suficientes ou indicações úteis que possam ajudar a monitorizar o desenvolvimento do projeto [28]. Consequentemente, há a necessidade de identificar critérios alternativos, de forma a melhorar o quadro Kanban, podendo, assim, fornecer outras informações do estado atual dos projetos.

Para o desenvolvimento das funcionalidades é utilizado o Git. O mesmo é um sistema de controlo de versão distribuído para ficheiros, desenvolvido para lidar com diferentes projetos [29]. Possibilita ter várias ramificações (*branches*) que podem ser totalmente independentes umas das outras [29]. Ao inicializar o Git é criado um repositório, representado por uma diretoria, onde vão ser ordenadas e catalogadas todas as versões do produto [30]. A *branch* é criada/usada para iniciar uma cópia separada e semelhante do *workspace* atual para diferentes utilidades. Por outras palavras, separar o estado atual do código para um novo "ramo" onde é possível continuar a desenvolver outras funcionalidades sem modificar o "ramo" principal [30]. Isto permite que os *developers* consigam trabalhar em simultâneo no mesmo projeto, sem causar conflitos durante o desenvolvimento, uma vez que, cada um deles desenvolverá numa *branch* específica. Depois de implementar a tarefa na *branch* é necessário juntar o trabalho desenvolvido a outra *branch*, geralmente, à *branch* principal. A ação de juntar duas *branches* é o *merge* [30]. Antes de se realizar o *merge* é necessário realizar um Merge Request (MR). O MR é um estado antes do *merge* onde normalmente são resolvidos possíveis conflitos e onde outros membros da equipa podem discutir sobre o código desenvolvido [30].

Apesar do código ser importante, o processo de compilação e distribuição do mesmo é essencial no desenvolvimento de software. Consequentemente, é utilizado o Gitlab para o controlo de versões e complementar o processo de desenvolvimento. O GitLab é uma plataforma *web* de DevOps que permite executar diversas tarefas de um projeto, como o planeamento do mesmo, gestão do código, monitorização e a segurança do projeto [31]. A equipa de desenvolvimento utiliza a plataforma para os fluxos de integração e entrega

contínua e para a gestão do projeto com a atribuição de etiquetas a *Issues* (User Stories), definidas nas reuniões de planeamento, que posteriormente são utilizadas no quadro Kanban da plataforma.

3.2.2. Processo de desenvolvimento

Devido ao crescimento da equipa e do produto, houve a necessidade de adicionar passos ao quadro Kanban. O primeiro passo a ser adicionado foi o "Idea" onde são criadas as tarefas e adicionadas as informações referentes à mesma. Esta fase não está só direcionada ao Product Owner, uma vez que, um membro da equipa pode adicionar uma ideia própria. Depois de serem criadas as tarefas, de seguida, são adicionadas ao passo "Story definition". Neste passo são criados os protótipos e definidas as funcionalidades e requisitos referentes à tarefa, pela equipa de produto (representada pelo Product Owner e os restantes elementos). Depois de apresentada aos membros executivos poderia voltar ao mesmo passo, "Story definition", caso houvesse alguns pontos a serem reconsiderados.

Caso a *story* tivesse sido aceite é movida pela equipa de produto para "Approved user story". Antes das reuniões de planeamento do *sprint* os membros executivos (Product Owner e Scrum Master) definem quais as *storys* a serem desenvolvidas no próximo *sprint*. Dessa forma, nessa reunião, as *storys* do "Approved-userstory" eram movidas para "Backlog" ou para um passo adicional "In preparation". As *storys* deste passo são abordadas e discutidas, pela equipa de desenvolvimento, numa reunião esporádica, como irão ser desenvolvidas. Depois de definida a implementação voltaria ao "Backlog". Caso não fosse necessário rever a *story*, na reunião de planeamento do *sprint*, eram movidas para *Open* as *storys* a serem desenvolvidas no *sprint* e continuaria a jornada do Kanban (*Open*, *In progress* e *Review*). Durante as reuniões de planeamento as tarefas são apresentadas à equipa de desenvolvimento (Scrum Master e *developers*) e são realizadas questões à equipa do produto. Durante esta reunião, a discussão é promovida, para que ambas as equipas estejam de acordo com tudo o que foi definido e não surjam contratempos durante o *sprint*.

Durante o *sprint* são realizadas as *stand-ups* diárias, no início do dia de trabalho, com o intuito de dar o feedback da progressão da mesma. Os *sprints*, geralmente, têm um período de uma semana, mas por vezes são alongadas para duas ou três semanas consoante a *story* a desenvolver e o esforço da mesma. No fim do *sprint* é realizado um *Sprint Review* onde cada

developer apresenta à equipa de *developers*, de produto e *marketing* as tarefas que concluiu durante o *sprint*.

Antes de uma tarefa passar para *Review* por um *developer*, o mesmo tem de certificar-se que realizou os testes necessários à funcionalidade para que a mesma esteja conforme o requerido. Quando o Merge Request é criado o código desenvolvido passa por diversos passos onde é validada a qualidade e a sintaxe do mesmo. Para além desses testes, quando uma tarefa é colocada em *Review* e não existem problemas nos testes anteriores, um elemento da equipa, Tester, certifica-se de realizar testes no Merge Request criado e na *branch* principal quando a mesma é *merged*. Com este passo adicional é garantida a qualidade na funcionalidade, permitindo uma redução nos problemas de integração ou outros comportamentos inesperados detetáveis apenas em situações específicas.

No Apêndice B é apresentado o cronograma definido para o desenvolvimento de todas as funcionalidades. No mesmo é possível observar que os *sprints* vão variando entre duas a quatro semanas. Apesar de não ser adequado trocar regularmente o número de semanas dos *sprints*, a quantidade de *developers* da equipa do GlarAssist é pequena o que impossibilita fazer a divisão das tarefas por vários *developers*. Consequentemente, os *sprints* são mais longos, uma vez que, o desenvolvimento do *backend* e *frontend* (*web*) eram, geralmente, realizados sempre pelo mesmo *developer*.

Todas as aplicações do produto GlarAssist estão no mesmo repositório, *backend* e ambos os *frontend* (aplicação *web* e *mobile*). Uma das principais vantagens desta abordagem é proporcionar apenas um fluxo de integração e entrega contínua ao lançar todas as plataformas de uma só vez. Contudo, esta abordagem causa muitos conflitos. Quando há a necessidade de realizar um *merge*, ocorrem muitos conflitos e muitas das vezes no código de outras aplicações (código desenvolvido por outros *developers*), que requer o auxílio de outros *developers*, interrompendo assim as tarefas dos mesmos.

3.3. Arquitetura da aplicação

Como referido anteriormente, o GlarAssist é um produto dividido em três aplicações: aplicação *web*, Android (*frontend*) e aplicação Go (*backend* ou servidor). Para além das aplicações, são utilizados diversos serviços e tecnologias. Para a implementação das novas funcionalidades, a arquitetura foi alterada durante o desenvolvimento, para que seja possível cumprir todos os requisitos e funcionalidades definidas. Consequentemente, a arquitetura apresentada representa o estado final da arquitetura (depois de todas as alterações efetuadas). A mesma é apresentada na Figura 2. Todas as tecnologias e funcionalidades apresentadas na arquitetura são abordadas e exploradas no subcapítulo seguinte (3.4).

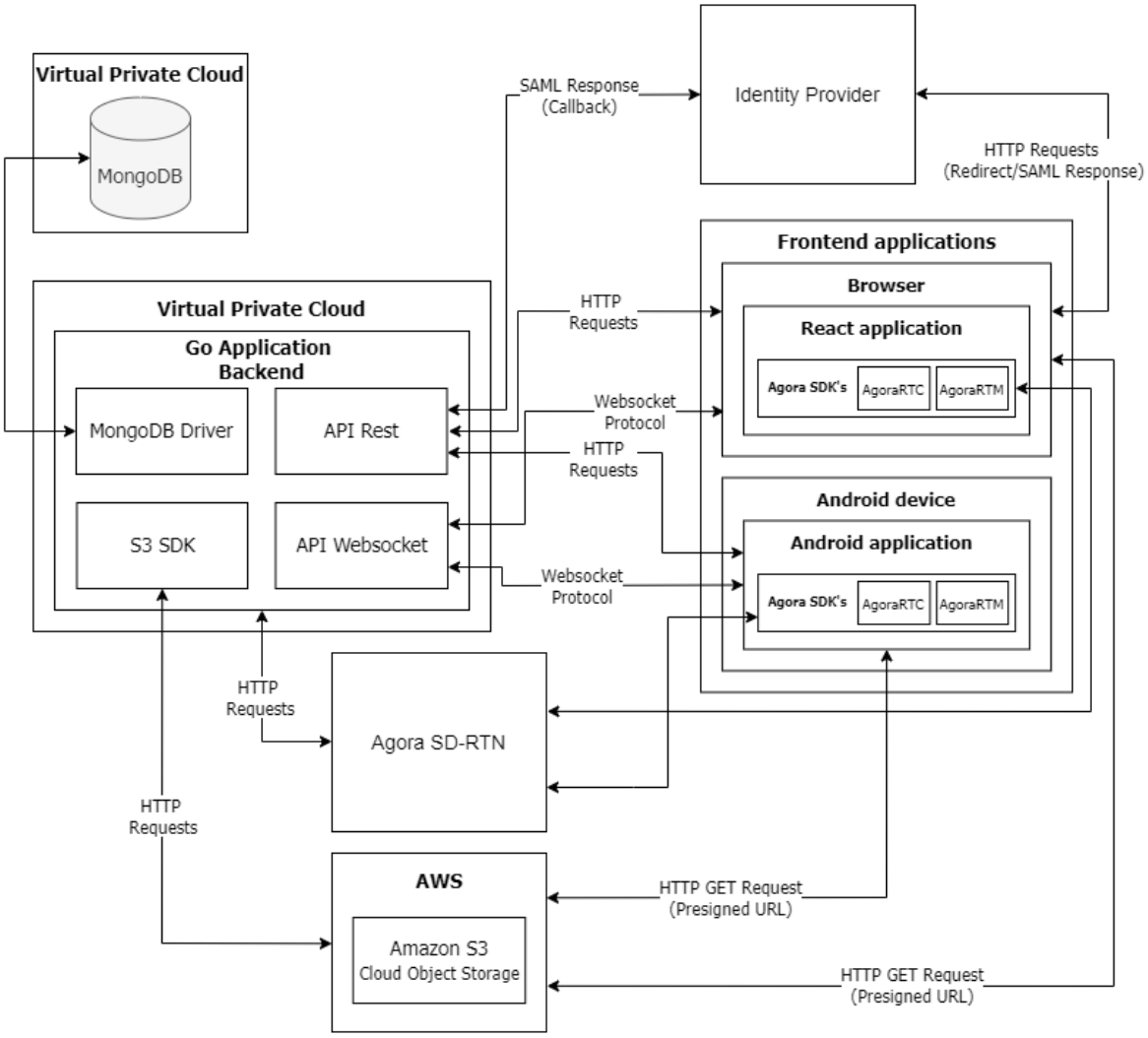


Figura 2 - Arquitetura do GlarAssist

O *backend*/servidor do GlarAssist é uma aplicação Go. Consequentemente, possibilita que todas as aplicações *frontend* consigam efetuar as assistências, possibilitando recolher e visualizar toda a informação da aplicação. Foram definidas diferentes estruturas na aplicação, para a identificação das diferentes entidades da aplicação, como por exemplo a sessão, a sala, o utilizador, entre outras. Para o armazenamento dos dados a aplicação utiliza uma base de dados MongoDB. Para além dos dados persistentes, ainda é necessário armazenar dados temporários, como por exemplo o estado dos utilizadores durante as sessões, o estado do microfone, qual o dispositivo, qual a cor que o utilizador está a utilizar para desenhar, entre outros. Esta informação é armazenada numa estrutura em *cache* na aplicação. O *backend* para além de gerir toda a lógica do GlarAssist, utiliza e gere outros serviços, para complementar as diversas funcionalidades. Um desses exemplos é o serviço Agora. O Agora permite a realização das videoconferências, ou seja, a transmissão de vídeo e voz entre utilizadores. Por consequência, o *backend* responsabiliza-se pela criação de *tokens* para a utilização dos diversos serviços do Agora, gestão das gravações de sessões, entre outras. Outro dos serviços utilizados é o AWS S3. O S3 é um serviço de armazenamento *Cloud* onde são guardados os ficheiros partilhados na aplicação. Com este serviço, o servidor gere os ficheiros, guardando-os numa forma estruturada e gerindo o acesso aos mesmos, consoante os utilizadores. Porém o *download* e *upload* de ficheiros é realizado por *presigned urls*. Estes *urls* são retornados às aplicações clientes para obterem ou enviarem os ficheiros, consoante os dados fornecidos. Para a autenticação centralizada foi utilizado o SAML para a comunicação com as plataformas dos utilizadores. A informação das plataformas é armazenada na organização correspondente, para que os utilizadores da organização consigam utilizar a plataforma da mesma. Para além dos serviços e funcionalidades mencionados anteriormente, ainda são utilizados outros que não irão ser explorados. A comunicação entre servidor e as aplicações *frontend* pode ser realizada por dois métodos: a API REST, utilizada principalmente para retornar e atualizar os dados da plataforma e o Websocket que, atualmente, é apenas utilizado para auxiliar a comunicação entre servidor e cliente ou entre clientes durante as assistências.

A aplicação *web* é utilizada como o BackOffice do GlarAssist, por esse motivo, durante as assistências o utilizador assume o papel de assistente/perito. A aplicação utiliza a biblioteca React, com a linguagem Typescript. Para o desenvolvimento das *user interfaces* (UI) são utilizados os componentes da biblioteca Material UI. Como a aplicação está direcionada a assistentes, para além da realização de assistências, todas as funcionalidades

de gestão de sessões e de organizações foram desenvolvidas na mesma. Estas funcionalidades permitem aos assistentes/administradores acederem às informações que foram registadas durante as sessões da organização (os participantes, as mensagens, os ficheiros partilhados, entre outros). Para além dos administradores, os outros utilizadores têm acesso ao seu histórico pessoal de assistências, onde podem observar a informação partilhada nas mesmas. Para a realização das assistências e apresentação de todos estes dados, são utilizados os dois métodos de comunicação do servidor, a API REST e o Websocket. A API Websocket é utilizada durante as assistências para a comunicação entre aplicações *frontend*, tendo sempre o *backend* como intermediário. Durante as assistências, também é utilizado o Agora RTC e RTM para a transmissão de vídeo e voz. Para além da comunicação com o servidor, a aplicação *frontend* utiliza o serviço AWS S3 para o *download* e *upload* de ficheiros. Esta ligação é efetuada pelos *pre-signed urls* retornados pelo *backend*.

A aplicação Android, por outro lado, pode assumir o papel de assistente ou assistido. A mesma não tem as funcionalidades de assistente que a aplicação *web* tem (históricos e gestão de organizações e utilizadores), mas possibilita realizar assistências com o papel de assistido. A aplicação Android comunica com as restantes aplicações da mesma forma que a aplicação *web* (API REST e Websocket). Na assistência são utilizadas diferentes tecnologias, como o ARCore, para a integração da tecnologia de RA no GlarAssist. Contudo, esta aplicação não irá ser explorada no presente relatório, como referido anteriormente.

A arquitetura foi expandida de forma a possibilitar o desenvolvimento das funcionalidades requeridas. As principais alterações foram: a integração do serviço da AWS S3 e a comunicação com as diferentes plataformas de autenticação das organizações, pelo SAML, para possibilitar a autenticação centralizada aos utilizadores das organizações. No início do desenvolvimento do projeto nenhum destes serviços/funcionalidades estavam a ser utilizados, e por consequência, foram integrados durante o desenvolvimento das funcionalidades. No próximo subcapítulo são abordadas e exploradas todas as tecnologias e serviços utilizados no GlarAssist e na realização do projeto.

3.4. Tecnologias e serviços

No presente subcapítulo são abordadas todas as tecnologias e serviços utilizados no GlarAssist e no desenvolvimento das funcionalidades solicitadas. A maioria dos serviços que são apresentados foram identificados na arquitetura, no subcapítulo anterior. A apresentação dos mesmos é feita em três partes. Na fase inicial são abordadas todas as tecnologias e serviços utilizados no servidor. A segunda fase é direcionada para as tecnologias e serviços da infraestrutura da aplicação do GlarAssist. Por fim, na última fase, são abordadas as tecnologias e serviços utilizados na aplicação *web*.

3.4.1. Servidor

A Go, desenvolvida pela Google, conhecida também por Golang, é uma linguagem de programação tipada, compilada estaticamente [32]. Esta linguagem facilita a criação de serviços e aplicações devido às diferentes ferramentas e API fornecidas pelos seus serviços de *Cloud*. As principais vantagens desta linguagem são: *open-source*, facilmente gerida a concorrência da aplicação, gestão de memória feita automaticamente, existência de documentação referente à linguagem, linguagem rápida e pode ser compilada em diferentes plataformas (Windows, Android, Linux, entre outros) [32]. Esta linguagem pode ser utilizada para desenvolver diversas aplicações, tais como, serviços de *Cloud* e *Network*, *interfaces* de linha de comandos, aplicações *web*, entre outras [33].

Para que o servidor consiga fornecer a informação às aplicações cliente são utilizadas duas API. Uma API (Application Programming Interface) é um conjunto de definições e protocolos utilizados para desenvolver e integrar diferentes aplicações [34]. A API permite que um fornecedor (*provider*) de informações e um utilizador comuniquem, fornecendo o conteúdo exigido pelo consumidor (a chamada) e o conteúdo exigido pelo produtor (a resposta) [34]. Consequentemente, a API funciona como um mediador entre utilizadores e recursos ou serviços *web* que os utilizadores pretendem obter [34]. Com a API a informação pode ser exposta aos seus utilizadores numa forma segura, controlada e com a devida autenticação assegurada [34]. As API podem ter quatro políticas diferentes [35]: privada, compartilhada, pública ou composta. Para além de políticas diferentes, as API podem utilizar diferentes protocolos [34], [35]. Estes protocolos foram desenvolvidos com o objetivo de

fornecer aos utilizadores um conjunto de regras pré-definidas, que especificam os tipos de dados e as operações/comandos aceites [35]. Alguns dos protocolos são [35]: SOAP (Simple Object Access Protocol), XML-RPC, JSON-RPC, REST (Representational State Transfer), entre outros. Os protocolos API utilizados na aplicação GlarAssist são: REST e o Websocket.

A API REST ou RESTful é um conjunto de princípios de arquitetura de API *web*, por esse motivo, o mesmo não é considerado um protocolo ou padrão [34], [35]. Para ser uma API REST, basta a interface obedecer a certas restrições arquitetónicas e, por consequência, a mesma pode ser implementada em diferentes abordagens. Quando um utilizador realiza um pedido por uma API RESTful, a mesma transfere uma representação do estado do recurso ou da ação para o *endpoint* solicitado. Esta informação é retornada, por HTTP, num dos diversos formatos: JSON, HTML, XLT, Python, PHP ou texto simples [34]. O JSON é o formato mais utilizado porque é independente da linguagem (Javascript) e é legível pelos humanos e/ou aplicações. Para além da resposta, os cabeçalhos e os parâmetros também são importantes quando é realizada uma solicitação à API. Existem cabeçalhos de pedidos e cabeçalhos de resposta, cada um tem as suas próprias informações de conexão HTTP e códigos de status [34].

O HTTP não foi desenvolvido para a comunicação de longa duração e em tempo real, entre duas aplicações [36]. Uma das abordagens para realizar estas conexões é a API Websocket. O Websocket é um protocolo que permite que uma aplicação *frontend* e uma aplicação servidor comuniquem usando uma conexão *full-duplex* (comunicação ponto a ponto, composta por duas ou mais partes que podem comunicar entre si em ambas as direções). O Websocket utiliza uma sequência de padrões pedido-resposta HTTP para estabelecer uma conexão [36]. Quando a conexão é estabelecida, a API Websocket fornece uma *interface* para o acesso e gestão de informação, pela conexão estabelecida *full-duplex* assíncrona [36]. O Websocket também fornece uma *interface* para fechar de forma assíncrona a conexão em ambos os lados [36].

3.4.2. Infraestrutura

Para o armazenamento dos dados é utilizada uma base de dados MongoDB. A MongoDB, desenvolvida por MongoDB Inc., é uma base de dados (BD) de documentos multiplataforma escalável e flexível [37]. Identificada como uma base de dados NoSQL, o MongoDB usa documentos semelhantes a JSON. Por consequência, os campos podem variar de documento para documento, na mesma coleção (referida como tabela numa base de dados SQL), e uma estrutura de dados pode ser alterada ao longo do tempo [37].

Numa base de dados relacional, em comparação, os dados são armazenados em linhas e tabelas, organizados de diferentes maneiras. As mesmas possibilitam o armazenamento e recuperação de dados, como também lidar com grandes quantidades de dados e consultas complexas [38]. A integridade dos dados é conseguida, uma vez que, cada informação é armazenada num único sítio, não existindo assim problemas com versões dos dados diferentes [38]. A linguagem normalmente utilizada nestas bases de dados é a SQL (Structured Query Language) [38].

Quando uma base de dados é NoSQL significa que é uma BD não relacional. Isso significa que podem existir documentos com diferentes estruturas na mesma "tabela", tornando-se assim mais flexível, ajustando o formato dos dados consoante o requerido [38]. Embora o SQL seja melhor para garantir a viabilidade dos dados, o NoSQL torna-se essencial quando a disponibilidade é prioridade, o conjunto de dados está em constante mudança ou é necessário trabalhar com modelos de dados flexíveis [38].

Para a realização das videoconferências foi necessário a integração de um serviço externo. Por consequência, foram integrados diversos serviços da Agora. A mesma é uma plataforma que fornece um conjunto de soluções de software para a comunicação de vídeo, voz e transmissão interativa ao vivo, para a interação com outras pessoas remotamente [39]. A plataforma tem diferentes API e Software Development Kits (SDK) que facilitam a integração de funcionalidades de áudio, vídeo, mensagens e gravação dos mesmos [39]. Os produtos principais da Agora são: RTC (Real-time Engagement Core) que possibilita a interação entre utilizadores por vídeo e voz e o RTM (Real-time Messaging) que possibilita a interação entre utilizadores por texto. Para além dos produtos principais tem: *Cloud* Recording, permite a gravação das interações por vídeo e voz; Agora Analytics, ferramentas

de análise de todas as chamadas efetuadas, com métricas de todos os produtos; entre outros produtos [39].

No armazenamento de recursos, como os ficheiros partilhados nas assistências, é utilizado o Amazon Simple Storage. A Amazon Web Services (AWS), assessora da Amazon, é a plataforma *Cloud* com diversos serviços e recursos *on-demand* e API para o uso pessoal, empresarial, entre outros [40]. Os serviços da AWS são entregues aos clientes por uma rede de servidores da AWS localizados em todo o mundo. Os mesmos são tecnologias de infraestrutura, como de processamento, armazenamento e base de dados ou tecnologias como Machine Learning, inteligência artificial e IoT [40]. Um dos diversos serviços da AWS é o Amazon Simple Storage Service ou Amazon Web Services S3 (AWS S3). O mesmo é um serviço de armazenamento de objetos que oferece escalabilidade, disponibilidade e segurança [41]. O AWS S3 armazena dados como objetos dentro de *buckets*. Um objeto é formado por um arquivo e, opcionalmente, os metadados que descrevem esse arquivo [41]. Ao carregar um arquivo, é possível definir permissões no objeto e os metadados necessários. Os *buckets* são os *containers* para os objetos. Como os objetos, nos *buckets* também podem ser definidos diversas definições [41], como por exemplo: gerir o acesso e as permissões do *bucket*, alterar a sua região, entre outras.

A Security Assertion Markup Language (SAML) é utilizada para a proporcionar o Single Sign-On (SSO) aos utilizadores na aplicação do GlarAssist. O mesmo permite que um Identity Provider (IP) autentique utilizadores e, de seguida, passe um *token* de autenticação para outra aplicação conhecida como Service Provider (SP) [42], [43]. A SAML permite que o SP opere sem ter de realizar a sua própria autenticação e passar a identidade, para integrar utilizadores internos e externos [42]. A mesma permite que as credenciais de segurança sejam compartilhadas com um SP numa rede, normalmente uma aplicação ou serviço. A SAML envia informações sobre os utilizadores, *logins* e atributos entre o IP e o SP [42], [43]. Cada utilizador autentica-se uma vez num IP e pode estender a sua sessão de autenticação de forma transparente para várias aplicações [42], [43]. O SP solicita a autorização e autenticação do utilizador [42], [43]. Com a SAML é possível realizar o SSO, em aplicações que suportem o protocolo e serviços SAML [42], [43]. O objetivo do SSO, para o utilizador, é permitir que seja necessário apenas uma autenticação e obtenha acesso a sistemas protegidos separadamente sem reenviar credenciais [42].

3.4.3. Aplicação *web*

React, também conhecido como React.js ou ReactJS, foi criada em 2013 pelo Facebook (atualmente Meta) e é uma biblioteca JavaScript de *frontend cross-plataform* e *open-source* para construir *user interfaces* baseadas em componentes de UI [44]. O React pode ser utilizado como base no desenvolvimento de aplicações *single-page*, móveis (com o React Native) ou renderizados com *frameworks* como Next.js [44]. No entanto, o React apenas gere e procede à renderização do estado para o DOM (Document Object Model), por consequência, durante a criação de aplicações React, geralmente, requerem o uso de bibliotecas adicionais [44].

A linguagem utilizada na aplicação *web* é a Typescript. A mesma é um *superset* (linguagem de programação que contém todos os recursos de uma determinada linguagem e foi expandida ou melhorada para incluir outros recursos) sintático do Javascript e adiciona uma tipagem estática (opcional) à linguagem [45]. Como é um *superset* de Javascript, as aplicações de Javascript também são válidas em Typescript [45].

Para a criação das UI da aplicação *web* são utilizados componentes da biblioteca do Material UI (MUI). A mesma é uma biblioteca de componentes React UI que implementa o Material Design [46]. A biblioteca inclui um conjunto de componentes pré-construídos para serem integrados nas aplicações [46]. Esta biblioteca é utilizada em aplicações *web*, tendo suporte na maioria dos *browsers* [46]. Para além de apenas poder ser executada em aplicações React, o MUI necessita também da linguagem Typescript, uma vez que, segue a política de tipagem da mesma [46].

Depois de ser abordado todo o processo de desenvolvimento utilizado, durante a implementação das funcionalidades, de seguida, são apresentadas as funcionalidades desenvolvidas, como também, todas as alterações realizadas nas aplicações.

4. Desenvolvimento das funcionalidades

De seguida são abordadas as funcionalidades (*features*) desenvolvidas. Na maioria das funcionalidades foi necessário o desenvolvimento das mesmas, no servidor e na aplicação React. Por consequência, são apresentadas com detalhe, para cada funcionalidade, todas as alterações realizadas em ambas as aplicações, indicando o esforço de cada alteração e o motivo das mesmas. No desenvolvimento realizado no servidor, todas as criações, atualizações ou remoções de modelos, pedidos e eventos implementados, também são apresentados. Tal como para a aplicação *web*, também são exibidas as vistas e os componentes criados no desenvolvimento das UI. Da mesma forma, as alterações na estrutura da BD, a integração ou utilização de serviços, entre outros, também são abordados.

A UI das funcionalidades desenvolvidas foi criada pela equipa de design da Glartek (membros da equipa de produto). Quando as funcionalidades foram apresentadas, toda a UI para a funcionalidade já estava desenvolvida, como também a jornada da mesma. Na página das *features* no Gitlab é adicionada a sua jornada, assim como os *mockups* da UI e outras informações relevantes. No entanto, depois de serem apresentadas as funcionalidades, ainda poderiam ser alteradas, devido a diversos problemas, como por exemplo, algum caminho da jornada não ser o mais correto, não ser possível implementar o requerido, entre outros. Quando necessária a alteração de alguma funcionalidade, era realizada uma reunião esporádica com os membros envolvidos (equipa de desenvolvimento e de produto) na funcionalidade, para expor os problemas e as soluções sugeridas pela equipa de desenvolvimento. É importante referir que a UI teve em constante alteração, uma vez que, outros *developers* a foram alterando noutras *user stories*. Devido a estas constantes alterações os componentes podem ser diferentes ao longo das figuras ou *mockups* que são apresentados. Porém, os *mockups* apresentados em cada funcionalidade são sempre os *mockups* finais da *user story* definida para a funcionalidade.

Antes de apresentar as funcionalidades desenvolvidas, é importante contextualizar alguns termos e conceitos utilizados no GlarAssist, sendo eles:

- User – Esta estrutura representa um utilizador que criou uma conta na plataforma do GlarAssist. Na mesma estão todos os dados relacionados com o utilizador, como por exemplo: nome, *id* da organização, email, entre outros.

- Session – Uma s*ession* representa uma sessão de utilizador. Na mesma são armazenados dados referentes à própria sessão, como por exemplo, *id* de utilizador, dispositivo onde foi criada, validade da sessão, entre outros. A sessão pode ser de um utilizador autenticado ou de um *Guest*. A principal diferença entre as duas sessões é que a sessão de um *Guest* está associada sempre a uma sala, por consequência, aquela sessão não pode ser utilizada noutra sala. Contudo, as sessões autenticadas podem entrar e sair de diferentes salas diversas vezes.
- Room – A sala onde são realizadas as assistências remotas. Na estrutura da sala são armazenados o código para entrar na sala, a validade, entre outros dados. Todos os recursos gerados (mensagens, ficheiros, gravações, entre outros) dentro de uma sala são referenciados ao *id* da sala correspondente.
- RoomLog – Como o nome indica são os registos das salas. Estes registos são utilizados para identificar os utilizadores que entraram nas salas.
- Entity – Entity ou entidade representa as organizações, grupos ou empresas criadas na aplicação. Nas aplicações de *frontend* são apresentadas como organização, mas no servidor são identificadas como entidades (Entity).

Durante o desenvolvimento da aplicação são adicionadas mais lógicas de negócio, consoante as necessidades das funcionalidades a serem desenvolvidas.

Para apresentar as alterações realizadas nas API foi definida uma estrutura, em formato tabela, onde estão inseridos todos os eventos e pedidos adicionados nas API do GlarAssist, durante o desenvolvimento das funcionalidades. As alterações nas API são apresentadas da Tabela 1-9. A estrutura definida para representar cada alteração foi a seguinte: Nome, identificador do pedido ou evento; API, qual a API alterada (REST ou WBS); Autenticado, se o pedido ou evento só pode ser executado por utilizadores autenticados (necessário *token*); Rota/Evento; *Payload*, os dados a enviar no pedido; e o Resultado esperado, a resposta ou consequência do pedido ou evento. O método dos pedidos HTTP da API REST e as estruturas de *payload* e dos resultados esperados não são referidos, para não comprometer informação sensível da empresa. Também as rotas e eventos foram alterados, pelo mesmo motivo.

4.1. *Chat*

A possibilidade de comunicar de diferentes formas, numa aplicação de assistência remota com RA, é importante, uma vez que, podem ser utilizadas em diversos contextos. A comunicação padrão da assistência, por áudio, poderá não ser possível utilizar. Desta forma, uma funcionalidade como o *chat* possibilita aos utilizadores terem sempre uma forma de comunicar, mesmo quando a comunicação por voz não está disponível. Para além de ser uma nova forma de comunicar, possibilita aos utilizadores enviarem outro tipo de informação que antes não era possível, como por exemplo *links*. Numa fase inicial as mensagens não eram identificadas com o nome ou email do utilizador, mas posteriormente com o desenvolvimento de novas funcionalidades, foi adicionado à mensagem o identificador do utilizador de quem a enviou.

Foi definido pela equipa de desenvolvimento que a comunicação desta funcionalidade seria pela API Websocket. A mesma poderia ser realizada pelo serviço do Agora RTM, uma vez que, já estava a ser utilizado para a troca de mensagens em relação à assistência (coordenadas do desenho, estado do assistido, entre outras mensagens). Contudo, não foi utilizado o Agora RTM porque o mesmo tem um limite de mensagens por minuto e o serviço já estava sobrecarregado com os eventos anteriores. Ainda, pela API Websocket o servidor poderá obter um registo das mensagens enviadas, posteriormente, ao contrário do Agora RTM onde não seria possível rastrear estes eventos.

Uma vez que, cada utilizador tem de se juntar a uma sala na API WBS, para desenvolver esta funcionalidade, no lado do servidor, foi apenas necessário estender os eventos Websocket da sala, visto que, a sala é identificada pela sessão do utilizador. Foi então criado um evento *RoomChat*, apresentado na Tabela 1. Esta mensagem recebe como *payload* os dados (por exemplo o texto da mensagem) que o utilizador pretende enviar. Quando o servidor recebe este evento, é enviada uma mensagem *broadcast* para a sala. No evento não são necessárias outras validações, uma vez que, a estrutura da *payload* já tem um limite de espaço (o tamanho da mensagem já era validado) e a sala é recolhida da sessão do utilizador, assim, a sala era sempre válida.

Nome/Prop.	API	Autenticado	Rota/Evento	*Payload*	Resultado esperado
RoomChat	Websocket		room_msg	Dados da mensagem	Mensagem *broadcast* enviado para a sala com a mensagem

Tabela 1 - Alteração da API na funcionalidade do chat

Esta funcionalidade é destinada a ambas as aplicações de *frontend*. Na aplicação React foram desenvolvidas as *interfaces* definidas pela equipa do produto. Essas *interfaces* são apresentadas nas Figuras 3 e 4. Na implementação das *interfaces* foi desenvolvido o componente lateral (componente *parent* do *chat*), visível na Figura 3, que é *collapse* (elemento com dois estados, visível e escondido, que altera consoante o objetivo, geralmente, com uma animação). O desenvolvimento deste elemento foi relevante, uma vez que, foi utilizado, posteriormente, para outras funcionalidades na chamada. Quando o utilizador recebe uma mensagem é guardada a hora em que a mesma foi recebida, para apresentar no componente do *chat*. Para o componente do *chat* ser exibido foi adicionado um botão à lista de botões dentro de uma assistência (ver Figura 3).

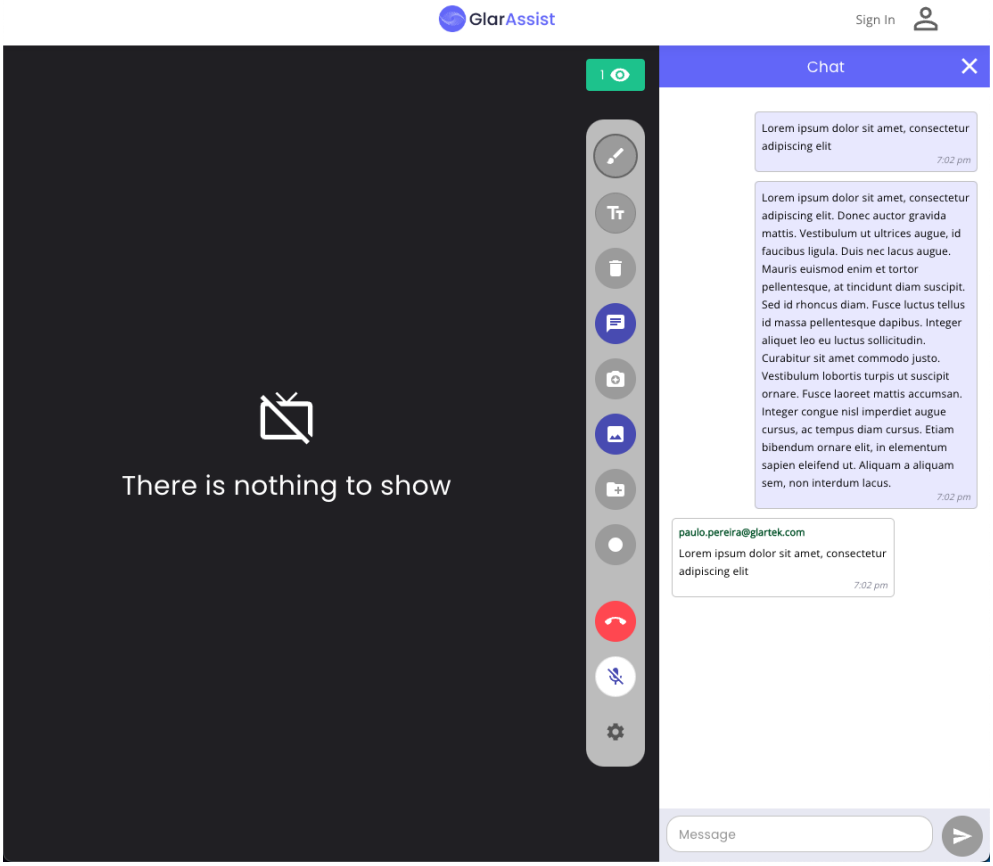


Figura 3 - Lista de botões da sala e o componente do chat

Quando um utilizador recebe uma mensagem é gerada uma notificação. As notificações (alerta e *icon*) eram apresentadas apenas quando o *chat* estava escondido (ver Figura 4). Durante o desenvolvimento foi detetado um ponto que não foi definido pela equipa de produto, que era o *scroll* do componente do *chat*, mais concretamente, o seu comportamento quando o *chat* alterava de estado (escondido/visível). Foi então definido em reunião, com os elementos de ambas as equipas envolvidas nesta funcionalidade, que o *scroll* voltava à posição inicial quando o utilizador recebia uma mensagem e o componente estava escondido. Contudo, nos restantes casos, manteria a posição em que o utilizador estava, para não perturbar a experiência do utilizador.

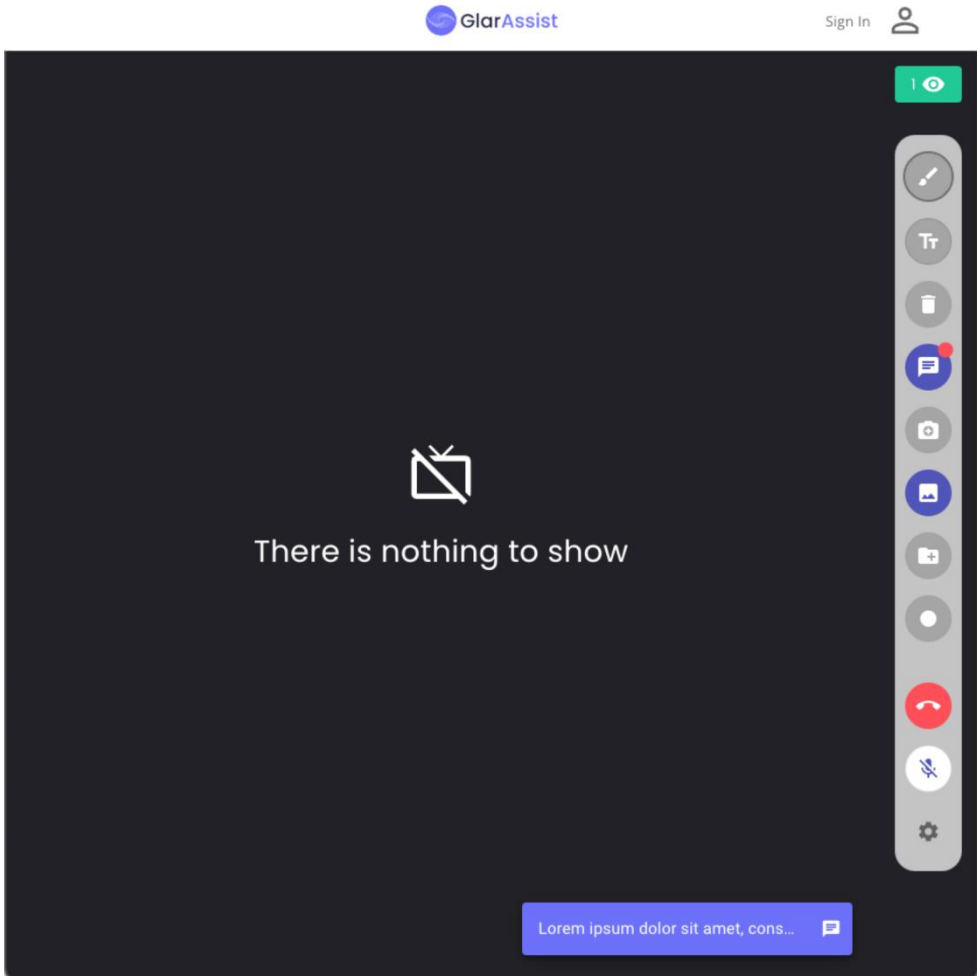


Figura 4 - Alertas de novas mensagens

Em relação à comunicação na aplicação *web*, a mesma teve de ser integrada pela API Websocket. O desenvolvimento desta foi previamente preparado, uma vez que, se fosse toda desenvolvida nesta funcionalidade iria adiar a entrega da tarefa no *sprint*. Apesar da funcionalidade estar desenvolvida, ainda teve de ser adaptada para ser utilizada durante as

chamadas. Com a junção da API Websocket na aplicação, foi alterada a jornada de entrada nas assistências. Esta jornada, na aplicação React, pode ser representada por fases, sendo elas:

1. Recolher a informação da *Room* pela API REST.
2. Conectar-se aos *channels* do Agora RTC e RTM.
3. Conectar-se ao Websocket.
4. Quando confirmada a autenticação do Websocket é enviado o evento para se juntar à sala.
5. Quando é recebida a confirmação que foi aceite, existe a troca de mensagens na assistência.
6. Posteriormente, quando o utilizador sai da sala é enviado um evento específico para sair da sala.
7. E por fim são desconectados os serviços do Agora e do Websocket.

Durante a assistência, quando o utilizador envia uma mensagem, é enviado o evento *room_chat* pela API Websocket para notificar os outros utilizadores. Da mesma forma, quando o utilizador recebe o evento de uma nova mensagem, o mesmo é tratado e a UI é atualizada para apresentar a mensagem.

4.2. Histórico das sessões de utilizadores

No capítulo anterior, onde as aplicações de assistência remota de RA são abordadas, foi demonstrada a importância da documentação ou histórico dos recursos partilhados nas assistências para os utilizadores e as organizações. Com o armazenamento da informação é possível obter todas as assistências já realizadas e o que foi partilhado nas mesmas. Esta funcionalidade é essencial para o BackOffice do GlarAssist. Contudo, esta *feature* é extensa, uma vez que, são apresentadas diversas informações da assistência e é necessária a criação de novos contextos na aplicação. Por consequência, a mesma foi desenvolvida parcialmente.

Na fase inicial foi apenas implementado o histórico de utilizadores. Este histórico está disponível a qualquer utilizador autenticado, que utilize a aplicação. Na tabela do histórico são apresentadas todas as assistências em que o utilizador participou. Os campos que podem ser visualizados na tabela são: o código da sessão, a data em que a mesma foi criada, quem

é que a criou, a última vez que o utilizador entrou na sessão e a data de expiração na sessão. A data de expiração é apresentada nas aplicações *frontend* como um botão para se juntar diretamente à sessão, caso a sessão não tenha expirado. Para possibilitar ao utilizador uma pesquisa mais complexa, foi desenvolvido o filtro de ordenação. Com este filtro, os utilizadores conseguem ordenar de forma ascendente e descente, texto ou valores numéricos. Por consequência, os valores da tabela que podem ser ordenados são: o código da sessão, a data em que foi criada a sessão, o email do utilizador que criou a sessão e a data da última vez que o utilizador entrou na sessão. É importante referir que esta fase foi desenvolvida e integrada no ambiente de produção, antes da funcionalidade do *Chat*. Assim, a API Websocket não estava ainda integrada nas aplicações. Como o servidor não tem referência das assistências que os utilizadores participaram, foi desenvolvida uma solução para resolver esta questão.

Para o desenvolvimento desta fase no servidor foram adicionadas novas lógicas de negócio e novos pedidos na API REST. Para solucionar a falta de registos dos utilizadores nas assistências, foi criado um pedido novo na API REST, para as aplicações do *frontend* notificarem o servidor que um utilizador entrou numa sala/assistência. Para realizar este pedido é necessário o *token* de autorização, uma vez que, o mesmo é apenas para utilizadores autenticados. O pedido está representado na Tabela 2, como *RoomJoin*. Ao realizar este pedido, é adicionado na base de dados, um novo registo numa nova estrutura, identificada por *RoomLog*. Estes registos são depois utilizados para obter o histórico do utilizador. Para obter o histórico dos utilizadores foi criado o *GetUserHistory*, representado também na Tabela 2. Nesse pedido as aplicações *frontend* poderão enviar na *payload* informação para alterar o histórico consoante o contexto, como por exemplo: qual o número de registos por página, o campo que irá ser ordenado, entre outros. Caso a *payload* não seja enviada são definidos valores padrão na pesquisa à BD. O resultado esperado deste pedido são todas as assistências que o utilizador participou, consoante a *payload* enviada.

Nome/Prop.	API	Autenticado	Rota/Evento	*Payload*	Resultado esperado
RoomJoin	REST	X	/rooms/join/id	-	Status 200
GetUserHistory	REST	X	/rooms/history/user	Filtros definidos	Todas as assistências do utilizador
HistoryDetails	REST	X	/rooms/history/room/id	-	Toda a informação

					partilhada na assistência
SetRoomTitle	REST	X	/rooms/id/title	Titulo definido pelo utilizador	Status 200
SetRoomLabel	REST	X	/rooms/id/label	*Label* definida pelo utilizador	Status 200
GetUserLabels	REST	X	/labels/user	-	Todas as *labels* do utilizador
AddUserLabel	REST	X	/labels/add	Dados da nova *label*	Nova *label* criada
UpdateUserLabel	REST	X	/labels/update	Novos dados da *label*	*Label* atualizada
DeleteUserLabel	REST	X	/labels/delete	ID da *label* a remover	Status 200

Tabela 2 - Alterações da API na funcionalidade do histórico de sessões

Na aplicação *web* foi adicionada uma nova rota, */history/user/<page>*, como definido nos *mockups* da equipa do produto, para os utilizadores acederem ao seu histórico. As *interfaces* estão representadas na Figura 5. Para além da nova página, foi implementada uma *sidebar* apresentada também na Figura 5. Esta *sidebar* só é visível a utilizadores autenticados e se não estiverem numa assistência. A página do histórico foi desenvolvida para que a rota definisse a página da tabela do histórico. Isto é, se a rota do *browser* for */history/user/2* a tabela vai apresentar a segunda página do histórico. Por consequência, se o utilizador alterar a página pelo componente de paginação da tabela, o utilizador é enviado para a nova rota. Para desenvolver esta funcionalidade, os dados referentes ao filtro são enviados pelo *hook* do React *history*. Assim, quando a página é carregada, os filtros são mantidos e aplicados automaticamente. Para além desta implementação, foi necessário reestruturar a jornada para entrar numa sala, uma vez que, se a sala for válida, o utilizador pode entrar diretamente da página do histórico. Essas alterações foram realizadas para que a jornada fosse a mesma, nas diferentes possibilidades de entrar (por exemplo, entrar pela *homepage*, introduzir diretamente o *link*, entre outras formas), removendo assim a necessidade de criar diferentes jornadas.

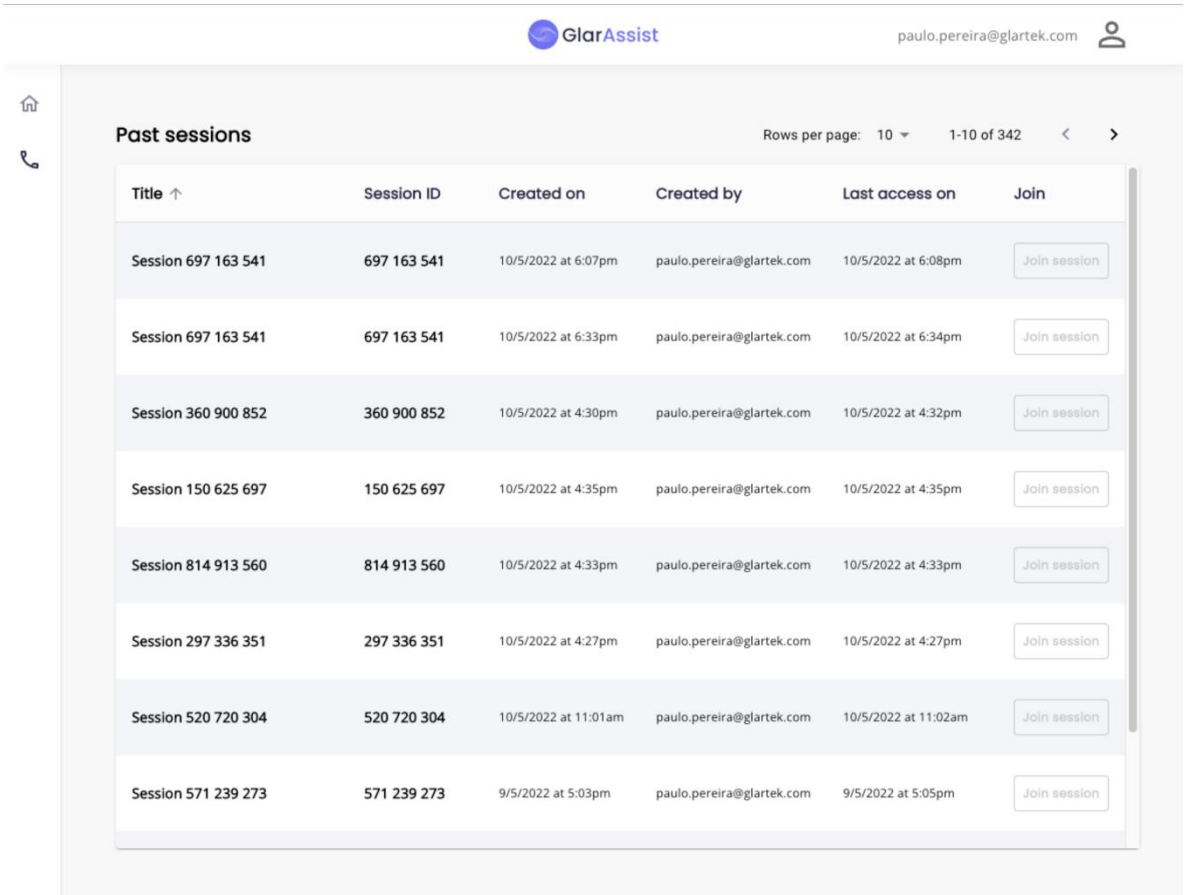


Figura 5 - Componente do histórico de utilizador, com a sidebar lateral

Depois da implementação da fase inicial foi desenvolvida a possibilidade de o utilizador ver ainda mais informação sobre a sessão. Desta forma, surgiu o conceito "detalhes da assistência". O objetivo é apresentar mais informação partilhada nas assistências. Se o utilizador pretender ver os detalhes da assistência, na tabela do histórico, ao clicar na linha da tabela referente à sessão, é enviado para a página dos detalhes da mesma. Nessa página poderá observar, por um lado, todos os detalhes apresentados na tabela de histórico referentes à sala, assim como, os participantes da assistência e as mensagens de texto trocadas no *chat* da sala. Os participantes não autenticados, *guest*s, não geram registos, uma vez que, de momento, os *guests* não têm nenhuma referência, e por consequência, não há a possibilidade de os identificar. Uma funcionalidade definida, a ser desenvolvida futuramente, é os *guests* terem de introduzir um nome, para se identificarem, antes de entrarem numa sala.

No servidor foi definido um novo pedido na API REST para carregar os detalhes de uma sala. Na Tabela 2 esse pedido é apresentado, como *HistoryDetails*. Neste pedido é devolvida ao utilizador toda a informação referente à assistência. O mesmo, para além de ser utilizado

apenas por utilizadores autenticados, só pode ser requerido por utilizadores que participaram na assistência pretendida. Este pedido, durante o desenvolvimento das restantes funcionalidades, será alterado, caso seja necessário adicionar mais informação da assistência. Para além deste pedido, foi criada uma estrutura, para armazenar as mensagens enviadas na base de dados. Quando um utilizador envia uma mensagem pelo evento *RoomChat* (ver Tabela 1), é guardado um registo do mesmo. Em cada registo é armazenado: o *id* da sala, o *id* do utilizador que enviou a mensagem (caso seja com uma sessão de autenticação), o texto da mensagem, entre outras informações. Para além do registo na BD, foi alterada a resposta do pedido para as aplicações *frontend* conseguirem identificar o utilizador que enviou a mensagem. Quando o pedido do *HistoryDetails* é realizado são retornadas todas as mensagens armazenadas nesta estrutura, que pertençam à assistência solicitada.

Para o desenvolvimento da funcionalidade na aplicação React foi necessário adicionar uma nova rota, */history/session/<room_id>*, com o componente correspondente. As UI desenvolvidas são apresentadas nas Figuras 6 e 7. Na *interface* são apresentados todos os detalhes referidos acima. O componente do *chat* é reutilizado com o componente desenvolvido na funcionalidade anterior. Por consequência, o código é reutilizado, não havendo necessidade de voltar a desenvolver componentes anteriormente criados, uma vez que, os componentes são iguais (com algumas exceções). No mesmo, foi, ainda, adicionada a funcionalidade de identificar quem enviou a mensagem. Para os utilizadores *guest* é associado um número por ordem de chegada, criando assim um nome dinâmico (*Guest 1*, *Guest 2*, …). Na Figura 6 é possível observar a nova *interface* da mensagem.

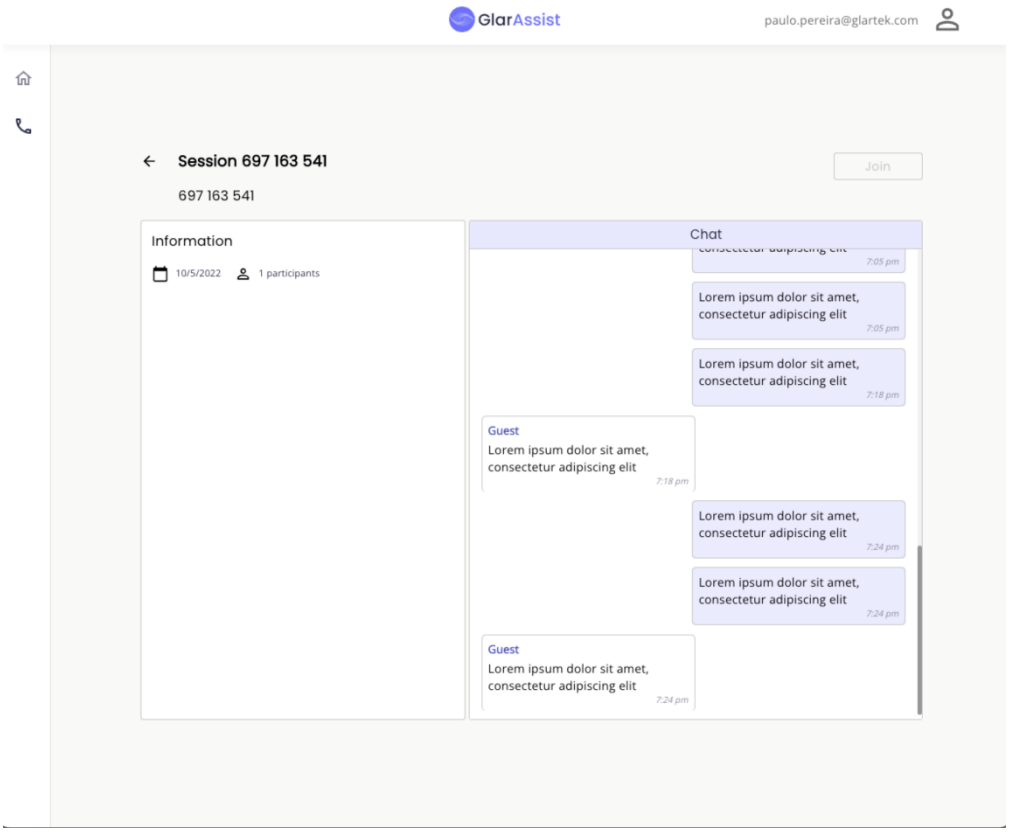


Figura 6 - Componente dos detalhes de uma assistência

Nas mensagens enviadas por utilizadores autenticados, o email é o identificador da mensagem. Para além do componente do *chat*, é importante referir o componente desenvolvido para apresentar os participantes. Este componente é um Popover, componente do Material UI, com a lista dos participantes da assistência, apresentado na Figura 7.

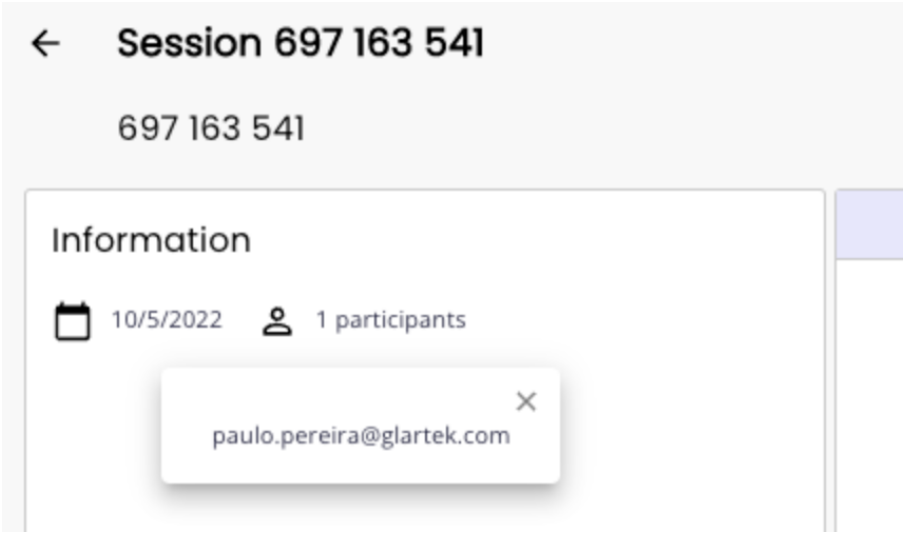


Figura 7 - Lista de utilizadores da assistência

Para complementar as fases anteriores e possibilitar diferentes formas de identificar as sessões, foram desenvolvidas duas funcionalidades: criar e associar uma *label* às sessões e definir um título/nome para cada sessão. A *label* e o título da sessão são referentes ao utilizador. Deste modo, os dados definidos são apenas apresentados ao utilizador que os definiu. Esta funcionalidade torna-se ainda mais importante quando foram integrados os filtros avançados na tabela do histórico. Com estas funcionalidades, os utilizadores conseguem definir identificadores às sessões, facilitando a distinção das mesmas. Para isso foi adicionada a possibilidade de gerir o título e a *label* da sessão na página de histórico das sessões e na página dos detalhes da sessão.

No servidor foi alterada a estrutura da *Room* para armazenar estes dados por utilizador. Se um utilizador associa algum título ou *label,* é adicionada a essa estrutura uma entrada, identificada pelo *id* do utilizador, com o título definido e o *id* da *label* associada. Por consequência, também foi adicionada uma nova estrutura à base de dados para armazenar as *labels* definidas pelos utilizadores. A estrutura foi a seguinte: *id* do utilizador, nome da *label*, cor da *label* (em formato hexadecimal), entre outros campos. Com estas alterações é possível armazenar toda a informação necessária para desenvolver esta funcionalidade. Também foram definidos pedidos para gerir estes dados. Os pedidos criados são apresentados na Tabela 2 e são os seguintes: *SetRoomTitle*, *SetRoomLabel*, *GetUserLabels*, *AddUserLabel*, *UpdateUserLabel* e *DeleteUserLabel*. O *SetRoomTitle* possibilita ao utilizador alterar o título definido para a assistência. O *SetRoomLabel* possibilita alterar a *label* definida para a assistência. O *GetUserLabels* retorna todas *labels* criadas pelo utilizador. O *AddUserLabel* adiciona uma nova *label* de utilizador, o *UpdateUserLabel* atualiza uma *label* do utilizador. O *DeleteUserLabel* remove uma *label* do utilizador.

Estes pedidos possibilitam às aplicações *frontend* gerirem os dados da funcionalidade da presente fase. Para além da criação destes pedidos, os pedidos criados anteriormente, *GetUserHistory* e *HistoryDetails,* foram alterados para retornar os dados, título e *label*, definidos pelo utilizador nas sessões retornadas.

Na aplicação React foram alteradas as páginas criadas anteriormente - a página do histórico das assistências e a página com os detalhes da assistência. As alterações realizadas são apresentadas nas Figuras 8, 9 e 10. O objetivo destas alterações é possibilitar aos utilizadores alterar o título e a *label* das salas, e ao mesmo tempo, gerir as suas *labels* em ambas as páginas. Neste sentido, foram desenvolvidos dois Popovers, componente do MUI,

um para alterar o título de uma assistência (ver Figura 8) e outro para gerir as *labels* do utilizador e ao mesmo tempo alterar a *label* de uma assistência, visível na Figura 9 (jornada para associar uma *label* a uma assistência, como a jornada para visualizar, adicionar, editar ou remover *label(s)*).

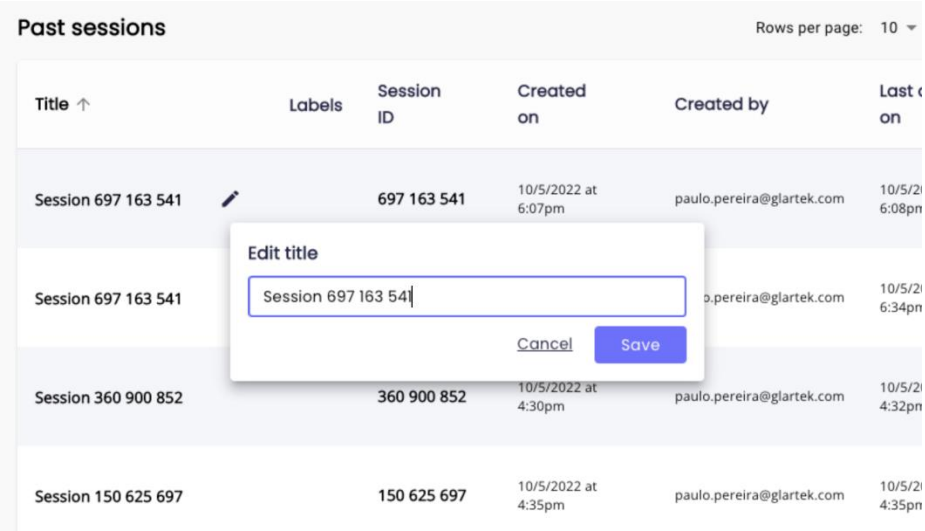


Figura 8 - Popover para editar o título de uma assistência

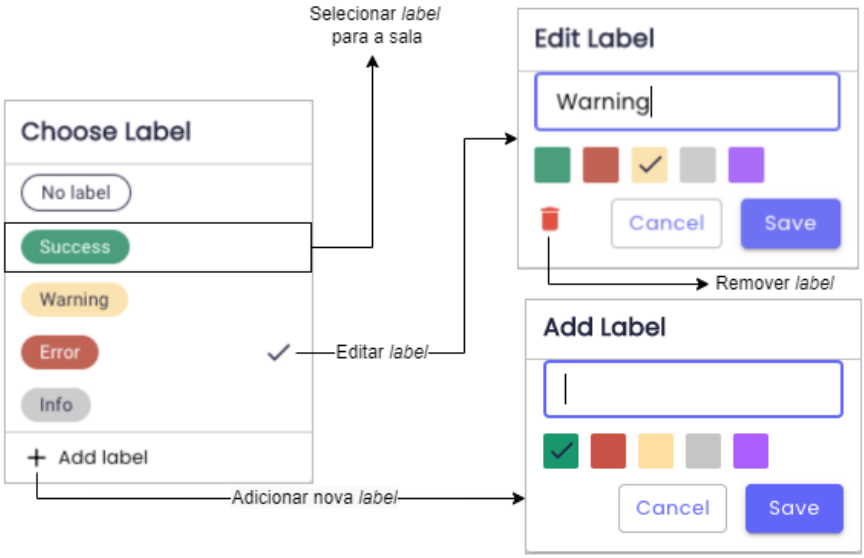


Figura 9 - Jornada do Popover para a gestão e associação de Labels

Para além dos Popovers foi, também, alterada a UI, mais propriamente as colunas da tabela do histórico de assistências e o *header* da página dos detalhes da assistência (ver Figura 10). Com a introdução do título das assistências à tabela do histórico, foi adicionada à coluna correspondente a possibilidade de ordenar pelo título. Ao executar as diferentes ações, a aplicação realiza os pedidos criados no servidor, para aceder e gerir os novos dados.

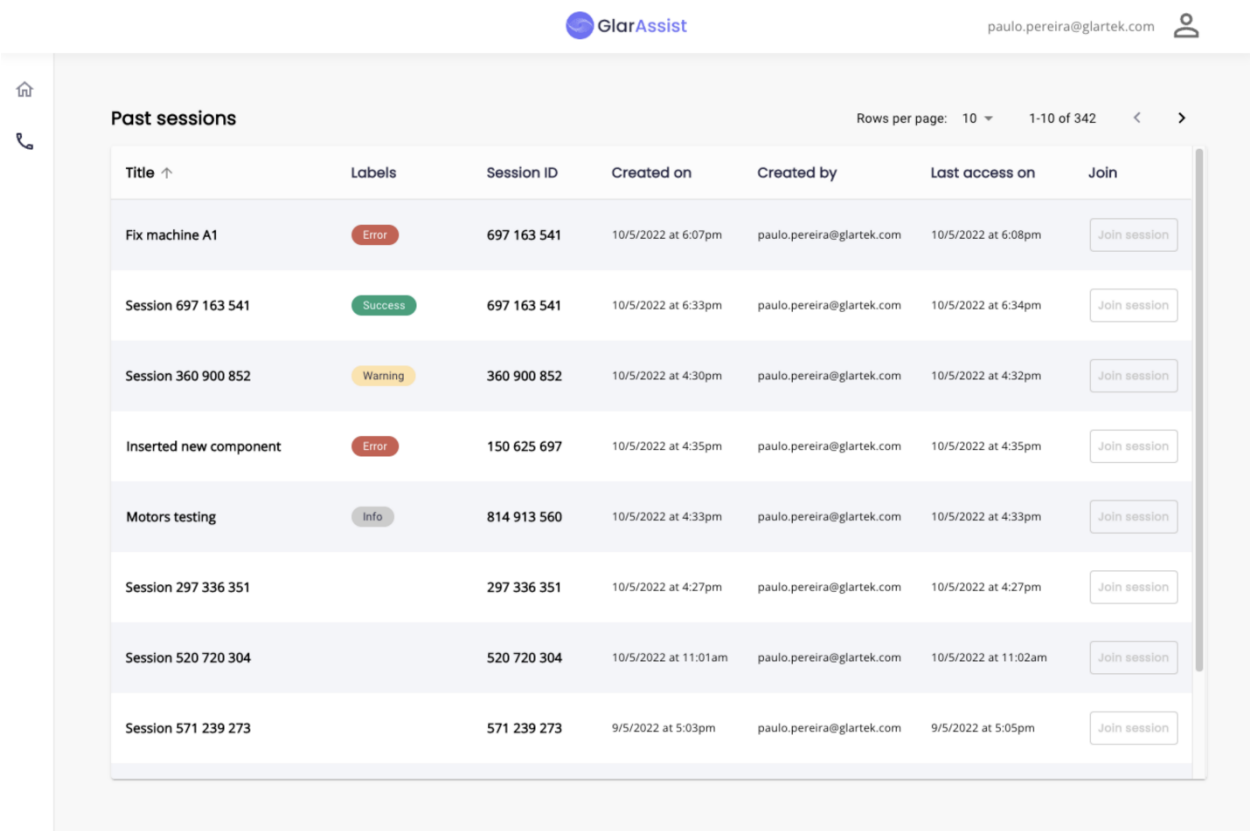


Figura 10 - Histórico de assistências do utilizador com os títulos e labels

Esta funcionalidade é muito importante na aplicação do GlarAssist, pois foi o primeiro passo para armazenar e possibilitar visualizar a informação partilhada na aplicação. Esta funcionalidade sofrerá alterações, uma vez que, será adicionada ainda mais informação ao histórico.

4.3. Envio e visualização de ficheiros

Atualmente, a aplicação apenas possibilita transferir ficheiros de utilizadores assistentes para utilizadores assistidos. Além disso, apenas os utilizadores que tiveram envolvidos na partilha é que sabem que a mesma existiu. Desta forma, não é possível partilhar ficheiros para todos os utilizadores presentes na sala da assistência. Uma maneira para solucionar este problema, e ao mesmo tempo explorar uma funcionalidade já implementada, é possibilitar a partilha de ficheiros pelo *chat* das assistências. Dessa maneira, os utilizadores podem partilhar documentos, que podem ser importantes na resolução de problemas, e, ao mesmo tempo, esses documentos ficam armazenados e associados ao histórico da assistência.

Esta funcionalidade foi divida em duas fases, uma vez que, para além da partilha de ficheiros pelo *chat*, foi requerido adicionar os ficheiros partilhados na página dos detalhes de uma assistência. Assim, primeiro, foi desenvolvida a funcionalidade da partilha de ficheiros pelo *chat* da sala e, só depois, foram adicionados os documentos partilhados ao histórico das assistências.

Na funcionalidade foi integrado um novo serviço no GlarAssist, para possibilitar o armazenamento de ficheiros. Esse serviço foi o AWS S3. A decisão da utilização do serviço foi efetuada pelo *Scrum Manager* e alguns elementos da equipa de desenvolvimento, uma vez que, o mesmo já era utilizado noutra aplicação e era pretendido que os serviços utilizados nas aplicações da empresa fossem semelhantes. Em reunião de equipa, foram consideradas diversas abordagens de como é que os ficheiros iam ser partilhados. Foi decidida a utilização dos *presigned urls* para o *download* e *upload* de ficheiros, uma vez que as principais vantagens desta abordagem são: possibilidade de adicionar diferentes credenciais de segurança aos *urls* (data de expiração, tamanho máximo e mínimo dos ficheiros, entre outras), facilitação da partilha de ficheiros (basta ter acesso ao *url*), garantia de privacidade (o servidor é quem gere os *urls*) e, por fim, remoção da carga do servidor (pois, é apenas um intermediário e, assim, os ficheiros são sempre enviados diretamente para o *bucket*).

No servidor foi adicionada a biblioteca para realizar operações no *bucket* do serviço da AWS S3. Estas operações são, por exemplo: listar todos os objetos de um certo caminho, *upload* ou *download* de ficheiros, geração de *presigned urls*, entre outros. Para além disso, foi estendida a *config* do servidor para armazenar os dados referentes à ligação ao serviço e ao *bucket* da AWS S3. De forma a organizar os ficheiros no *bucket* foi definida uma estrutura, cujo objetivo é armazenar toda a informação de uma assistência no mesmo caminho. Assim quando for necessário guardar ou recolher alguma informação do *bucket*, os ficheiros estão todos organizados pelas salas de assistência. Os objetos a guardar seriam os ficheiros partilhados e as gravações da assistência realizadas pelo serviço do Agora Cloud Recording. As gravações das assistências já tinham sido desenvolvidas e integradas na aplicação como prova de conceito, uma vez que, os utilizadores, para acederem às gravações, teriam de as requerer à Glartek. Com esta funcionalidade é possível aceder às gravações efetuadas diretamente no histórico da assistência.

Para obter um objeto do *bucket* é criado um *presigned GET url*. Nestes *urls* são definidos diferentes tempos de expiração consoante o tipo de ficheiro a ser retornado (públicos ou

privados). Para o envio de ficheiros foram consideradas duas formas: *presigned PUT url* e *presigned POST url*. Ambas as abordagens possibilitam o *upload* de ficheiros para o *bucket*, com políticas de segurança semelhantes. O método POST foi o escolhido, uma vez que, o PUT não possibilita limitar o tamanho do ficheiro a ser enviado. Desta forma, podem ser enviados ficheiros com tamanhos que excedam o limite autorizado. Apesar das diversas políticas, em ambos os métodos, não é possível controlar o tipo de ficheiro enviado. Como não é possível restringir pelo *presigned url*, foi definida uma lista de ficheiros válidos no *bucket*. Assim, caso um utilizador malicioso utilize um *url* para enviar um ficheiro não autorizado, o *bucket* rejeita o pedido. Outra possibilidade dos *presigned urls* é a associação de metadados aos objetos do *bucket*. Com esses metadados foram armazenados dados referentes ao ficheiro, como por exemplo: utilizador que enviou o ficheiro, nome original do ficheiro, entre outros.

Na abordagem definida, optou-se por não armazenar as referências dos ficheiros na base de dados. Esta decisão foi tomada, uma vez que, os ficheiros já estavam organizados por assistências e os metadados associados aos objetos do *bucket* são as únicas informações necessárias. Para além disso, é removida a necessidade de armazenamento e de gestão dos registos na BD quando um ficheiro é adicionado ou removido. Quando é necessário obter os ficheiros ou gravações de uma sala, basta realizar uma pesquisa no caminho da assistência, que é devolvida toda a informação referente aos ficheiros.

Depois de implementar toda a lógica do novo serviço foram criados e alterados pedidos da API REST, como também outras funcionalidades. Como os ficheiros eram enviados pelo *chat*, qualquer utilizador poderia enviar ficheiros. Isso cria vulnerabilidades, pois, um utilizador malicioso poderá enviar uma grande quantidade de ficheiros para tentar quebrar ou piorar o desempenho do servidor ou do *bucket*. Assim, foram definidas normas para impedir este tipo de situações. Em relação ao *bucket*, foi definida uma lista de ficheiros permitidos, como referido anteriormente. Como o servidor consegue controlar o tamanho dos ficheiros pelos *presigned POST urls*, foi estabelecido um tamanho máximo por ficheiro. Da mesma forma, foi definido um limite de espaço que cada sala de assistência poderia ter. Assim como, foi definido um limite de ficheiros por mensagem. Em relação aos utilizadores *guest*, foi estabelecido um limite de ficheiros que os mesmos podem enviar. O acesso aos ficheiros partilhados é restrito, pois, só os utilizadores que participaram na assistência é que estão habilitados a realizar estas ações.

Para a partilha de ficheiros pelo *chat* foi alterada a *payload* do evento *RoomChat*, como se pode verificar na Tabela 1, para guardar a lista de *ids* referentes aos ficheiros anexados à mensagem. Os pedidos da API REST adicionados estão representados na Tabela 3, sendo eles: *GetRoomFile*, *AddRoomFile* e *DeleteRoomFile*. Todos os pedidos não são exclusivos a utilizadores autenticados, porém, existe a necessidade de enviar a sessão de forma a gerir as restrições referidas anteriormente. Com estes pedidos, os utilizadores conseguem partilhar os ficheiros durante a assistência.

Nome/Prop.	API	Autenticado	Rota/Evento	*Payload*	Resultado esperado
GetRoomFile	REST		/files/room/room_id	-	Dados do ficheiro, com o *presigned GET url*
AddRoomFile	REST		/files/add	Dados referentes aos ficheiros	Dados do *presigned POST url*
DeleteRoomFile	REST		/files/delete	-	Status 200

Tabela 3 - Alterações da API na funcionalidade envio e visualização de ficheiros

No desenvolvimento da aplicação *web*, o componente *chat* desenvolvido anteriormente foi adaptado para permitir a partilha de ficheiros. Para além deste componente, foi desenvolvido um novo, com o objetivo de possibilitar aos utilizadores verem os ficheiros partilhados com mais detalhe, criando assim o componente de pré-visualização dos ficheiros. Este componente é um Dialog, do MaterialUI, que permite aos utilizadores observarem as imagens com mais detalhe, visualizar vídeos, ver outros detalhes do ficheiro ou fazer o *download* dos ficheiros. Nas Figuras 11, 12 e 13 são apresentadas estas implementações.

Antes de enviar uma mensagem com ficheiros anexados, os ficheiros terão de estar já disponíveis no *bucket*, por forma a que os outros utilizadores consigam visualizar ou obter os ficheiros. Consequentemente, o *upload* de todos os ficheiros anexados é realizado antes da mensagem ser enviada. Desta forma, quando um utilizador anexa um ou vários ficheiros numa mensagem, a UI tem de apresentar ao utilizador o estado do *upload* dos ficheiros (ver Figura 11). Enquanto o *upload* de ficheiros é realizado, o botão para enviar a mensagem fica desabilitado. Quando o *upload* de todos os ficheiros é concluído, o botão para enviar a mensagem fica, novamente, habilitado. Desta forma, quando os ficheiros são enviados, é garantido que os mesmos estão disponíveis para todos os utilizadores.

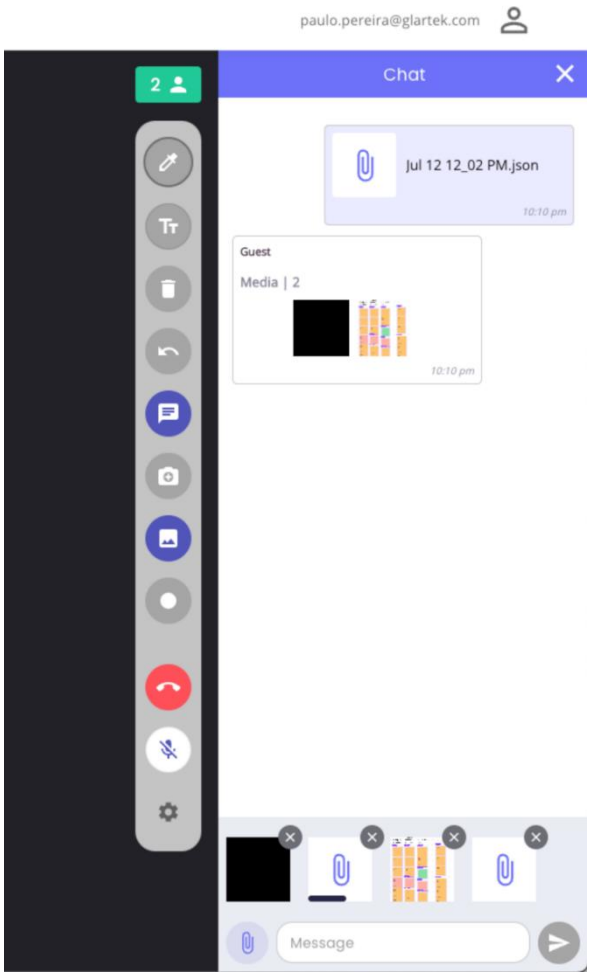


Figura 11 - Chat com ficheiros anexados às mensagens

Porém, com esta abordagem, os ficheiros têm de ser geridos quando um utilizador os remove da mensagem, depois do *upload* do ficheiro ser concluído. Caso estas situações não sejam tratadas, podem causar problemas, pois, os ficheiros vão para o *bucket* enquanto estão no estado "por enviar". Nestes casos, os ficheiros têm de ser removidos do *bucket*, uma vez que, os mesmos nunca chegaram a ser enviados para os outros utilizadores. Todas estas situações foram devidamente testadas com ficheiros anexados (fechar o *browser* ou página, sair da chamada, fechar o componente do *Chat*, entre outras). Para lidar com as mesmas, foi desenvolvida uma função que é executada automaticamente nestas situações para remover estes ficheiros.

Para além do envio de ficheiros, o componente de visualização das mensagens do *chat* foi adaptado para apresentar os ficheiros anexados às mensagens, como se pode verificar na Figura 11. O componente apresenta os ficheiros de diferentes maneiras, consoante o tipo de ficheiro. Caso não seja possível apresentar todos os ficheiros, os mesmos são agrupados. Ao clicar num desses ficheiros o componente de pré-visualização dos ficheiros é exibido e o

utilizador poderá ver o mesmo com mais detalhes (ver Figura 12). Caso a mensagem tenha múltiplos ficheiros, o utilizador pode transitar entre ficheiros diretamente do Dialog.

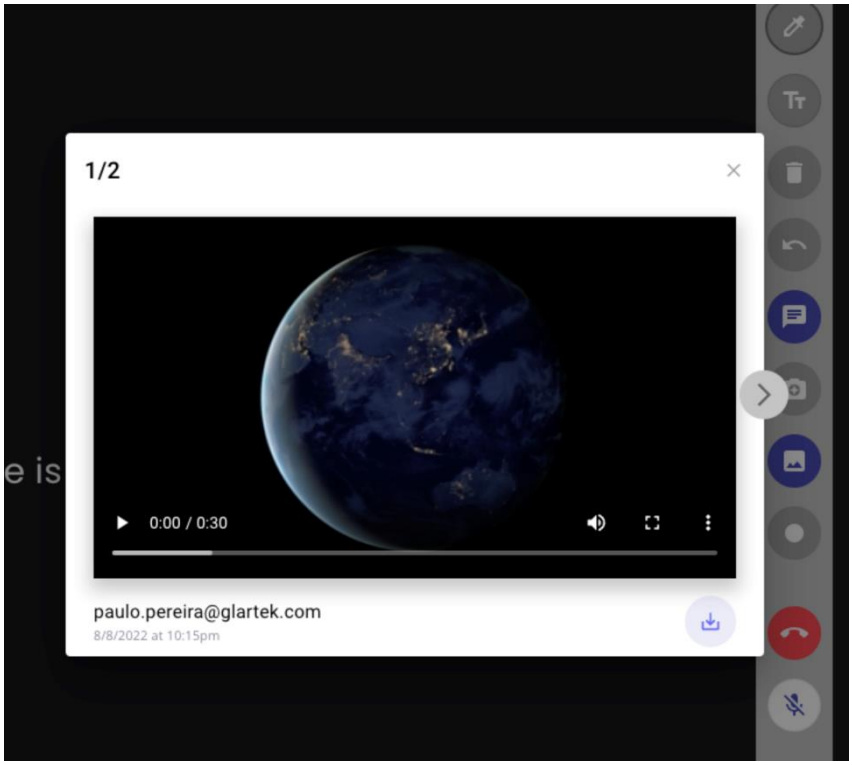


Figura 12 - Componente Dialog de pré-visualização de ficheiros

Posteriormente, foi desenvolvida a última fase desta funcionalidade: a adição dos ficheiros nos detalhes das assistências. Os ficheiros e as gravações estão todos armazenados num caminho específico referente à sala da assistência, o que facilita o retorno dos mesmos. No servidor quando uma aplicação *frontend* faz o pedido *HistoryDetails*, apresentado na Tabela 3, são adicionados os ficheiros da assistência à resposta do pedido, para que possam ser apresentados no histórico. Quando o servidor recebe esse pedido, verifica se existem ficheiros partilhados ou gravações na assistência. Caso existam, são criados *presigned GET urls* para cada ficheiro, com uma validade pré-definida, para os utilizadores acederem aos ficheiros. Na apresentação do *chat*, no histórico da assistência, também é importante identificar quais as mensagens onde foram adicionados os ficheiros. Uma vez que, todos os ficheiros estão identificados pelo *id*, as aplicações cliente conseguem associar os ficheiros às mensagens correspondentes.

Com a alteração realizada no pedido *HistoryDetails*, é possível apresentar esta informação no histórico da aplicação *web*. Na página dos detalhes das assistências os utilizadores poderão, assim, aceder a todos os ficheiros partilhados e gravações realizadas durante a assistência. Para isso, o componente da página foi alterado para apresentar esta

informação. As alterações realizadas podem ser visualizadas na Figura 13. Na mesma, é possível observar que foi criado um espaço dedicado para mostrar os ficheiros e as gravações. Para além deste espaço, as mensagens do *chat* estão com os ficheiros correspondentes. Para possibilitar aos utilizadores acederem aos ficheiros com mais detalhe, o componente Dialog, criado anteriormente para pré-visualização dos ficheiros, também é utilizado na página do histórico da assistência (ver Figura 12).

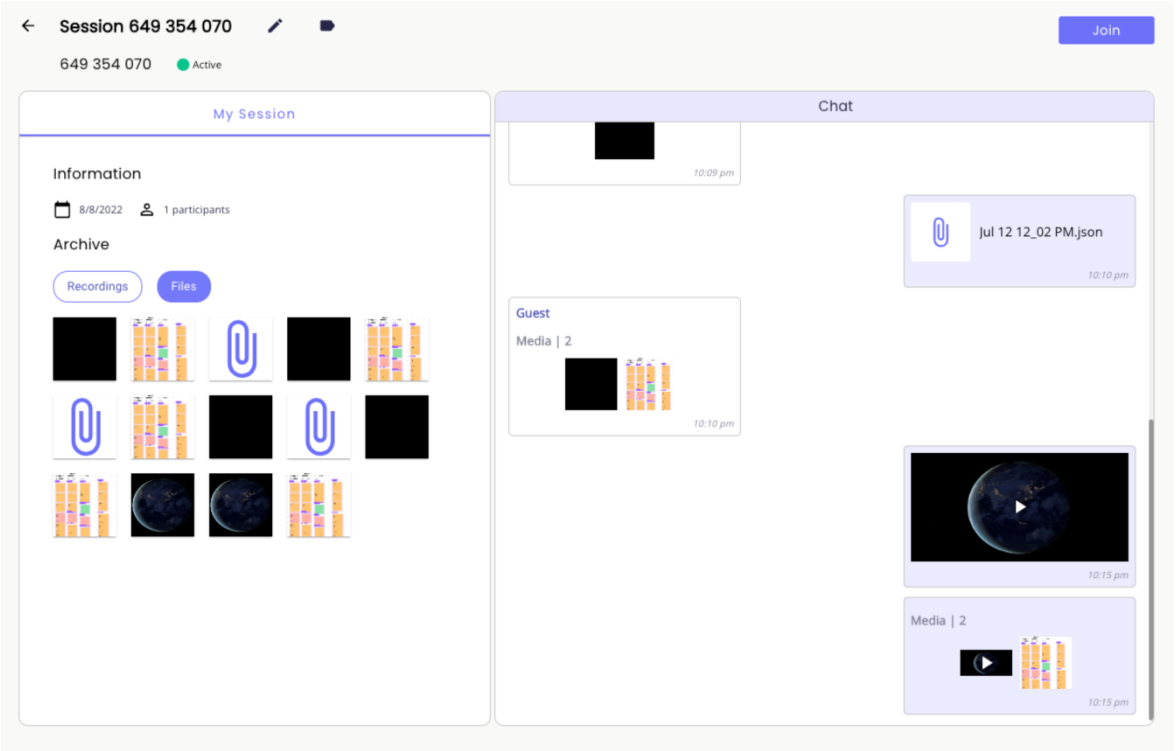


Figura 13 - Componente dos detalhes das assistências com os ficheiros partilhados

4.4. Organizações

A funcionalidade anterior permitiu aos utilizadores terem um registo de tudo o que foi realizado durante as assistências. Porém, as entidades que pretendem integrar o GlarAssist nas suas operações, não têm forma de gerir os seus recursos, pois, a abordagem atual do GlarAssist não permite agrupar ou juntar utilizadores ou assistências a um grupo ou equipa. Para solucionar este problema, foram criadas as organizações. O objetivo desta funcionalidade é permitir que um utilizador consiga criar uma organização, convide ou crie utilizadores para a mesma e que depois possa gerir os utilizadores e assistências realizadas pelos utilizadores da sua organização. Por consequência, tiveram de ser alteradas algumas

funcionalidades, uma vez que esta funcionalidade cria uma hierarquia de acesso e permissões na aplicação e nos seus recursos.

Para a implementação desta funcionalidade, foi realizada uma reunião onde estiveram presentes ambas as equipas (desenvolvimento e produto) para serem definidos os conceitos da nova funcionalidade. Ficou decidido o seguinte:

- Um utilizador apenas poderá pertencer a uma organização.
- Uma organização tem controlo de todos os seus utilizadores. Isto significa que, os administradores de uma organização poderão ativar/desativar ou remover contas de utilizadores, alterar a *role* dos utilizadores e acederem a todas as assistências que realizou.
- Nesta fase inicial, apenas existirão duas *roles*: Admin e Regular User. O Admin terá permissões para gerir a organização, enquanto, o Regular User apenas poderá visualizar informação acerca da organização (assistências de outros utilizadores não fazem parte dessa informação).
- As organizações têm um nome único, isto é, não é possível ter organizações com o mesmo nome.
- Uma organização pode alterar de nome, se o mesmo estiver disponível.
- Será possível convidar utilizadores para a organização via email.
- Só poderá ser enviado um convite por email.
- Caso o email esteja associado a uma conta já com uma entidade, o mesmo não é enviado.
- Todas as assistências criadas por utilizadores da organização são identificadas como assistências da organização.
- O nome "Organização ou *Organization*" é apenas utilizado nas aplicações *frontend*, enquanto no servidor o conceito é representado por "Entidade ou *Entity*". Esta decisão foi efetuada pelo Scrum Master e pela equipa de desenvolvimento, com o intuito de continuar com alguns conceitos anteriormente definidos.

Depois de definidos os conceitos da funcionalidade, foi iniciado o desenvolvimento do servidor. A implementação do servidor foi algo complexa, uma vez que, tiveram de ser adicionadas muitas das funcionalidades e alteradas funcionalidades já implementadas. Foram realizadas diversas alterações, como por exemplo: foi adicionado à estrutura *User* um novo campo para verificar se a conta está ativa; no *Login* tem de ser considerado o estado da

conta (ativada ou desativada); foi alterada a estrutura da *Room*, para armazenar o *id* da entidade, caso a mesma tenha sido criada por um utilizador de uma organização; entre outras modificações. A estrutura das entidades, *Entity*, já tinha sido criada anteriormente, mas não tinha sido ainda utilizada na aplicação. Para satisfazer todos os conceitos definidos foram apenas alteradas algumas funções e estruturas que já estavam implementadas. Para além das alterações nas estruturas da aplicação, também foi necessário realizar alterações na BD, devido às alterações realizadas previamente. Por consequência, foi necessário realizar migrações na BD. Para além das alterações foram criados diversos pedidos na API REST. Esses pedidos são apresentados na Tabela 4.

Nome/Prop.	API	Autenticado	Rota/Evento	*Payload*	Resultado esperado
GetEntity	REST	X	/entity	-	Informação da entidade do utilizador
GetEntityUsers	REST	X	/entity/users	Dados referentes ao filtro	Membros da pesquisa efetuada
GetEntityInvites	REST	X	/entity/invites	Dados referentes ao filtro	Convites pendentes da organização
CreateEntity	REST	X	/entity	Dados da entidade	Entidade criada
UpdateEntity	REST	X	/entity	Dados da entidade	Entidade atualizada
InviteUsersToEntity	REST	X	/entity/invite	Dados dos emails	Status dos convites
DeleteEntityInvites	REST	X	/entity/invite	*Id* do convite	Status 200
JoinByEntityInvite	REST		/entity/invite/id	Dados do convite	Utilizador adicionado à entidade
GetInviteData	REST		/invite	-	Dados do convite
UpdateUsersRoles	REST	X	/users/role	Dados das contas e da role	Status da atualização
UpdateUsersStatus	REST	X	/users/status	Dados das contas e do status	Status da atualização
RemoveEntityUsers	REST	X	/users/delete	Dados das contas	Status da remoção

Tabela 4 - Alterações da API na funcionalidade das organizações

O pedido *GetEntity* possibilita aos utilizadores obterem os dados da sua entidade. O *GetEntityUsers* retorna os utilizadores da entidade. A *payload* desse pedido tem uma estrutura semelhante ao pedido do histórico das assistências, para permitir a utilização de filtros, para que seja possível personalizar a pesquisa. O *GetEntityInvites* permite aos administradores obterem os convites pendentes da organização e, tal como o pedido anterior, é possível alterar a pesquisa com os filtros enviados na *payload*. O *CreateEntity* e o *UpdateEntity*, como os nomes indicam, possibilitam criar ou atualizar a entidade. Atualmente, o único campo que pode ser alterado/definido referente às entidades é o nome. Neste sentido, a *payload* enviada nestes pedidos é o nome da entidade.

O pedido *JoinByEntityInvite* recebe o *id* do convite e é processado de forma diferente consoante o convite efetuado. Caso o utilizador não tenha conta, no pedido é criada uma automaticamente e é associada à entidade. Se o utilizador já tiver uma conta criada e não esteja associada a uma entidade é adicionado à organização automaticamente. Caso já pertença a uma, é devolvido o erro correspondente. O pedido *GetInviteData* retorna os dados do convite, como por exemplo: a data de expiração, os dados da entidade que criou o convite, entre outros.

Os pedidos *InviteUsersToEntity*, *DeleteEntityInvites*, *UpdateUsersRoles*, *UpdateUsersStatus* e *RemoveEntityUsers* recebem uma lista de *ids* ou emails, com a respetiva *role* ou *status* na *payload*, consoante o pedido. Na resposta do pedido é retornada uma lista de *status* de cada uma das operações. No caso do pedido *InviteUsersToEntity*, nos convites realizados com sucesso é automaticamente enviado um email, com o convite, para o endereço do utilizador. Este convite é apresentado na Figura 14. Ao aceitar o convite o utilizador é redirecionado para as aplicações *frontend*, para concluir o processo.

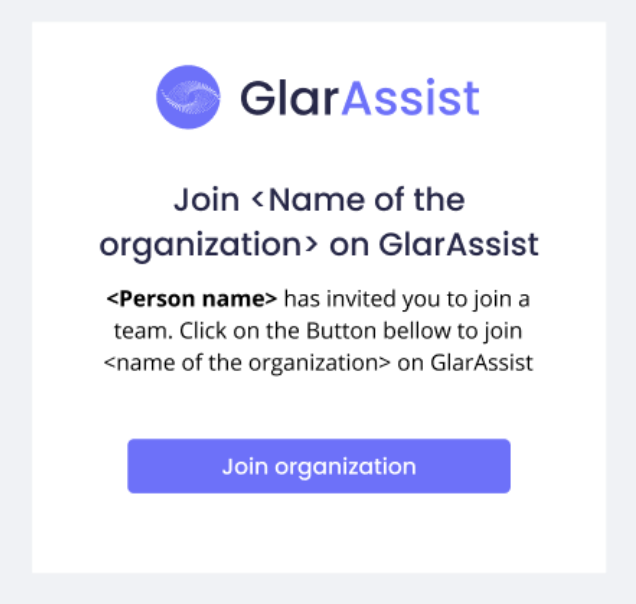


Figura 14 - Convite em formato de email para adicionar utilizadores à organização

Depois do desenvolvimento da funcionalidade no servidor, foram implementadas as UI na aplicação *web*. As interfaces desta funcionalidade são apresentadas da Figura 15-22. Para aceder à página da organização, foi adicionado mais um botão na *sidebar*, apresentado na Figura 15. Atualmente, a página de definições (ver Figura 15) apenas terá a funcionalidade da organização, mas futuramente, é utilizada para integrar outras funcionalidades.

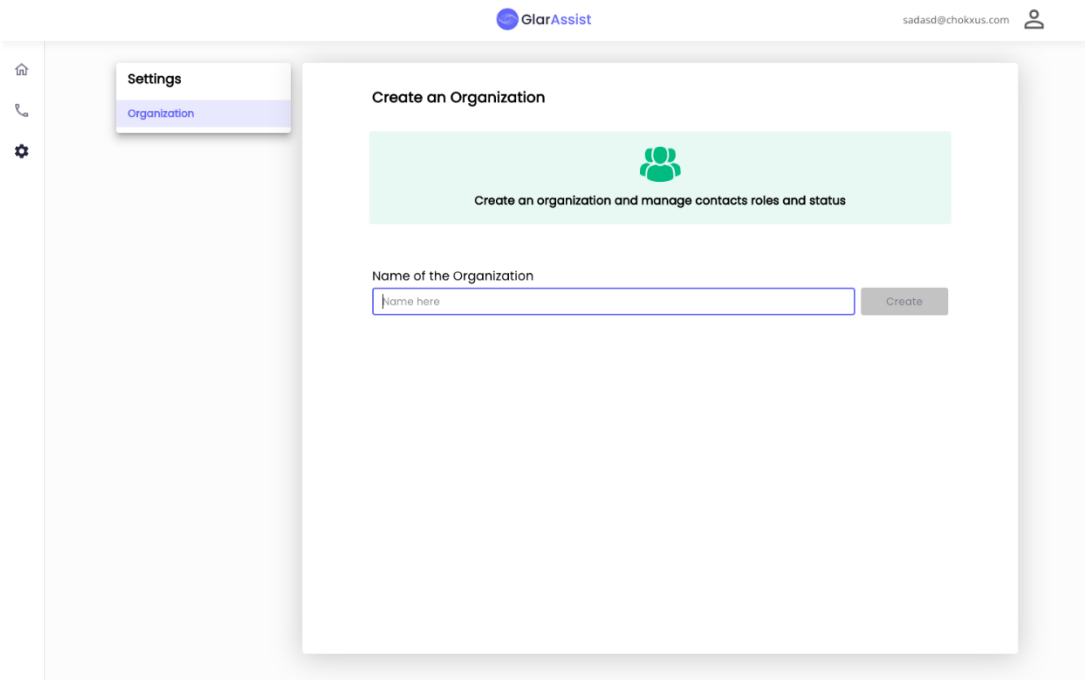


Figura 15 - Componente para criação de uma organização

Quando um utilizador entra na página das Organizações são apresentados dois componentes: caso o utilizador não esteja numa organização, o componente para a criação de uma organização (apresentado na Figura 15); ou caso o utilizador esteja numa organização, o componente para gerir e/ou visualizar a organização (ver Figura 16). Na página de criação, o utilizador poderá preencher o formulário para criar a sua organização. Caso a mesma seja criada com sucesso, o utilizador é redirecionado para a página de gestão da organização, apresentada na Figura 16. O componente de gestão da organização é diferente consoante a *role* do utilizador. A *interface* apresentada na Figura 16 é a vista de administrador. Caso o utilizador não seja administrador são removidas todas as funcionalidades para gerir a organização.

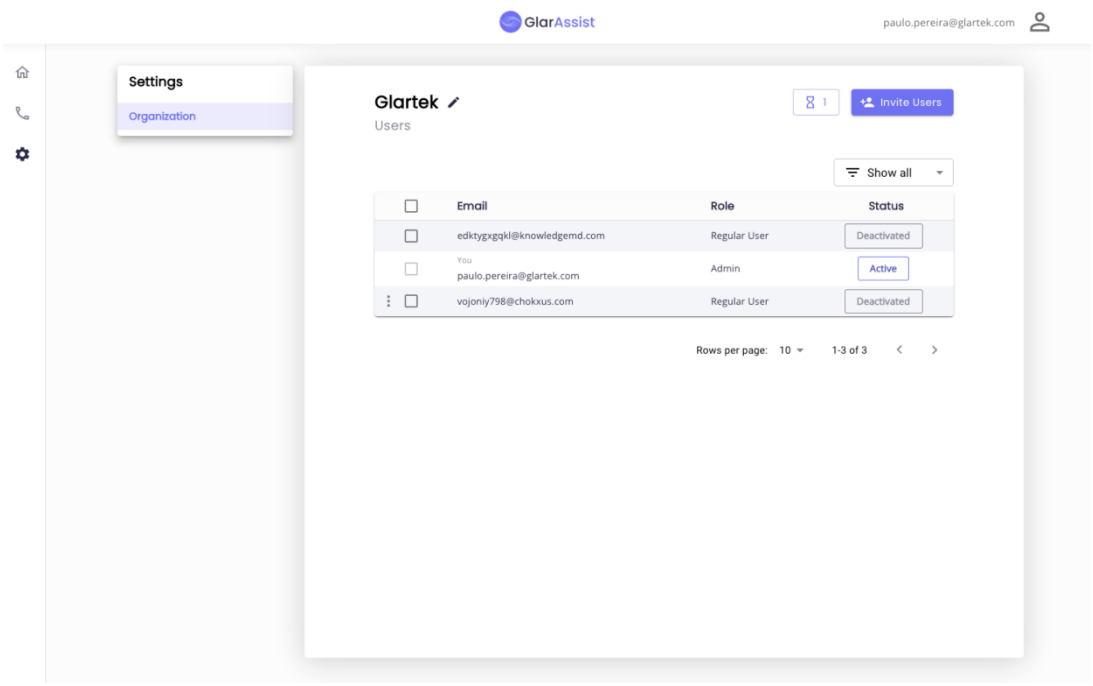


Figura 16 - Componente para gestão e visualização da organização

Em ambos os componentes é possível utilizar os filtros na tabela dos utilizadores. Tal como a tabela de histórico das assistências, é possível personalizar a pesquisa de utilizadores: ordenar a pesquisa pela coluna correspondente, de forma ascendente e descendente ou filtrar pelo *Status* (*input* "*Show all*" visível na Figura 16).

Ao selecionar algum destes *inputs*, a tabela é atualizada consoante a informação selecionada. No componente dos administradores, pode ser alterado o nome da organização. Para isso é exibido um Popover para a alteração do nome, apresentado na Figura 17.

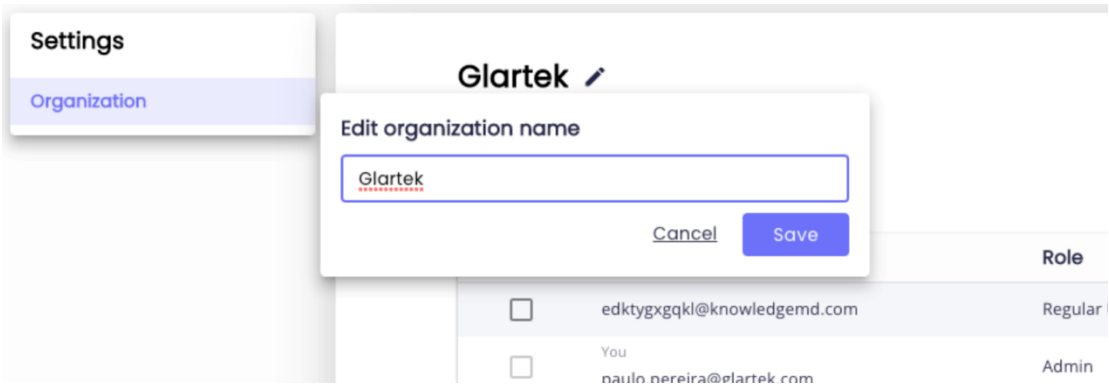


Figura 17 - Popover para alterar o nome da organização

Foi criada uma página para os administradores convidarem utilizadores para a organização (ver Figura 18). Depois de convidar utilizadores, os convites, até serem aceites, podem ser visualizados na página dos convites pendentes, apresentado na Figura 19. No componente dessa página é possível remover convites previamente enviados.

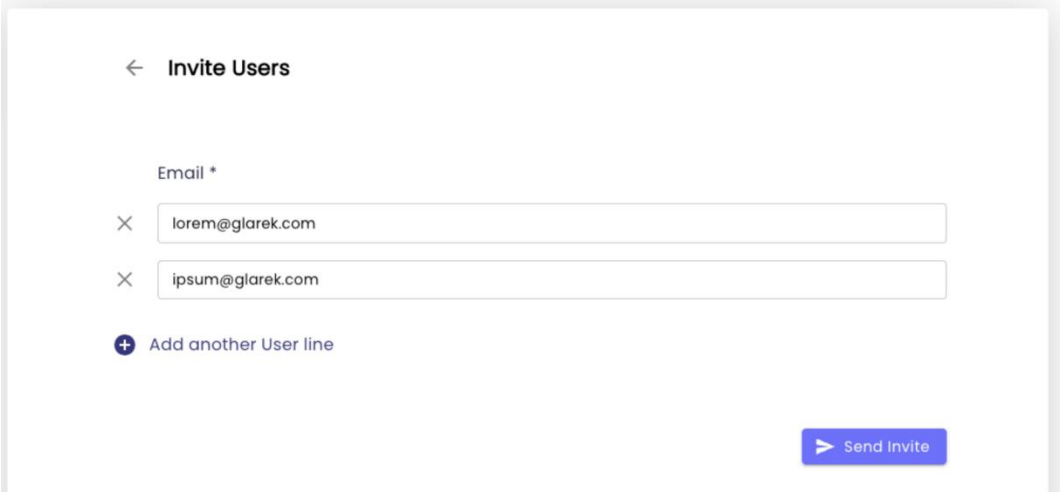


Figura 18 - Componente para convidar utilizadores para a organização

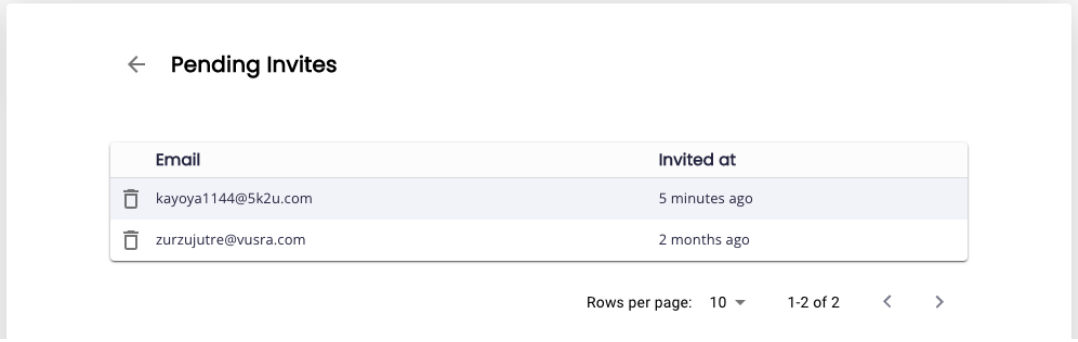


Figura 19 - Componente para apresentar os convites pendentes da organização

Para gerir os utilizadores da organização foram desenvolvidas diferentes formas de o fazer no componente. Para alterar a *role* dos utilizadores apenas é possível alterar um utilizador de cada vez, ao contrário da alteração do *status* e remoção de contas, em que podem ser atualizados vários utilizadores de uma só vez. Na Figura 20 é possível observar a alteração da *role* de apenas um utilizador.

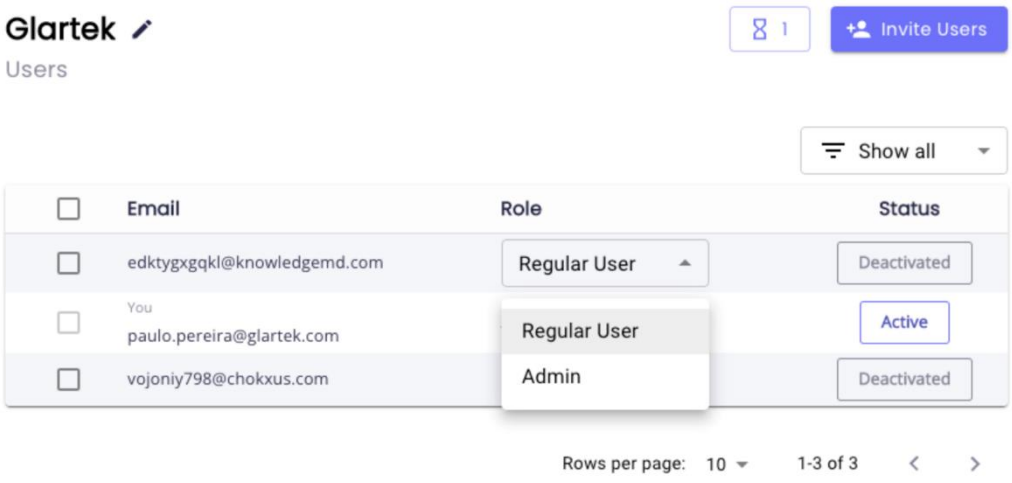


Figura 20 - Alteração da role dos utilizadores na tabela da organização

Caso pretenda alterar múltiplos utilizadores de uma só vez, poderá selecionar individualmente os utilizadores no *input check*, na linha da tabela dos utilizadores correspondentes, ou selecionar todos de uma só vez, no *input check* do *header* da tabela (ver Figura 21). Ao selecionar um ou vários utilizadores, no topo da tabela são exibidos botões para realizar estas operações. Como não é permitido a um utilizador manipular a sua conta, são desativadas todas as funcionalidades para atualizar a sua própria conta.

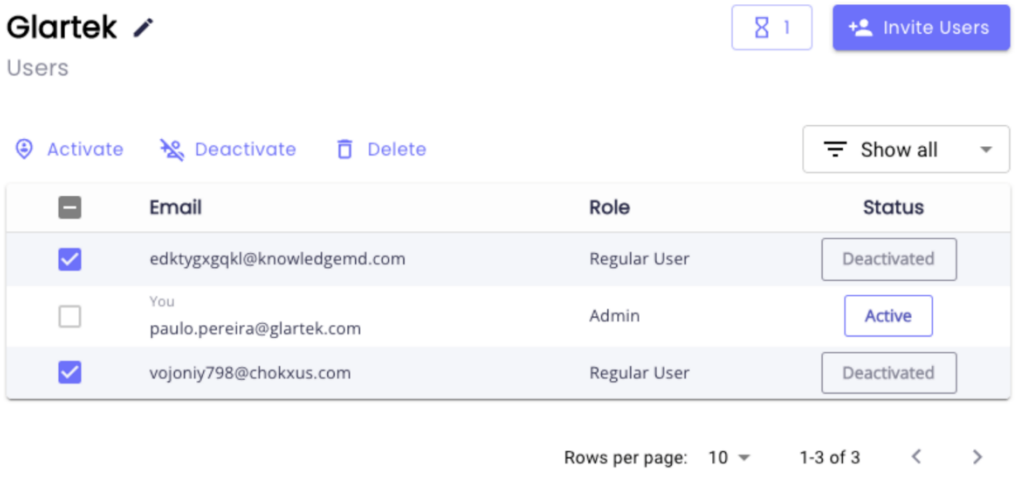


Figura 21 - Alteração de múltiplos utilizadores de uma só vez da organização

Quando um utilizador aceita um convite de uma organização, o mesmo é redirecionado para a aplicação *web*. Caso o utilizador tenha conta criada e aceite um convite de uma organização, destinado à sua conta, o convite é aceite automaticamente e o utilizador é adicionado à organização. Porém, se o convite realizado for direcionado a um email que não tem uma conta associada, o utilizador é redirecionado para uma página específica, apresentada na Figura 22. Nesta página, o utilizador terá de preencher o resto do formulário para a conta ser criada. Ao submeter os dados corretamente a conta é criada e é automaticamente associada à organização.

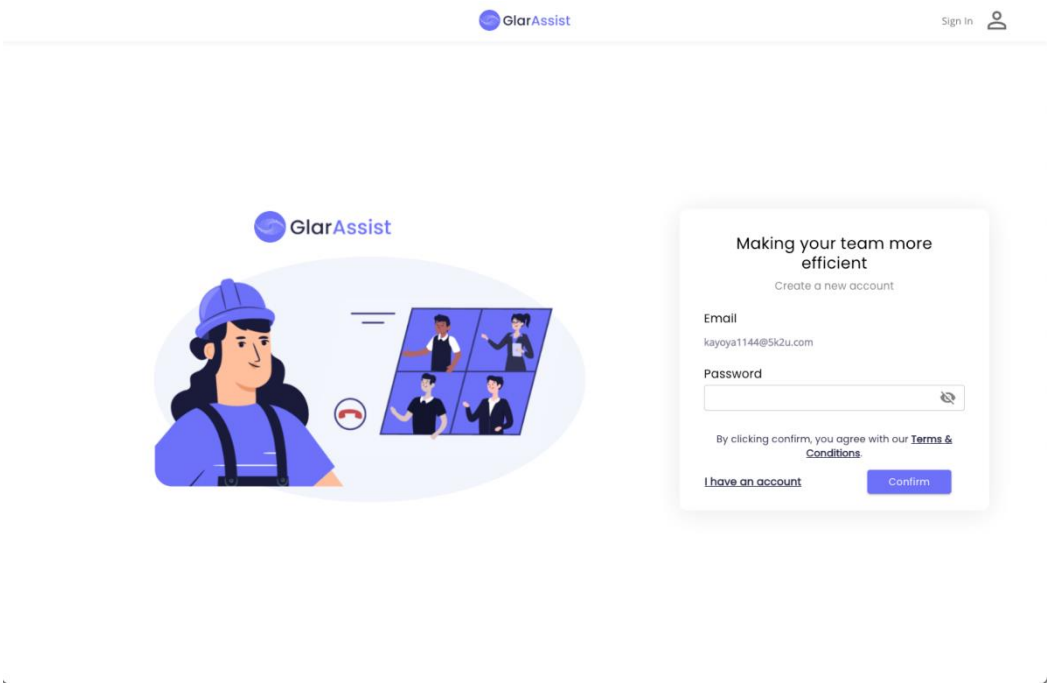


Figura 22 - Aceitar convite de uma organização e criação da conta

4.5. Histórico das sessões da organização

A implementação da *feature* das organizações, possibilitou às empresas ou equipas começar a convidar e gerir os utilizadores das suas organizações. Como definido anteriormente, a partir do momento que um utilizador se juntasse a uma organização, todas as assistências que o mesmo criasse, seriam consideradas assistências da organização. Contudo, não existe a possibilidade de aceder às assistências da organização, pelos administradores das mesmas. Desta forma, surge a necessidade de desenvolver o histórico das assistências da organização. Foi definido pela equipa de produto, que este histórico é

visualizado na página do histórico das assistências do utilizador. Contudo, apenas os utilizadores com a função de administradores, poderão aceder ao mesmo.

Para o desenvolvimento desta funcionalidade, foi necessário acrescentar um novo pedido à API REST do servidor. Este pedido é apresentado na Tabela 5. O objetivo do mesmo é devolver o histórico das assistências da organização. O *GetEntityHistory* recebe e retorna a mesma estrutura do pedido *GetUserHistory* (ver Tabela 2), uma vez que, a informação em ambos os pedidos é semelhante. Por consequência, a *query* utilizada em ambos os pedidos é semelhante, porém, são enviados diferentes parâmetros para retornar à informação pretendida. Ainda assim, ocorreram algumas dificuldades na definição e desenvolvimento da *query* destes pedidos, devido à necessidade de junção de dados de diferentes *collections* e à quantidade de dados retornados. Esta situação é abordada no capítulo seguinte (5).

Nome/Prop.	API	Autenticado	Rota/Evento	*Payload*	Resultado esperado
GetEntityHistory	REST	X	/rooms/history/entity	Filtros definidos	Todas as assistências da entidade

Tabela 5 - Alteração da API na funcionalidade histórico de sessões da organização

No desenvolvimento da aplicação React foi alterado o componente do histórico das assistências do utilizador, para que seja possível apresentar ambos os históricos. A *interface* definida pela equipa de produto é apresentada na Figura 23. Foi acrescentada uma nova rota à aplicação, */history/entity*. Nesta rota, como na rota do histórico do utilizador, é exibido o mesmo componente. O que difere entre as mesmas é que uma apresenta o histórico do utilizador e outra o da entidade. Caso o utilizador não seja administrador, o botão da "Organization" (ver Figura 23) não é exibido, uma vez que, um *Regular User* de uma organização não pode aceder a essa informação.

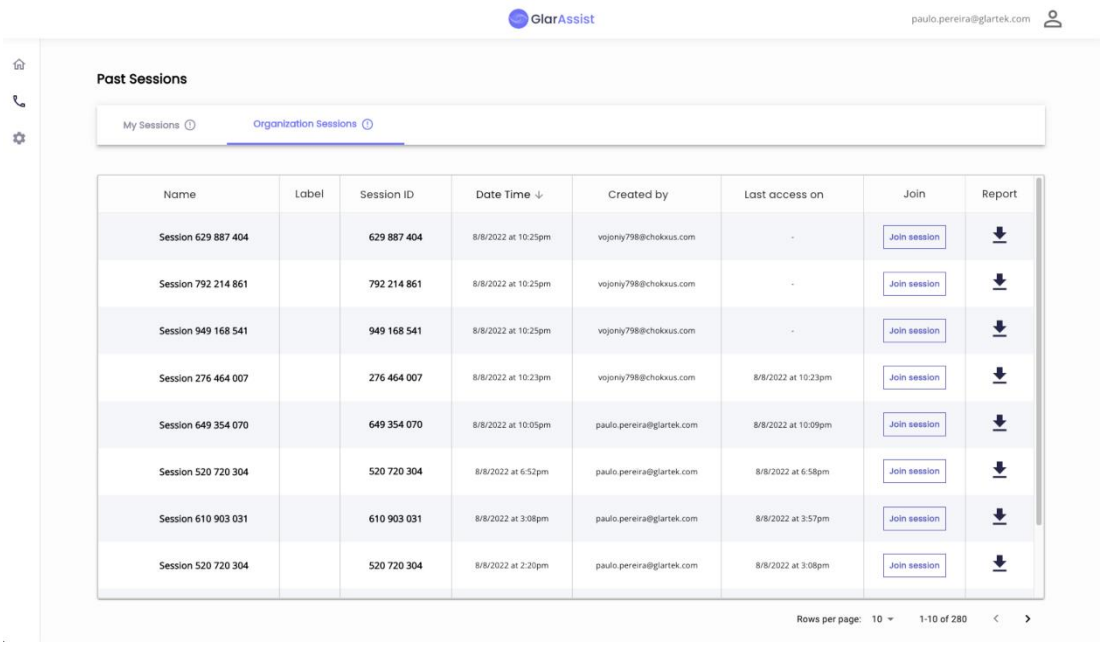


Figura 23 -Componente dos históricos das assistências

4.6. Filtros nas tabelas de histórico

Os históricos desenvolvidos podem-se tornar muito extensos e com uma grande quantidade de informação. Os utilizadores, por vezes, não conseguem recolher toda a informação que pretendem, porque não é possível filtrar a pesquisa da tabela. Com a integração dos diferentes históricos e da possibilidade de identificar assistências, com diferentes títulos e *labels*, é possível filtrar a informação pretendida. Esta funcionalidade é importante, uma vez que, permite aos utilizadores personalizar as suas pesquisas e facilitar a procura de dados específicos. Desse modo, a equipa de produto criou a funcionalidade de filtros nas tabelas dos históricos das assistências.

No *backend* os pedidos *GetUserHistory* (ver Tabela 2) e *GetEntityHistory* (ver Tabela 5) tiveram de ser alterados para aplicar os diferentes filtros que poderiam ser utilizados na pesquisa dos históricos. Foi definida uma estrutura de filtros, que as aplicações *frontend* teriam de enviar, quando pretendem filtrar as tabelas. Os filtros enviados são adicionados à *query* permitindo assim retornar o que é pretendido. Os filtros definidos foram os seguintes:

- Ordenação: este filtro já estava a ser utilizado e permite ordenar os resultados por um campo específico.

- L*abel*: possibilita aos utilizadores procurar por assistências que contenham as *labels* selecionadas.
- Número de participantes: filtrar assistências pela quantidade de participantes nas mesmas.
- Criado por: selecionar se na pesquisa são apenas consideradas assistências criadas pelo utilizador ou por terceiros.
- Data: filtrar as pesquisas por um intervalo de data.
- *Status*: selecionar se na pesquisa são apenas consideradas assistências expiradas ou não.
- Procura de texto: procurar uma certa *string* nos campos título, nome da *label*, código da sala e email do utilizador que criou.

Esta foi a única alteração necessária no servidor, uma vez que, o resto da informação já é fornecida em diferentes pedidos.

Os filtros poderiam ser aplicados em ambos os históricos. Neste sentido, na aplicação React, foram desenvolvidos um construtor de filtros e um visualizador dos filtros selecionados. Estes componentes foram integrados na página dos históricos para permitir filtrar a informação das tabelas dos históricos. Estas alterações são apresentadas nas Figuras 24 e 25. Na Figura 24 é possível observar que foram as alterações realizadas no componente do histórico. No mesmo, foram adicionados dois *inputs* ao *header* da página.

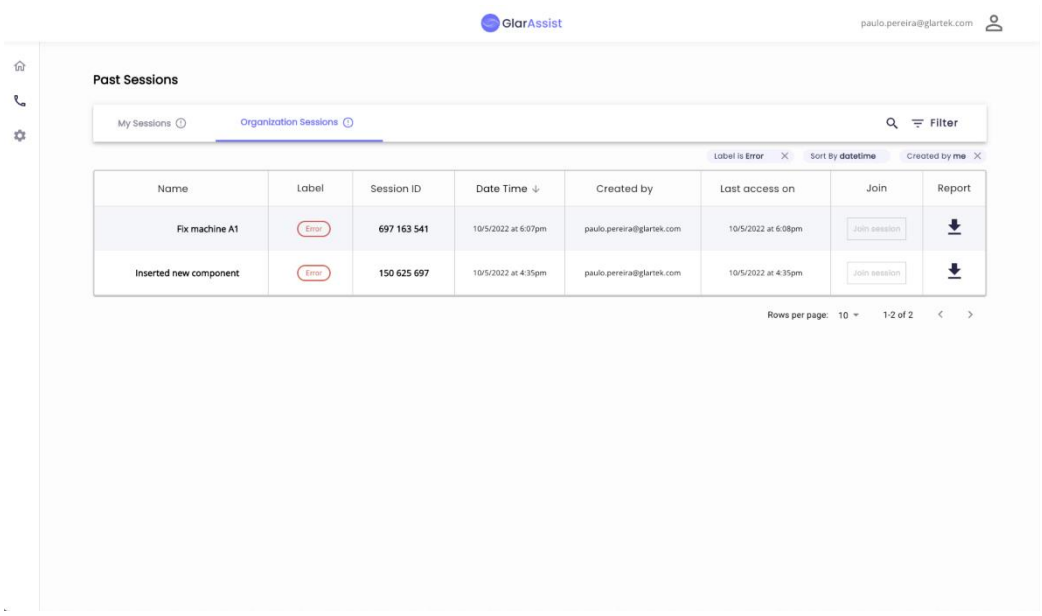


Figura 24 - Componente do histórico das assistências com filtros ativos

O primeiro *input* representa o filtro de procura de texto. Quando o utilizador preenche o *input*, o filtro é automaticamente adicionado. Para não requerer a informação filtrada cada vez que o utilizador introduz um caracter, evitando pedidos desnecessários, foi desenvolvida uma funcionalidade para apenas realizar o pedido depois do utilizador parar de preencher o *input*. Quando o utilizador e o intervalo de tempo definido terminar, é verificado o que o utilizador inseriu. Se a *string* for diferente da última pesquisa é realizado um novo pedido para atualizar a informação. O outro *input* inserido foi o botão "Filter" para apresentar o construtor de filtros (ver Figura 25).

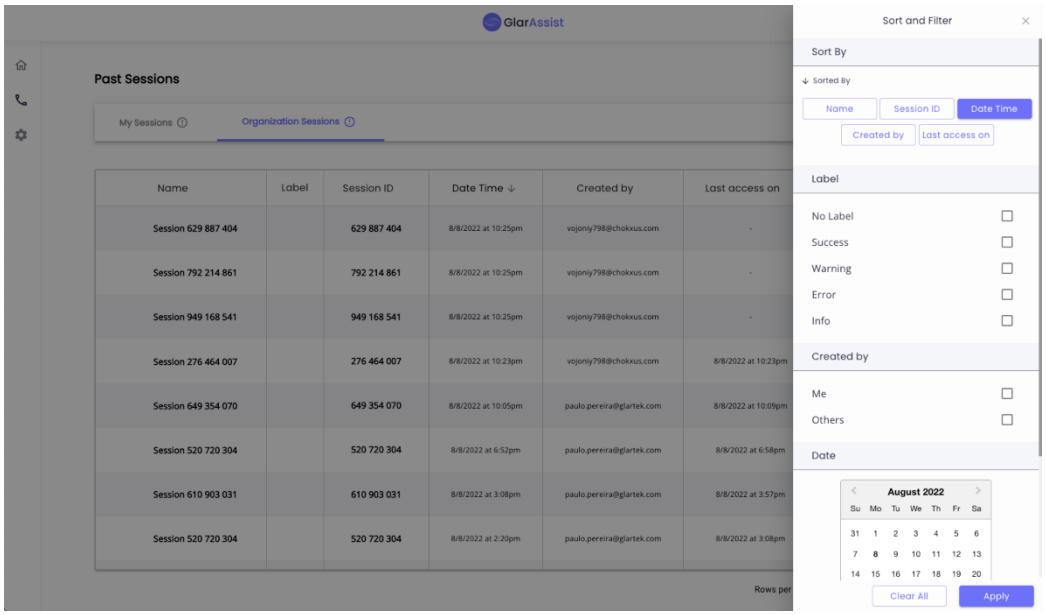


Figura 25 - Sidebar com o construtor de filtros

O componente do construtor de filtros aparece como uma *sidebar*, mas sobreposta ao componente da página. Neste componente é possível selecionar as restantes opções de filtragem. Para cada tipo de filtro existe um conjunto de *inputs* diferentes. No componente existem duas ações para aplicar ou remover os filtros selecionados. A pesquisa com filtros ativos pode ser visualizada na Figura 24. Cada filtro pode ser removido diretamente no botão do filtro correspondente. Caso o utilizador altere o histórico, os filtros são mantidos de um histórico para o outro.

4.7. Alteração do volume do utilizador assistido

A presente funcionalidade não é relacionada com o BackOffice do GlarAssist. Contudo, a mesma foi requerida durante o desenvolvimento do projeto. Apesar de não influenciar a documentação recolhida, como apresentado no capítulo da contextualização, as aplicações de assistência remota com RA têm de proporcionar soluções "mãos livres". Desta forma, esta funcionalidade é importante, pois a razão da mesma ter sido criada é devido aos colaboradores assistidos terem dificuldades em alterar o volume de dispositivos *simple devices*, como por exemplo os Vuzix. A *interface* dos mesmos, geralmente, não é fácil de usar, e, por vezes, os operadores têm as mãos ocupadas nas suas tarefas durante a assistência. Por este motivo, a equipa de produto criou a funcionalidade e os respetivos *mockups*.

Esta funcionalidade não requer nenhuma implementação no servidor, uma vez que, estas mensagens são transmitidas apenas por aplicações *frontend*, mais especificamente pelo serviço Agora RTM. As mensagens trocadas no serviço Agora RTM são apenas utilizadas para realizar as operações da assistência, como por exemplo a troca de coordenadas na funcionalidade de desenho em RA. Em reunião, a equipa de desenvolvimento definiu a mensagem que seria utilizada para alterar o volume do assistido. A aplicação *web*, como *role* de assistente, poderá alterar o valor do volume do assistido. As alterações realizadas na UI são apresentadas na Figura 26. Nesta *feature* foi adicionado um botão às ações do assistido, onde o assistente pode alterar o volume do dispositivo do assistido. Este botão está apenas habilitado se o assistido estiver a utilizar um Simple Device. Quando o utilizador define o volume é enviado o evento para o assistido e o volume é automaticamente definido no dispositivo do assistido. Assim o volume dos dispositivos dos assistidos pode ser gerido, permitindo aos operadores continuarem as suas operações sem interrupções.

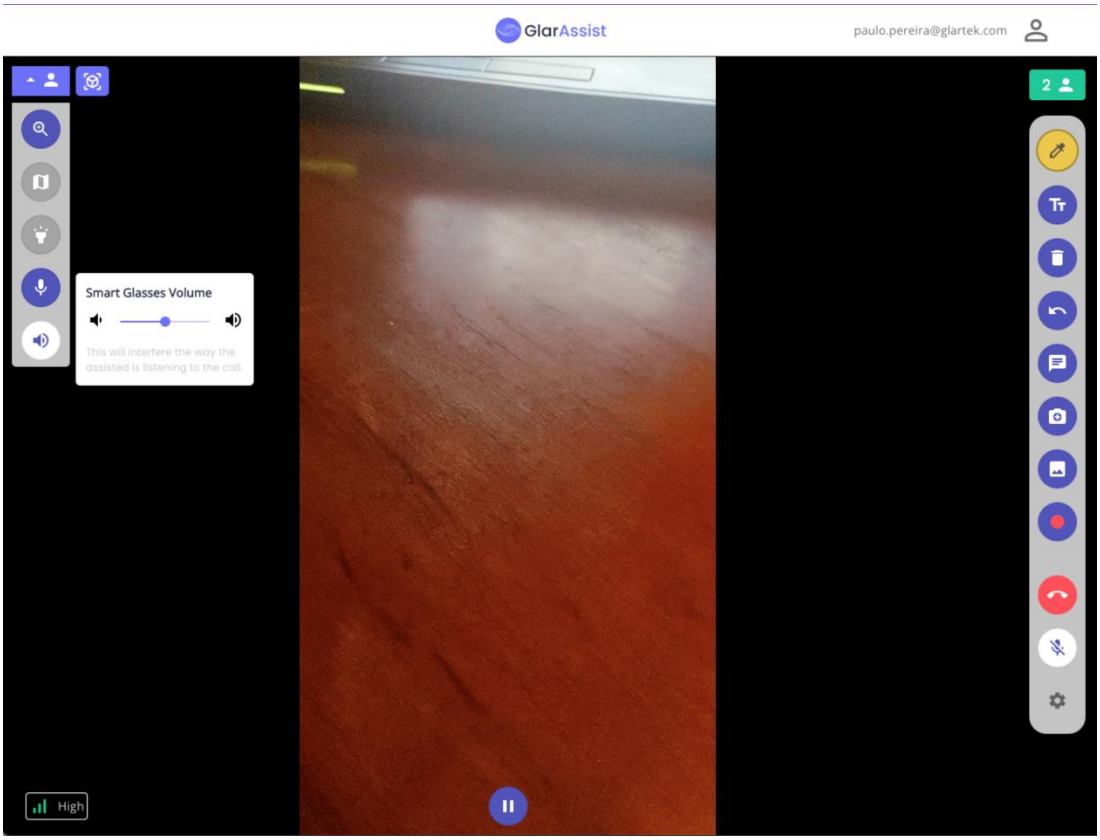


Figura 26 - Funcionalidade para alterar o volume do utilizador assistido

4.8. Gestão de conta dos utilizadores

No desenvolvimento da funcionalidade da organização, apenas foi utilizado o email para identificação dos utilizadores. A identificação dos utilizadores é importante e, para isso, é necessário adicionar outras informações, para além do email. Atualmente, na aplicação, não é possível definir e alterar dados das contas dos utilizadores, como por exemplo nome, foto de perfil, entre outros. Para além da falta de informação das contas, não é possível alterar a *password* das mesmas. Esta situação dificulta a gestão dos utilizadores da organização porque, por vezes, não é possível identificar o utilizador apenas pelo seu email. Por consequência, foi definida a presente funcionalidade para os utilizadores poderem inserir informações relevantes nas suas contas. Com a mesma, os utilizadores podem definir o seu primeiro e último nome e a sua foto de perfil. Para além da alteração dos dados da conta, foi desenvolvida a funcionalidade para os utilizadores alterarem a *password* das suas contas.

Para o desenvolvimento destas funcionalidades foi necessário alterar a estrutura de utilizador para armazenar esta informação. Foram, ainda, adicionados novos pedidos à API

REST. Como as fotos de perfil têm de ser armazenadas no *bucket*, foi definido um caminho específico onde são guardadas as fotos dos utilizadores. Para alterar esta informação foram criados os pedidos apresentados na Tabela 6.

No pedido *UpdateUser*, no corpo do pedido, são enviados os dados que o utilizador pretende atualizar. Como referido anteriormente, podem ser atualizados os campos primeiro e último nome e a foto de perfil. Se for necessário atualizar a foto é gerado um *presigned POST url* para a mesma ser armazenada no *bucket*. O pedido *UpdateUserPassword* permite aos utilizadores atualizarem a sua *password*. Para realizar o pedido tem de ser enviada a *password* atual e a nova *password*. O pedido *GetUserPhoto* retorna a foto de perfil do utilizador. Estes três pedidos permitem às aplicações cliente obterem e gerirem estas informações dos utilizadores.

Nome/Prop.	API	Autenticado	Rota/Evento	*Payload*	Resultado esperado
UpdateUser	REST	X	/user/update	Dados do utilizador	Dados atualizados e *presigned POST url* se necessário atualizar a foto
UpdateUserPassword	REST	X	/user/update/password	Nova *password* e a atual	Status 200
GetUserPhoto	REST	X	/user/photo	-	*Presigned GET url* da foto de perfil

Tabela 6 - Alterações da API na funcionalidade gestão de utilizadores

Na aplicação *web* foram alteradas algumas funcionalidades previamente desenvolvidas e criados novos componentes para possibilitar gerir esta informação. As alterações foram realizadas na apresentação dos utilizadores em toda a aplicação. Isto é, anteriormente, os utilizadores eram apenas identificados pelo seu email. Com esta funcionalidade, se o utilizador tiver definido o primeiro e último nome, é utilizado o nome, em vez do email, para identificar o utilizador. Um exemplo onde esta alteração foi realizada é na tabela dos utilizadores da organização.

A foto dos utilizadores apenas é apresentada ao próprio utilizador. Futuramente, a foto será utilizada em diferentes funcionalidades, como a lista de contactos dos utilizadores. As UI desenvolvidas são apresentadas nas Figuras 27-30. A página de definições da aplicação *web* foi desenvolvida com o objetivo de apresentar diversas funcionalidades. Neste sentido, as funcionalidades atualização dos dados do perfil e atualização da *password* foram inseridas neste componente, pois são definições da conta. Consequentemente, na *tab* lateral foram adicionados os itens para o utilizador aceder às respetivas funcionalidades (ver Figura 27).

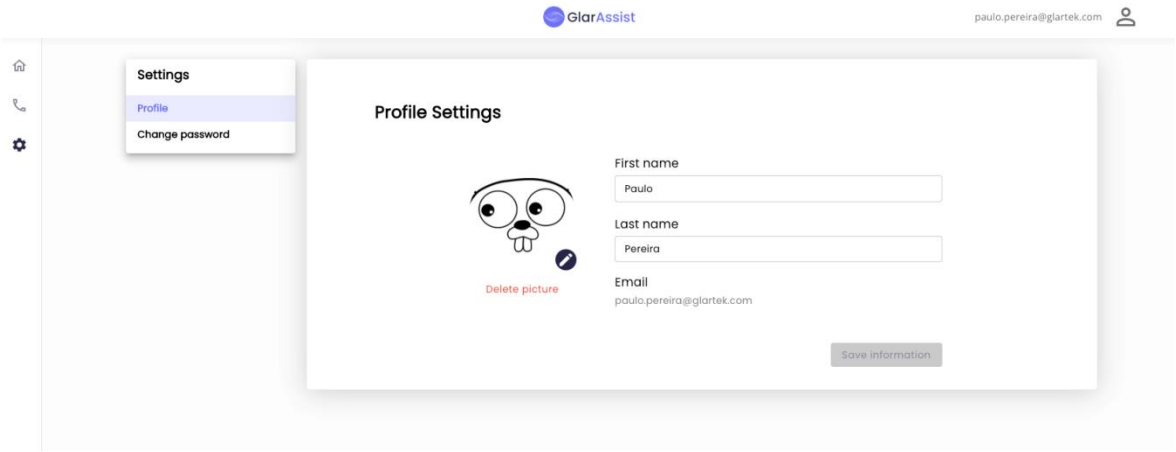


Figura 27 - Gestão do perfil do utilizador

Na página para gerir o perfil do utilizador, apresentada na Figura 27, o utilizador pode alterar a sua foto de perfil e o primeiro e último nome. Se o utilizador tentar sair para outra rota e tiver informação por guardar, é exibido um Dialog a informar que o utilizador tem dados que ainda não foram atualizados (ver Figura 28).

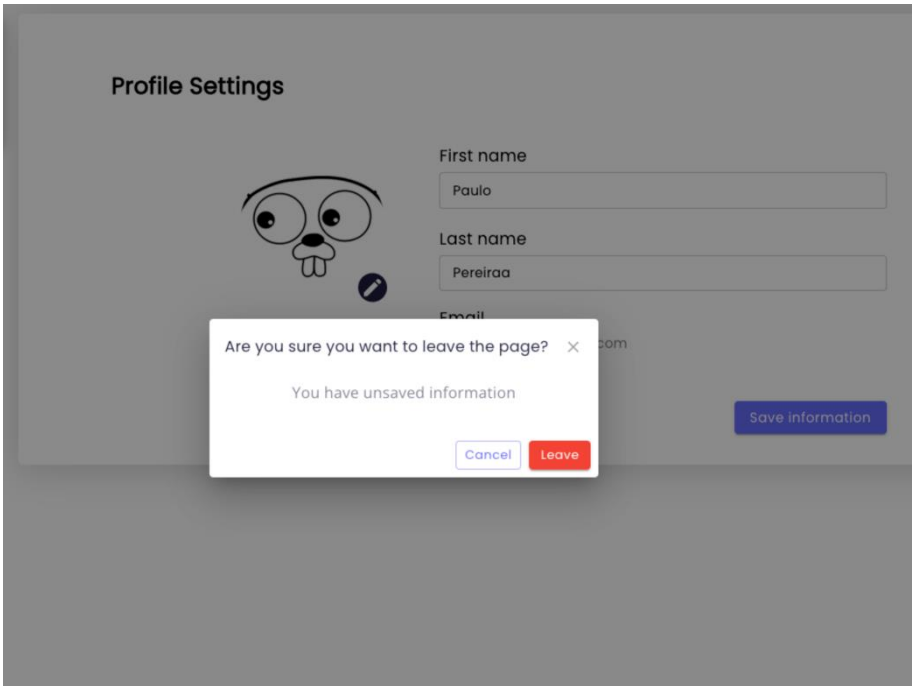


Figura 28 - Dialog de confirmação quando existem dados por guardar

A página para alterar a *password* do utilizador é apresentada na Figura 29. Na mesma o utilizador consegue atualizar a sua *password*. Como na página anterior o Dialog de confirmação é exibido se o utilizador quiser alterar de rota e tiver informação por guardar (ver Figura 28).

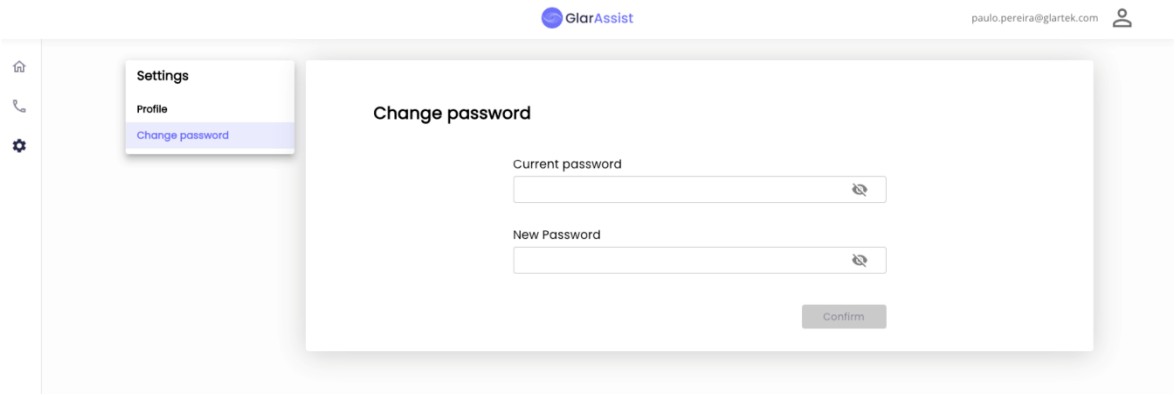


Figura 29 - Componente para alterar a password de uma conta

Além da implementação das páginas para gerir a informação, foi atualizado o Popover da *navbar*, apresentado na Figura 30. No mesmo foi adicionada a foto de perfil e o primeiro e último nome do utilizador. Também foi adicionado um novo item à lista, caso o utilizador queira aceder à página de gestão do perfil do utilizador.

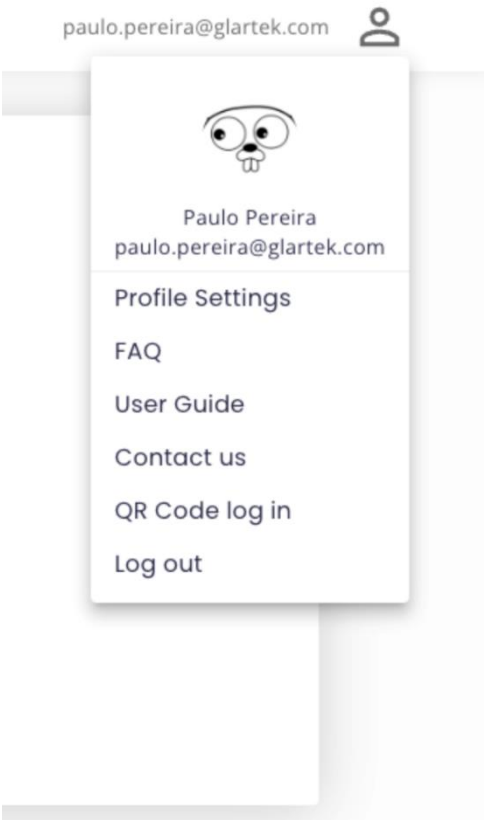


Figura 30 - Popover da navbar da aplicação web

4.9. Feature flags

Devido às constantes alterações na aplicação, é difícil manter o normal funcionamento nas aplicações antigas ou desatualizadas, em todas as funcionalidades. Como aplicações de diferentes versões podem, também, realizar assistências, por vezes há comportamentos inesperados durante as assistências. Para que seja possível integrar aplicações de diferentes versões e, ao mesmo tempo, evitando estas situações, foram implementadas as *feature flags*. Segundo a pesquisa efetuada pela equipa de desenvolvimento, as *feature flags* são um mecanismo de restrição baseado na versão semântica (exemplo, 0.19.0). Isto significa, que quando uma aplicação utiliza uma determinada funcionalidade, verifica com o servidor a sua compatibilidade para a usar. Neste sentido, é possível desabilitar funcionalidades dinamicamente, evitando, assim, estes comportamentos inesperados. Para a utilização deste mecanismo foi definida uma estrutura no servidor, onde é guardada a versão de cada funcionalidade do GlarAssist.

Quando existe uma *breaking change* numa funcionalidade, a versão da mesma é atualizada. A validação de uma funcionalidade é feita por restrições. Por exemplo, na estrutura está definido que a Feature1 é "Feature1 >= 0.19.0", ou seja, apenas aplicações com a versão "0.19.0" ou superior é que podem utilizar esta funcionalidade. Esta abordagem evita que aplicações com versões antigas se tornem obsoletas, uma vez que, são apenas desativadas as funcionalidades que não podem ser executadas, mantendo assim estabilidade nessas versões. Esta funcionalidade foi requerida pela equipa de desenvolvimento, uma vez que, estavam a ser reportados diversos comportamentos inesperados em aplicações antigas.

Quando é atualizado o ambiente de produção, atualmente, o servidor e a aplicação *web* são atualizadas ao mesmo tempo. Por consequência, a aplicação *web* está sempre na versão mais recente. A aplicação Android, devido a diferentes validações da PlayStore, demora mais tempo a ficar atualizada. Contudo, o objetivo principal é suportar as aplicações previamente instaladas, que nunca estarão na versão mais recente.

Como referido acima, é necessário definir uma lista com todas as funcionalidades onde poderão existir as *breaking changes*. Depois de estabelecida essa estrutura, foi determinada uma versão para cada uma das funcionalidades. Para as aplicações cliente não terem de realizar um pedido ao servidor para verificarem se podem ou não executar a funcionalidade, sempre que a vão utilizar pela primeira vez, foi decidido criar um pedido em que são

validadas logo todas as funcionalidades de uma só vez. As aplicações *frontend* ao iniciarem, fazem esse pedido ao servidor e é retornado o estado de todas as funcionalidades, consoante a versão da aplicação.

De forma a fornecer esta lista a todas as aplicações foi criado um pedido novo na API REST, apresentado na Tabela 7. O pedido *GetFeatureStatus* recebe a versão da aplicação, verifica quais as funcionalidades que a aplicação pode executar e retorna o objeto com o estado de cada funcionalidade. Para além do pedido criado, foi também adicionada uma lista de funcionalidades da assistência aos detalhes da sessão de cada utilizador. Assim quando existe uma assistência, com aplicações de diferentes versões, é possível perceber quais as funcionalidades da assistência que estão disponíveis.

Nome/Prop.	API	Autenticado	Rota/Evento	*Payload*	Resultado esperado
GetFeatureStatus	REST		/config/features	Versão da aplicação	Lista de funcionalidades com o seu estado

Tabela 7 - Alteração da API na funcionalidade feature flags

Esta funcionalidade foi direcionada para as aplicações Android desatualizadas, contudo, a mesma foi desenvolvida na aplicação *web*. Apesar da atualização da aplicação *web*, no ambiente produtivo, ser realizada ao mesmo tempo do servidor, a aplicação terá de suportar a assistência a aplicações Android desatualizadas e não é garantido que o processo de atualização do ambiente produtivo não possa alterar no futuro. Neste sentido, desenvolvendo esta funcionalidade na aplicação React nunca existirão os problemas previamente identificados.

Para adicionar a funcionalidade, foi criado um Context Provider, do React, na raiz do projeto, para que seja possível aceder à lista de funcionalidades em qualquer componente da aplicação. Para além da realização do pedido à API REST, foi adicionada na jornada de cada funcionalidade uma validação, com o objetivo de validar se a mesma está disponível para o utilizador. Caso não esteja é exibida uma mensagem de erro, apresentada na Figura 31. Para além da validação das funcionalidades da aplicação *web*, também tem de ser verificado se o assistido tem disponíveis as funcionalidades de assistência (desenho, inserir texto, *freeze*, entre outras). Caso alguma delas não esteja disponível, o assistente é notificado que o assistente tem a aplicação desatualizada, com um alerta como o da Figura 31 (com uma

mensagem diferente). Com esta implementação é possível realizar assistências, com aplicações de diferentes versões, desativando apenas as funcionalidades não suportadas, evitando que a aplicação do assistente seja inutilizável.

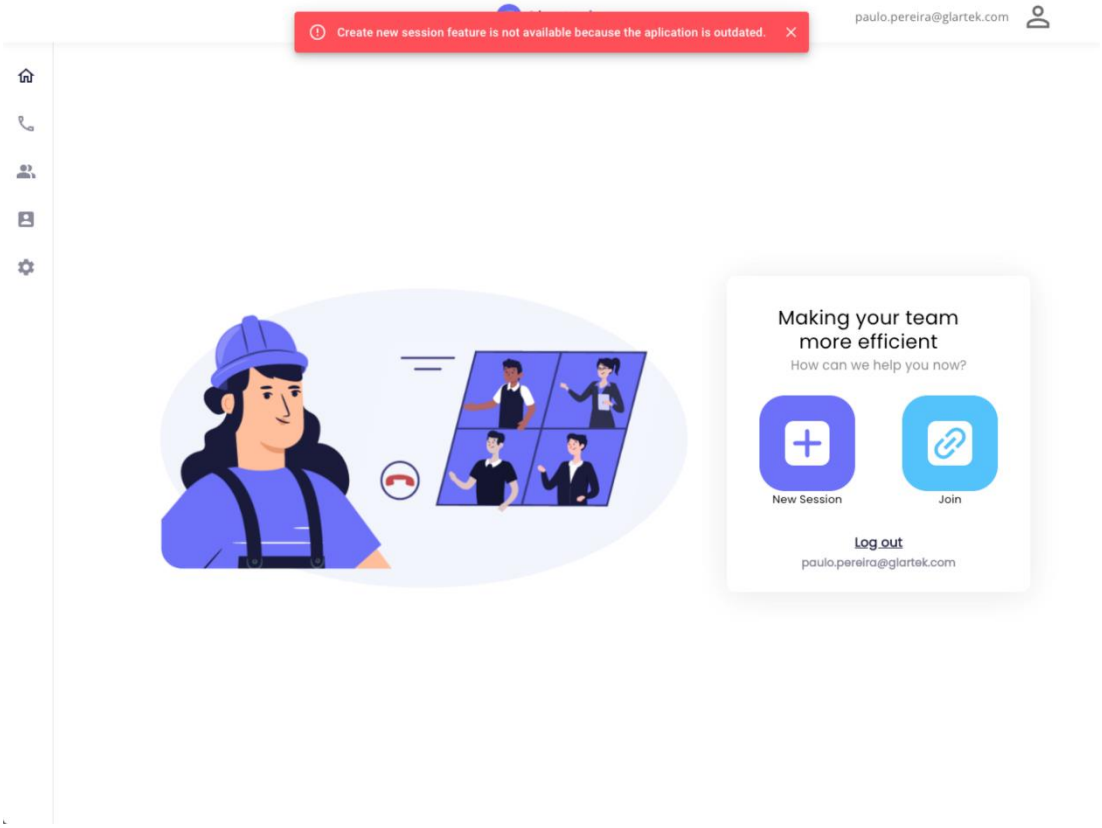


Figura 31 - Alerta de uma funcionalidade não disponível

4.10. Lista de contactos dos utilizadores

Como foi identificado na análise competitiva, a lista de contactos é uma das funcionalidades mais utilizadas nas aplicações de assistência remota com RA. Isto demonstra a importância de comunicar diretamente com utilizadores nas aplicações de AR para realizar assistências. Nesta fase inicial é pretendido que os utilizadores possam adicionar utilizadores à sua lista de contactos, para depois facilitar o convite dos mesmos, para as suas assistências. O convite neste momento irá ser apenas por email. Futuramente, será possível utilizar outras formas de convidar ou conectar com utilizadores, por chamadas diretas, agenda de assistências, entre outras. A equipa de produto definiu toda a jornada e os *mockups* necessários para o desenvolvimento da funcionalidade.

Esta funcionalidade está divida em duas partes, a lista de contactos e o envio de convites para os contactos. Para a realização da lista de contactos foi criado um modelo no servidor, para armazenar os contactos dos utilizadores na base de dados. Este modelo, que representa um contacto e a sua estrutura, é composta por: o *id* do *User* do contacto, o *id* do *User* que contém o contacto, entre outros. Para gerir a lista de contactos foram adicionados diferentes pedidos à API REST. Esses pedidos são apresentados na Tabela 8.

Nome/Prop.	API	Autenticado	Rota/Evento	*Payload*	Resultado esperado
GetUserContacts	REST	X	/contacts/user	Filtros definidos	Lista de contacto
AddContacts	REST	X	/contacts/add	Lista de emails	Contactos criados
DeleteContact	REST	X	/contacts/delete	Lista de *ids* de contactos	Status 200
CheckIfContactsExist	REST	X	/contacts/exist	Lista de *ids* de utilizadores	Lista de status dos *ids* dos enviados

Tabela 8 - Alterações da API na funcionalidade lista de contactos do utilizador

O *GetUserContacts* devolve a lista de contactos do utilizador. Caso seja necessário filtrar a *query*, é possível adicionar filtros (ordenação, por campo, por texto, entre outros). O pedido *AddContacts* recebe uma lista de emails e cria os contactos do utilizador. O *DeleteContacts* recebe uma lista de *ids* de contactos de utilizador e remove-os da lista. Por fim, o pedido *CheckIfContactsExist* recebe uma lista de *ids* de utilizadores e verifica se os mesmos já estão adicionados à lista de contactos. Este pedido foi criado devido a uma funcionalidade, que permite adicionar contactos diretamente da lista de utilizadores. Para convidar utilizadores para as assistências já existe um pedido na API REST, que já estava a ser utilizado e onde é possível enviar um convite para uma lista de emails.

No desenvolvimento da aplicação *web* foram alteradas algumas funcionalidades e criadas novas UI. Foi acrescentada uma nova página para os utilizadores acederem à lista de contactos. Todas as UI desenvolvidas são apresentadas nas Figuras 32-35. De forma a aceder à lista de contactos, foi adicionado um novo botão na *sidebar*, apresentado na Figura 32. Caso não existam contactos criados é apresentado o componente da Figura 32.

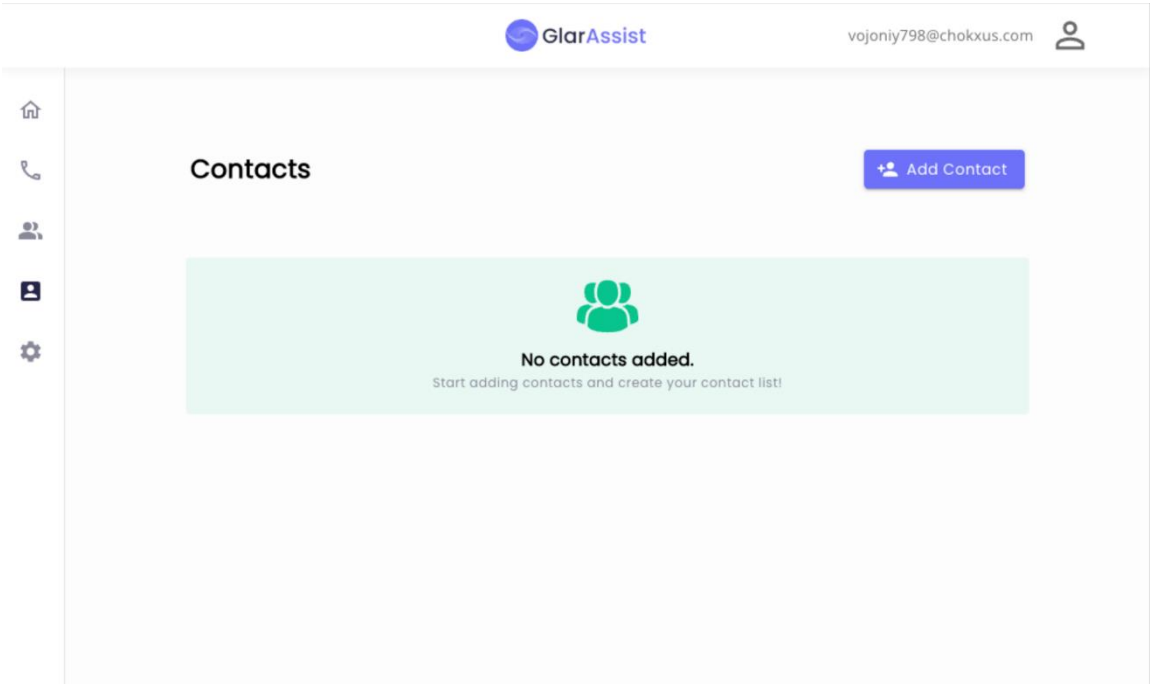


Figura 32 - Componente da lista de contactos sem contactos

Se o utilizador pretende adicionar novos contatos é exibido o Dialog apresentado na Figura 33. No Dialog o utilizador pode adicionar múltiplos contactos de uma só vez. Ao submeter o formulário, são criados os contactos e o Dialog é fechado.

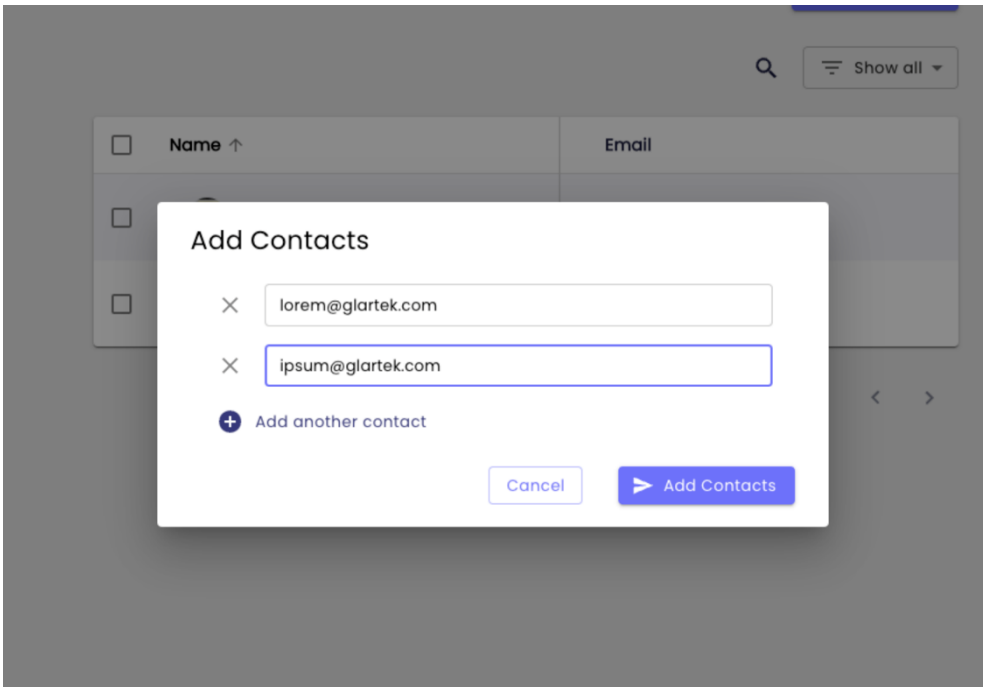


Figura 33 - Componente para adicionar novos contactos

Se o utilizador tiver pelo menos um contacto inserido é exibido o componente da Figura 34. Na tabela dos contactos é apresentada a seguinte informação do contacto: o nome ou email do utilizador, a foto de perfil do utilizador, se pertence à sua organização e o email. Como na tabela dos utilizadores da organização é possível selecionar utilizadores específicos, no *input check* da linha correspondente ao contacto ou todos, no *input check* do *header*, para gerir os mesmos.

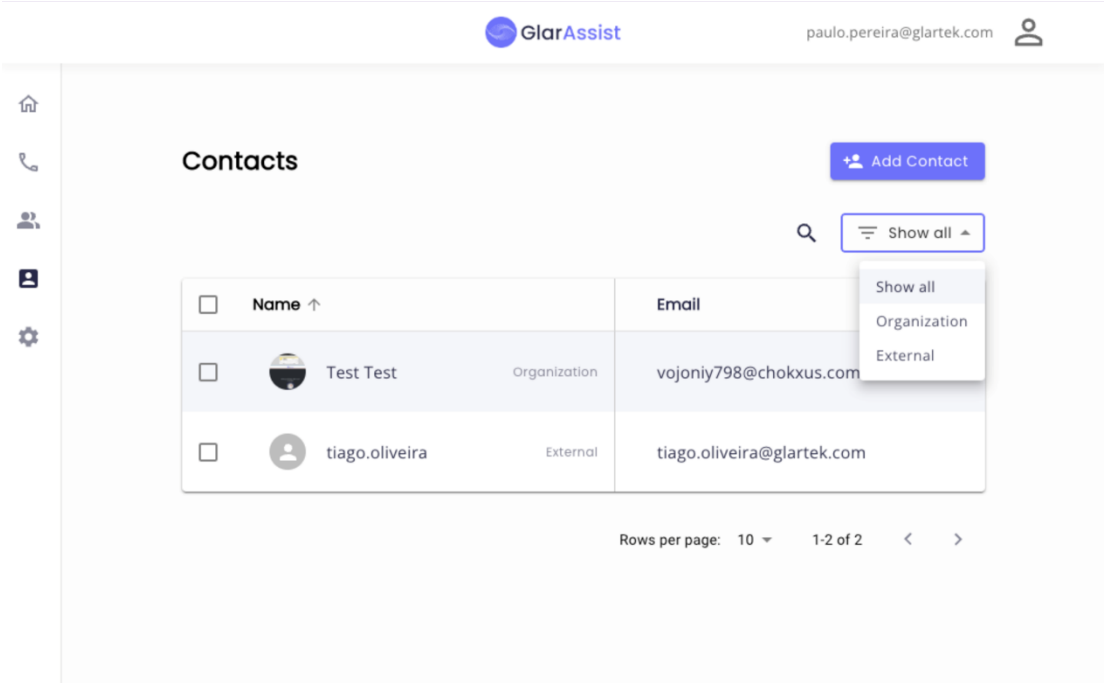


Figura 34 - Componente da lista de contactos com contactos

Para além da gestão, é possível personalizar a pesquisa da tabela. Junto dos *headers*, na tabela, é possível ordenar a tabela pelo um campo específico. Também é possível filtrar por contactos da organização ou de outras organizações e ainda utilizar o filtro de procura de texto. Este filtro procura pela *string* escolhida no campo do nome e email.

Para um utilizador convidar outro utilizador da sua lista de contactos, foi integrada uma nova funcionalidade dentro do componente das assistências. No componente para partilhar a assistência, é exibido o componente para convidar utilizadores por email, apresentado na Figura 35.

Quando o utilizador começa a introduzir um email ou nome de um contacto, nesse *input*, são carregados contactos que contenham essa *string* no nome ou no email. Caso exista algum contacto, são apresentados como opções no *input* (ver Figura 35), com as informações sobre o contacto: a foto de perfil, nome, email e se pertence à sua organização. Caso o email não exista na lista de contactos, o utilizador pode selecionar o email e o mesmo é adicionado às

opções selecionadas. Ao submeter as opções selecionadas, é realizado o pedido à API REST e são enviados os convites por email aos utilizadores correspondentes. Apesar de ser apenas um *input* de seleção, o componente original do MUI teve de ser todo alterado/manipulado e para além dessas alterações, tiveram de ser realizadas diferentes validações para obter os contactos nas opções do *input*. O desenvolvimento deste *input* é abordado no capítulo 5.

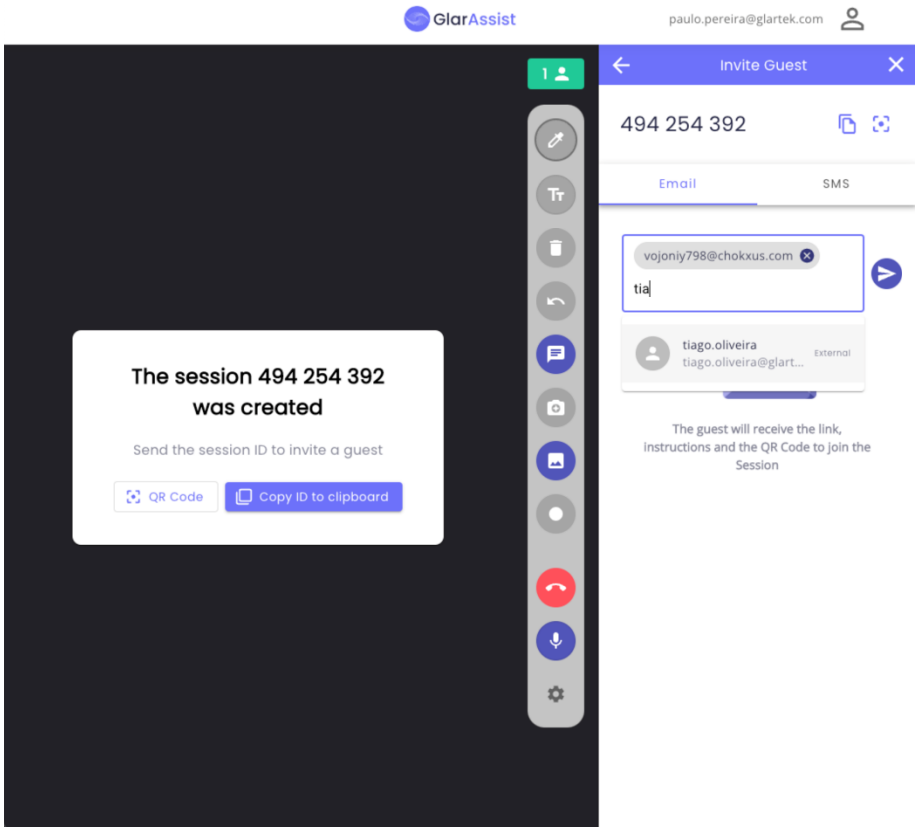


Figura 35 - Input para convidar utilizadores para a sala, com opções selecionadas

4.11. Autenticação centralizada

Atualmente, as plataformas de SSO são utilizadas na maioria das organizações. Com estas plataformas, os utilizadores apenas necessitam de se autenticar numa plataforma e essa autenticação pode ser utilizada noutras plataformas ou serviços que estejam configurados com a plataforma de SSO. Desta forma, os colaboradores têm acesso a todos os serviços da sua organização, apenas com um *login*. O SSO torna-se essencial para as organizações, porque possibilita às organizações configurarem a sua plataforma, para os seus colaboradores se autenticarem em diversos serviços. Com SSO integrado na aplicação, os colaboradores conseguem, assim, autenticar-se na aplicação com as contas da sua

organização. Caso a conta de um utilizador não exista no GlarAssist, a conta é automaticamente criada pelo SSO. Para realizar o SSO a aplicação utiliza a autenticação com SAML para a comunicação ser entre pelos *providers* (*indentity provider* e *service provider*). Para que os utilizadores consigam configurar as suas plataformas nas organizações é necessário armazenar a informação referente às plataformas.

A equipa de produto desenvolveu esta funcionalidade e os seus *mockups*. Durante o desenvolvimento, a equipa de desenvolvimento e de produto estiveram em diversas reuniões, uma vez que, os dados da configuração das plataformas, nos *mockups* iniciais, não estavam corretos. Também ao longo do desenvolvimento foram acrescentados outros campos e informações, para auxiliar os utilizadores na configuração da plataforma de SSO da sua organização.

No desenvolvimento do servidor foi adicionada toda a lógica para utilizar este tipo de serviços. Também foram alteradas as estruturas das entidades para armazenar os novos dados. Foi adicionada uma estrutura, para que seja possível guardar os dados referentes ao SAML das organizações. Para além desses dados, foram adicionados dois campos para armazenar os atributos do utilizador. Caso sejam atributos da plataforma, são guardados automaticamente na estrutura do utilizador. Se forem outros atributos que não sejam utilizados no GlarAssist (como por exemplo, a seção do colaborador, a idade, entre outros), são guardados numa estrutura à parte, uma vez que, poderão ser importantes no futuro.

Para além dos atributos do SAML, as organizações podem logo definir as *roles* dos utilizadores pelo SSO. Para isso, os utilizadores podem criar grupos de *roles*. Os grupos são representados pelo nome do grupo enviado na resposta do SAML e a *role* que o mesmo representa na aplicação do GlarAssist.

Esta informação pode ser gerida no pedido *EntityUpdate*, apresentado na Tabela 9. Foram também criados pedidos na API REST de forma a permitir que as aplicações *frontend* e as plataformas de SSO comuniquem, para a realização do SSO (ver Tabela 9). No pedido *GetSamlUrl* é enviado como parâmetro um *id*. O *id* enviado é validado no servidor e verificado se está associado a alguma entidade. Se for identificada uma entidade que tenha a configuração do SAML definida, é retornado o *url* para realizar o *login* no respetivo *service provider*. Caso a autenticação seja feita com sucesso na plataforma das organizações, o serviço faz o pedido *SamlCallback*. Nesse pedido o servidor lida com todos os dados recebidos. Caso a conta não exista é criada uma conta automaticamente. Se a conta for criada

pelo SSO, o utilizador apenas poderá utilizar o SSO para se autenticar, uma vez que, o *login* por email e *passsword* fica desativado. Tal como, a possibilidade de alterar a *password* fica, também, desativada.

Nome/Prop.	API	Autenticado	Rota/Evento	*Payload*	Resultado esperado
GetSamlUrl	REST		/saml/<id>	-	Informação sobre *service provider*
SamlCallback	REST		/saml/callback/<id>	SAML Response	Login realizado com sucesso

Tabela 9 - Alterações da API na funcionalidade autenticação centralizada

Se os atributos recebidos no pedido forem diferentes dos atributos do utilizador na plataforma, os mesmos são atualizados automaticamente, dando sempre prioridade aos valores enviados dos *service providers*. Caso a *role* do utilizador seja enviada na resposta do SAML, é também atualizada a conta do utilizador do GlarAssist. O resultado esperado do pedido *SamlCallback* é redirecionar o utilizador para as aplicações *frontend*, com o *token* da sessão.

Para integrar esta funcionalidade na aplicação *web*, foram atualizados os componentes do *login* e adicionada uma nova rota para os utilizadores configurarem o SAML. As UI desenvolvidas e alteradas são apresentadas nas Figuras 36-38. No componente do *login* foi adicionado um novo botão para realizar a autenticação pelo SSO (ver Figura 36).

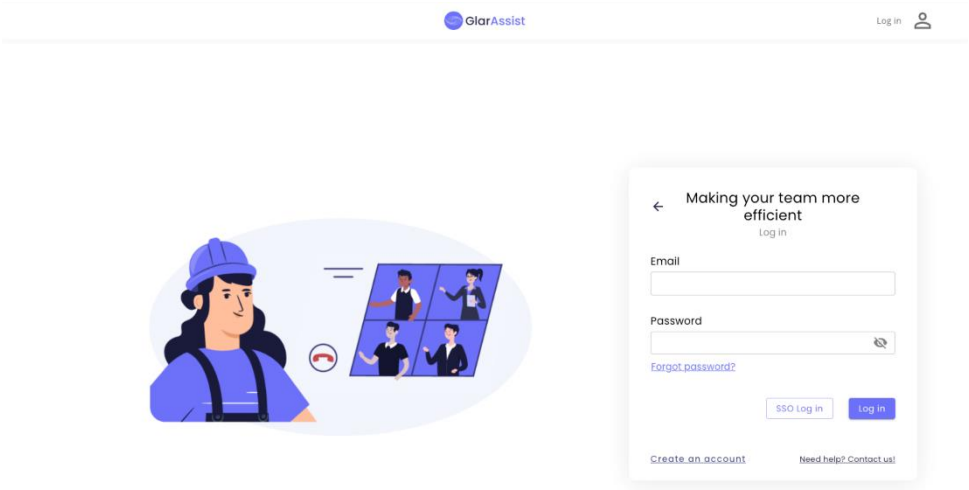


Figura 36 - Componente do Login com o SSO

Ao clicar no mesmo, o utilizador é redirecionado para o formulário para realizar a autenticação. A identificação da entidade pode ser realizada pelo email do utilizador ou pelo nome da mesma. Caso a entidade seja identificada o utilizador é redirecionado para a plataforma da entidade.

A nova página possibilita aos administradores das entidades configurarem o SAML (ver Figura 37). Se o utilizador for administrador, é exibido um novo item no menu do componente das organizações para o utilizador configurar o SAML.

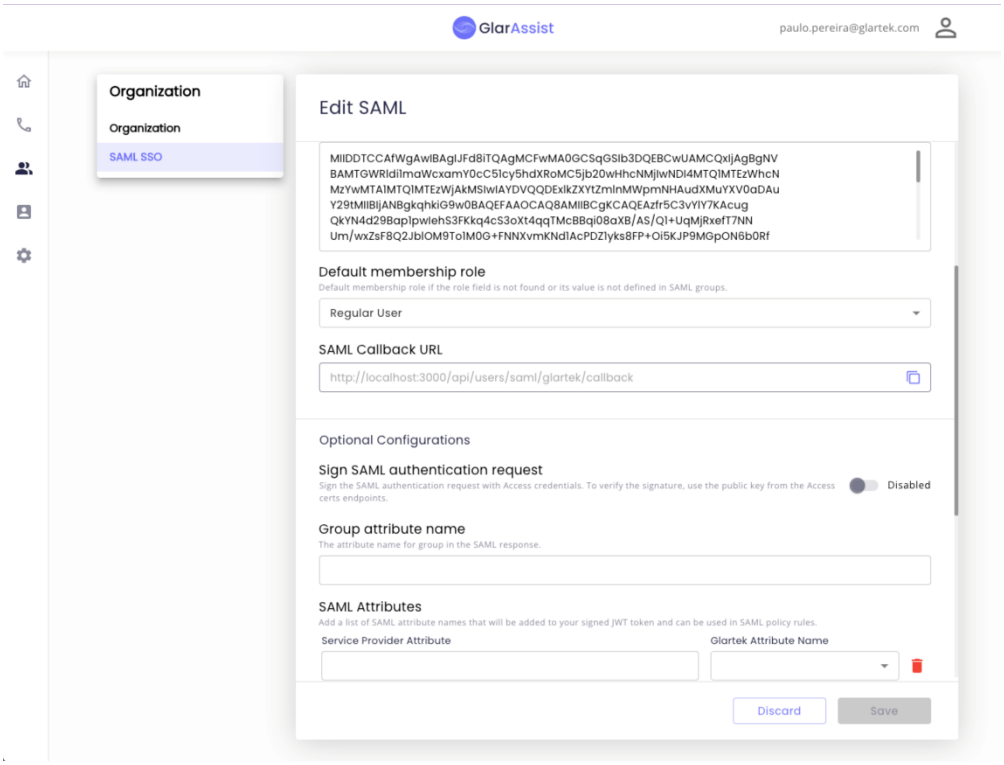


Figura 37 - Componente para gerir a configuração do SAML

Neste formulário o utilizador pode atualizar a configuração da sua entidade. O formulário é composto por diferentes *inputs*, sendo eles: texto, que pode ser multilinha para atributos geralmente grandes; “mapa”, que é representado por linhas de *inputs* de texto ou de seleção, em que cada “key” (*input* da esquerda) tem um “value” (*input* da direita); seleção, que é utilizado para a seleção de *roles*; e a tabela, onde são apresentados os grupos definidos na configuração. Os *inputs* alteram o respetivo atributo, tendo todos uma pequena descrição para auxiliar os administradores.

O *input* “SAML Attributes” tem como “value”, apenas os valores que podem ser armazenados na estrutura de utilizadores do GlarAssist, apresentado na Figura 38. Como nos formulários anteriormente desenvolvidos, caso o utilizador mude de rota com informação por guardar é exibido o Dialog (ver Figura 28).

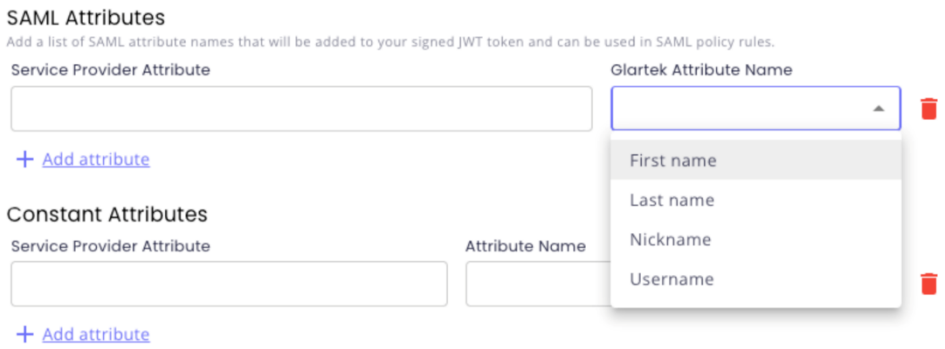


Figura 38 - Input dos atributos de utilizador do SAML

Como referido anteriormente, as contas que sejam criadas pelo SSO não poderão definir a sua *password*. Por consequência, o item do menu secundário das definições do utilizador, para atualizar a *password* é removido (identificado como “Change Password” na Figura 29).

4.12. Resultado do desenvolvimento

As funcionalidades requeridas foram todas desenvolvidas e implementadas no ambiente produtivo com sucesso. A junção destas funcionalidades concedeu ao GlarAssist novas formas de complementar as assistências remotas e diferentes ferramentas para os utilizadores da aplicação utilizarem no seu dia-a-dia. Com a criação do histórico e dos filtros avançados, toda a informação da assistência é armazenada e permite aos assistidos e assistentes partilharem e visualizarem as ações realizadas durante as assistências remotas. Para além do histórico, foi integrada a funcionalidade do *chat* com partilha de ficheiros, na assistência, que permite aos utilizadores comunicarem em diferentes ambientes e partilharem informações relevantes para a assistência. A introdução das organizações é a primeira fase da funcionalidade, uma vez que, a mesma tem uma grande escalabilidade. Atualmente, possibilita gerir os recursos de uma organização e integrar plataformas de SSO das organizações. A possibilidade de os utilizadores terem uma lista de contactos, que pode ser utilizada para convidar utilizadores para assistências, é importante para melhorar a

experiência dos utilizadores durante as assistências, devido à facilidade de entrar em contacto com outros utilizadores. A possibilidade de alterar o volume dos dispositivos dos assistidos, também, é relevante nas abordagens de aplicabilidades “mãos-livres”. E por fim as *feature flags*, que apesar de não terem um impacto direto na experiência dos utilizadores, permite que aplicações desatualizadas continuem a funcionar com aplicações mais recentes sem comportamentos inesperados.

No entanto, cada funcionalidade teve os seus desafios, durante as fases de desenvolvimento, testes e integração com o ambiente produtivo. Esses desafios são abordados no capítulo seguinte.

5. Testes e resultados

Antes de uma funcionalidade ser integrada no ambiente produtivo, teve de passar por diversas fases de testes, para evitar que aconteçam comportamentos inesperados em produção. Essas fases vão desde o desenvolvimento, a testes do código desenvolvido, a testes de usabilidade e, por fim, é testada a junção noutro ambiente (geralmente, ambiente com mais dados). Estas fases, permitem despistar possíveis problemas. Para a realização de alguns testes, foi introduzido um elemento na equipa de desenvolvimento, o Tester. O objetivo do Tester é realizar diferentes testes de usabilidade nas funcionalidades, definidos pelo mesmo, para confirmar se a implementação tem o resultado esperado. Como os testes são definidos pelo Tester, não existe um registo dos testes executados. Contudo, quando o Tester identifica algum problema o mesmo é reportado na plataforma do Gitlab. Assim, toda a equipa de desenvolvimento tem acesso aos problemas encontrados e, desta forma, os mesmos são resolvidos antes de ser concluída a funcionalidade. Para além dos testes realizados pelo Tester, existem passos na *pipeline* onde o código é validado. Para isso é utilizado o *lint* (ferramenta para verificar a integridade do código) com diversos parâmetros definidos pela equipa de desenvolvimento. O *lint* é executado em todas as aplicações do GlarAssist, em cada instância da pipeline do ambiente de desenvolvimento.

Foram desenvolvidas diferentes funcionalidades, logo foram alteradas diferentes operações e lógicas da aplicação. Neste sentido, os testes e os resultados diferenciam consoante a funcionalidade. Uma vez que, não existe uma lista ou formulário com os testes realizados pelo Tester, foram recolhidos todos os problemas registados na plataforma Gitlab, em relação às funcionalidades desenvolvidas. Estes registos podem ser referentes às três aplicações do GlarAssist. Uma vez que, apenas são abordadas duas aplicações do GlarAssist, apenas foram levantados problemas referentes a essas duas aplicações. Cada registo foi identificado com um nível de gravidade ou importância: baixo, médio ou alto. Sendo que, o nível baixo refere-se a problemas que não afetam o funcionamento da *feature*, nem a experiência do utilizador. O nível médio poderá afetar a *feature* ou a UI, mas não afeta o funcionamento esperado da aplicação. O nível alto representa problemas que criam comportamentos inesperados na aplicação, informação sensível é exposta, quebra da *performance* da aplicação, entre outros. Desta forma, os problemas detetados de nível baixo foram:

- Alertas no código – na *pipeline* existem testes para validar possíveis problemas no código, sendo eles: indentação não era respeitada, variáveis definidas não estavam a ser utilizadas, importações de componentes ou bibliotecas que não estavam a ser utilizados, variáveis que não tinham um tipo definido, não eram lidados possíveis erros, entre outros. Esta lista de problemas era exposta quando a pipeline executava no ambiente de desenvolvimento. Quando eram reportados problemas, tanto no servidor como na aplicação *web*, com o auxílio de *scripts* e resolução dos problemas no código, estes alertas eram facilmente resolvidos. Este problema aconteceu na maioria das funcionalidades, depois de serem implementadas, mas foram facilmente resolvidos.
- Estruturas desatualizadas – Quando existe a constante alteração de conceitos numa aplicação, é difícil manter todas as estruturas com os campos realmente necessários. Por vezes, ao atualizar uma nova estrutura poderão não ser removidos todos os campos que deixavam de ser utilizados, criando assim estruturas desatualizadas com campos obsoletos. Este problema pode acontecer por erros de desenvolvimento dos *developers*. Porém, geralmente, estas falhas geram apenas informação que nunca vai ser utilizada, e por consequência, não afeta o funcionamento das aplicações. Contudo, os mesmos têm de ser resolvidos para garantir integridade no código das aplicações. Estes erros são detetados, normalmente, por outros *developers* quando está a ser revisto o código da funcionalidade. Este problema ocorreu na funcionalidade dos históricos das assistências e na funcionalidade das organizações. Depois de identificados, os problemas foram corrigidos.

Os problemas detetados de nível médio no desenvolvimento das aplicações foram:

- Manutenção das *feature flags* – A funcionalidade das *feature flags* é importante para garantir o funcionamento expectável em aplicações de diferentes versões, desativando apenas as funcionalidades que não são suportadas entre as mesmas. Contudo, a lista de funcionalidades das *feature flags* tem de ser mantida, sempre que é desenvolvida uma nova operação que poderá criar *breaking changes*. Na funcionalidade da lista de contactos dos utilizadores a lista das *features* não foi atualizada com as novas funcionalidades. Assim, nas *feature flags* não estava a ser considerada a nova funcionalidade. Depois de ser detetado este problema foi criado uma *branch*, onde foi resolvida esta questão, em todas as aplicações (*Bug fixing*). As *features flags*, neste momento, não têm impacto nenhum nas funcionalidades referentes à *feature* da lista

de contactos, contudo, a mesma tem de ser definida para futuros desenvolvimentos que causem *breaking changes*. Ainda assim, este problema não teve qualquer impacto nas aplicações.

- Problemas nas *user interfaces* – O impacto deste problema é as *user interfaces* desenvolvidas terem algumas falhas em relação aos *mockups* definidos pela equipa do produto. Estas falhas, normalmente, estão relacionadas com parâmetros de CSS (Cascading Style Sheets), isto é, algum texto não tem o tipo de fonte ou tamanho certos, o espaçamento entre elementos não está correto, as cores dos elementos ou do texto não são as indicadas, entre outras. Estes problemas ocorreram nas funcionalidades dos históricos das assistências de utilizador e organização, da organização e da autenticação centralizada. Contudo, estes problemas foram detetados pelo Tester, ainda no ambiente de desenvolvimento. Caso fossem detetados já em produção, os utilizadores conseguiam utilizar todas as funcionalidades sem qualquer restrição ou problema.
- Dispositivos móveis – Este problema está relacionado também com a UI, mas em dispositivos com ecrãs mais pequenos, geralmente, dispositivos móveis. Apesar da aplicação *web* estar direcionada para computadores pessoais ou *tablets* (ecrãs maiores), a mesma pode ser utilizada em dispositivos móveis. A equipa de produto, em particular, a equipa de UX/UI é composta por apenas dois elementos. Uma vez que, a equipa tem de desenvolver as UI do GlarAssist e do GlarVision, não tem disponibilidade para implementar o comportamento das *interfaces* nos diferentes tamanhos de ecrãs. Por esse motivo, foi decidido entre as equipas de produto e de desenvolvimento que os *developers* que estivessem a desenvolver as funcionalidades das aplicações *web*, deviam garantir que a mesma é utilizável em dispositivos móveis. Assim, o Tester também testa o comportamento das *interfaces* em diferentes ecrãs. Na funcionalidade da organização, os elementos da UI, em ecrãs pequenos, não eram responsivos. Por este motivo, teve de ser resolvida esta questão para possibilitar a utilização desta funcionalidade em dispositivos móveis. O problema foi detetado em ambiente de desenvolvimento.

Foi apenas detetado um problema de nível alto. Este problema está relacionado com a *query* que é utilizada para retornar o histórico das assistências dos utilizadores e das organizações. Para devolver a informação da tabela e permitir utilizar os diversos filtros, é agrupada a informação de diferentes *collections*, na mesma *query*. Para juntar a informação

toda são realizadas diversas operações. Uma vez que, as bases de dados não relacionais, não são as mais apropriadas para estas operações de junção, o desempenho na realização deste tipo de operações é muito afetado. O ambiente utilizado durante a implementação desta *query* foi o local (ambiente de uma máquina). Por consequência, a *performance* neste ambiente é muito melhor, uma vez que, não existe tanto *delay* nos pedidos e não há tantos dados em comparação ao ambiente de produção. Para além do ambiente local, no ambiente de desenvolvimento, não foi detetado pelo Tester nenhum problema, uma vez que, a quantidade de informação e carga de utilização desse servidor e base de dados é muito reduzida. Quando a funcionalidade foi adicionada ao ambiente produtivo, foi detetada uma quebra de *performance* muito grande. Na utilização da *query*, o sistema de monitorização da base de dados estava a detetar a mesma como uma pesquisa com elevado custo computacional. Quando o problema foi detetado, a sua resolução passou a ser de prioridade máxima, visto que, estava a gerar comportamentos inesperados e quebras na *performance* do servidor, na base de dados e nas aplicações do *frontend*. Para resolver o problema, foram testadas várias variantes da *query*, com diferentes operações, entre outras abordagens. Durante os testes, foi identificada uma operação que estava a causar as quebras de *performance*. Depois de ser identificada, foi desenvolvida outra abordagem que devolvia o mesmo resultado, sem executar a operação. Na conclusão, a resolução do problema foi exposta a diversos testes pelo *developer* e pelo resto da equipa, ainda no ambiente de desenvolvimento. Os testes tiveram todos sucesso, sem qualquer perda de *performance*. Quando a resolução foi integrada no ambiente produtivo, o problema nunca mais foi detetado e, assim, foi considerado como resolvido. Esta situação revela que, as bases de dados não relacionais, quando expostas a *queries* mais complexas pode ter um grande impacto na *performance* da mesma.

No fim de todo o processo de desenvolvimento das funcionalidades, é possível afirmar que todas tiveram o resultado esperado. A maioria das funcionalidades teve o mesmo peso, em relação ao esforço necessário para a sua implementação. As funcionalidades dos históricos e da organização foram as que necessitaram de mais tempo para serem desenvolvidas, devido às alterações que tiveram de ser realizadas e às UI que tiveram de ser desenvolvidas. Para além destas funcionalidades, a UI da lista de contactos demorou mais tempo do que o esperado, devido ao componente de seleção dos contactos (ver Figura 35). Para o mesmo ser implementado, como nos *mockups* definidos pela equipa de produto, o componente do MUI teve de ser alterado. Isto é, todas as partes que compõem o componente

tiveram de ser modificadas para a informação ser apresentada da forma requerida. Para além do desenvolvimento da *interface*, este componente faz diversos pedidos ao servidor para retornar os contactos que combinam com a *string* introduzida. Este processo de obtenção dos contactos, também, teve de ser corretamente tratado, para não serem realizados pedidos desnecessários.

Em síntese, a maioria dos problemas detetados durante o desenvolvimento das funcionalidades foi no ambiente de desenvolvimento. Apenas um foi detetado no ambiente produtivo, mas foi rapidamente resolvido. Neste sentido, as funcionalidades foram desenvolvidas como esperado, proporcionando assim, ao GlarAssist, novas ferramentas para os seus utilizadores explorarem.

6. Conclusão

O presente relatório apresenta a iniciação do BackOffice da aplicação GlarAssist. Durante este processo de iniciação foram desenvolvidas onze funcionalidades, para possibilitar a gestão das informações partilhadas durante as assistências remotas. Para além da gestão das assistências, também foram desenvolvidas funcionalidades para que as organizações consigam integrar, criar e gerir as suas equipas.

Na fase inicial, foi realizada uma contextualização das assistências remotas com RA, onde foram identificadas as vantagens da utilização de assistências remotas com RA. Para além das suas vantagens, foram abordados padrões de melhores práticas, a ter em consideração, no desenvolvimento de aplicações de assistência remota de RA. Os mesmos foram importantes para compreender a importância de cada funcionalidade desenvolvida e o seu impacto no GlarAssist.

Também foram abordados os resultados obtidos nas diferentes investigações efetuadas. É possível afirmar que as assistências remotas com RA melhoraram as diferentes operações, nos diversos contextos, onde foram integradas. Foi também identificada como a principal adversidade, as aplicações e os dispositivos utilizados não estarem preparados para os diferentes contextos ambientais, onde as assistências são realizadas (pouca iluminação, muito ruido, pouca cobertura de Internet, entre outros).

De seguida, foi realizada uma análise competitiva de outras aplicações de assistência remota de RA, em comparação ao GlarAssist e às funcionalidades desenvolvidas, com as funcionalidades das aplicações identificadas.

Na fase seguinte, foi apresentado todo o processo de desenvolvimento, onde foram abordados os requisitos definidos, a metodologia de desenvolvimento utilizada durante a implementação das funcionalidades, a arquitetura do GlarAssist e as tecnologias e serviços utilizados na aplicação.

Posteriormente, foram apresentadas as funcionalidades desenvolvidas, onde foi abordado todo o processo de desenvolvimento das mesmas. Por fim, foram apresentados os testes realizados nas funcionalidades implementadas, os problemas identificados durante os testes, a resolução desenvolvida para cada problema e os resultados dos testes.

Em síntese, as funcionalidades requeridas foram todas desenvolvidas e integradas com sucesso no GlarAssist, cumprindo assim todos os requisitos definidos. O desenvolvimento

realizado permite, futuramente, que estas funcionalidades sejam expandidas para integrarem novos conceitos e aplicabilidades.

A aplicação está a ser direcionada para ambientes empresariais, onde é gerida muita informação das organizações e as funcionalidades requerem gradualmente mais dados. Por esse motivo, um problema ou limitação identificado foi a utilização de uma base de dados não relacional. A BD poderá tornar-se numa adversidade, uma vez que, as pesquisas efetuadas na mesma, irão se tornar cada vez maiores e retornarão cada vez mais dados. A transição para uma base de dados relacional, que proporciona melhor desempenho: na associação de registos diferentes; em contextos com uma grande quantidade de dados; e na execução de consultas complexas, poderá ser necessário para não limitar implementações futuras.

As atuais funcionalidades do GlarAssist permitem aos seus utilizadores realizarem assistências remotas de RA, com diversas funcionalidades, para diferentes contextos. Porém, a análise realizada em relação a este tipo de assistências, identificou outras questões, que poderão ser importantes na definição de futuras funcionalidades na aplicação. No entanto, o GlarAssist é uma ferramenta cada vez mais completa, que possibilita as organizações utilizarem-na nas diferentes operações do dia-a-dia.

As assistências remotas com o uso de realidade aumentada demonstraram como a RA pode melhorar uma assistência remota, proporcionando novas formas de comunicar e assistir utilizadores. Ao mesmo tempo, permite que organizações reduzam os seus custos e melhorem a sua eficiência nas suas operações diárias.

Bibliografia

[1] L. F. de Souza Cardoso, F. C. M. Q. Mariano, and E. R. Zorzal, "A survey of industrial augmented reality," *Comput Ind Eng*, vol. 139, p. 106159, Jan. 2020, doi: 10.1016/J.CIE.2019.106159.

[2] D. Röltgen and R. Dumitrescu, "Classification of industrial Augmented Reality use cases," in *Procedia CIRP*, 2020, vol. 91, pp. 93–100. doi: 10.1016/j.procir.2020.01.137.

[3] R. E. Petruse, A. Matei, M. Kayser, M. Maier, and S. Cuković, "Academia-industry Collaboration for Augmented Reality Application Development," in *Balkan Region Conference on Engineering and Business Education*, 2019, vol. 3, no. 1, pp. 243–250. doi: 10.2478/cplbu-2020-0028.

[4] D. Fang, H. Xu, X. Yang, and M. Bian, "An Augmented Reality-Based Method for Remote Collaborative Real-Time Assistance: from a System Perspective," *Mobile Networks and Applications*, vol. 25, no. 2, pp. 412–425, Apr. 2020, doi: 10.1007/s11036-019-01244-4.

[5] A. Gallala, B. Hichri, and P. Plapper, "Survey: The Evolution of the Usage of Augmented Reality in Industry 4.0," in *IOP Conference Series: Materials Science and Engineering*, 2019, vol. 521, no. 1. doi: 10.1088/1757-899X/521/1/012017.

[6] F. Obermair *et al.*, "Maintenance with Augmented Reality Remote Support in Comparison to Paper-Based Instructions: Experiment and Analysis," *2020 IEEE 7th International Conference on Industrial Engineering and Applications, ICIEA 2020*, pp. 942–947, Apr. 2020, doi: 10.1109/ICIEA49774.2020.9102078.

[7] P. Z. Lodetti, A. B. dos Santos, L. T. Hattori, E. G. Carvalho, and M. A. I. Martins, "Mobile Remote Assistance with Augmented Reality Applied in a Power Distribution Utility: A Qualitative Study," Jun. 2022, pp. 1–6. doi: 10.1109/ietc54973.2022.9796952.

[8] D. Calandra, A. Cannavò, and F. Lamberti, "Improving AR-powered remote assistance: a new approach aimed to foster operator's autonomy and optimize the use of skilled resources," *International Journal of Advanced Manufacturing Technology*, vol. 114, no. 9–10, pp. 3147–3164, Jun. 2021, doi: 10.1007/s00170-021-06871-4.

[9] R. T. Azuma, “A Survey of Augmented Reality,” *Presence: Teleoperators and Virtual Environments*, vol. 6, no. 4, pp. 355–385, Aug. 1997, doi: 10.1162/pres.1997.6.4.355.

[10] Z. Alavikia and M. Shabro, “Pragmatic Industrial Augmented Reality in Electric Power Industry,” in *34th International Power System Conference, PSC 2019*, Dec. 2019, pp. 25–32. doi: 10.1109/PSC49016.2019.9081538.

[11] J. Wolfartsberger, J. Zenisek, and N. Wild, “Data-driven maintenance: Combining predictive maintenance and mixed reality-supported remote assistance,” in *Procedia Manufacturing*, 2020, vol. 45, pp. 307–312. doi: 10.1016/j.promfg.2020.04.022.

[12] D. G. M. Vargas, K. K. Vijayan, and O. J. Mork, “Augmented reality for future research opportunities and challenges in the shipbuilding industry: A literature review,” in *Procedia Manufacturing*, 2020, vol. 45, pp. 497–503. doi: 10.1016/j.promfg.2020.04.063.

[13] D. Mourtzis, V. Siatras, and J. Angelopoulos, “Real-time remote maintenance support based on augmented reality (AR),” *Applied Sciences (Switzerland)*, vol. 10, no. 5, Mar. 2020, doi: 10.3390/app10051855.

[14] B. Marques, S. Silva, A. Rocha, P. Dias, and B. S. Santos, “Remote asynchronous collaboration in maintenance scenarios using augmented reality and annotations,” in *Proceedings - 2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops, VRW 2021*, Mar. 2021, pp. 567–568. doi: 10.1109/VRW52623.2021.00166.

[15] F. de Pace, F. Manuri, and A. Sanna, “Augmented Reality in Industry 4.0,” *American Journal of Computer Science and Information Technology*, vol. 06, no. 01, 2018, doi: 10.21767/2349-3917.100017.

[16] M. Xi, M. Adcock, and J. McCulloch, “Future agriculture farm management using augmented reality,” *2018 IEEE Workshop on Augmented and Virtual Realities for Good, VAR4Good 2018*, Dec. 2018, doi: 10.1109/VAR4GOOD.2018.8576887.

[17] D. Aslan, B. B. Çetin, and İ. G. Özbilgin, “An Innovative Technology: Augmented Reality Based Information Systems,” in *Procedia Computer Science*, 2019, vol. 158, pp. 407–414. doi: 10.1016/j.procs.2019.09.069.

[18] S. T.V, L. Ragha, R. Desai, and A. Vaishya, “Innovative Interaction Software for Remote Assistance Application,” *SSRN Electronic Journal*, 2022, doi: 10.2139/SSRN.4159662.

[19] M. Rebol *et al.*, “Remote assistance with mixed reality for procedural tasks,” in *Proceedings - 2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops, VRW 2021*, Mar. 2021, pp. 653–654. doi: 10.1109/VRW52623.2021.00209.

[20] A. Jakl, L. Schöffer, M. Husinsky, and M. Wagner, “Augmented Reality for Industry 4.0: Architecture and User Experience,” 2018.

[21] M. Rice *et al.*, “Evaluating an augmented remote assistance platform to support industrial applications,” in *IEEE World Forum on Internet of Things, WF-IoT 2018 - Proceedings*, May 2018, vol. 2018-January, pp. 592–597. doi: 10.1109/WF-IoT.2018.8355133.

[22] A. Aschauer, I. Reisner-Kollmann, and J. Wolfartsberger, “Creating an Open-Source Augmented Reality Remote Support Tool for Industry: Challenges and Learnings,” in *Procedia Computer Science*, 2021, vol. 180, pp. 269–279. doi: 10.1016/j.procs.2021.01.164.

[23] W. Vorraber, J. Gasser, H. Webb, D. Neubacher, and P. Url, “Assessing augmented reality in production: Remote-assisted maintenance with HoloLens,” in *Procedia CIRP*, 2020, vol. 88, pp. 139–144. doi: 10.1016/j.procir.2020.05.025.

[24] P. Mohr, S. Mori, T. Langlotz, B. H. Thomas, D. Schmalstieg, and D. Kalkofen, “Mixed Reality Light Fields for Interactive Remote Assistance,” Apr. 2020. doi: 10.1145/3313831.3376289.

[25] P. Hohl *et al.*, “Back to the future: origins and directions of the ‘Agile Manifesto’ – views of the originators,” *Journal of Software Engineering Research and Development*, vol. 6, no. 1, p. 15, 2018, doi: 10.1186/s40411-018-0059-z.

[26] S. Al-Saqqa, S. Sawalha, and H. Abdelnabi, “Agile Software Development: Methodologies and Trends,” *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 14, no. 11, pp. 246–270, Jul. 2020, doi: 10.3991/IJIM.V14I11.13269.

[27] M. Morandini, T. A. Coleti, E. Oliveira, and P. L. P. Corrêa, “Considerations about the efficiency and sufficiency of the utilization of the Scrum methodology: A survey for analyzing results for development teams,” *Comput Sci Rev*, vol. 39, p. 100314, Feb. 2021, doi: 10.1016/J.COSREV.2020.100314.

[28] H. Alaidaros, M. Omar, and R. Romli, “The state of the art of agile kanban method: challenges and opportunities,” *Independent Journal of Management & Production*, vol. 12, no. 8, pp. 2535–2550, Dec. 2021, doi: 10.14807/IJMP.V12I8.1482.

[29] “Git.” https://git-scm.com/ (accessed Aug. 02, 2022).

[30] M. Vuorre and J. P. Curley, “Curating Research Assets: A Tutorial on the Git Version Control System:,” *https://doi.org/10.1177/2515245918754826*, vol. 1, no. 2, pp. 219–236, Apr. 2018, doi: 10.1177/2515245918754826.

[31] “The One DevOps Platform | GitLab.” https://about.gitlab.com/ (accessed Aug. 02, 2022).

[32] “Documentation - The Go Programming Language.” https://go.dev/doc/ (accessed Aug. 02, 2022).

[33] “Why Go - The Go Programming Language.” https://go.dev/solutions/#use-cases (accessed Aug. 02, 2022).

[34] “What is a REST API?” https://www.redhat.com/en/topics/api/what-is-a-rest-api (accessed Aug. 02, 2022).

[35] “What is an Application Programming Interface (API) | IBM.” https://www.ibm.com/cloud/learn/api (accessed Aug. 02, 2022).

[36] “WebSocket - IBM Documentation.” https://www.ibm.com/docs/en/was-liberty/base?topic=liberty-websocket (accessed Aug. 02, 2022).

[37] “What Is MongoDB? | MongoDB.” https://www.mongodb.com/en/what-is-mongodb (accessed Aug. 02, 2022).

[38] “SQL vs. NoSQL Databases: What’s the Difference? | IBM.” https://www.ibm.com/cloud/blog/sql-vs-nosql (accessed Aug. 02, 2022).

[39] “Agora Real-Time Voice and Video Engagement.” https://www.agora.io/en/ (accessed Aug. 02, 2022).

[40] “Cloud Computing Services - Amazon Web Services (AWS).” https://aws.amazon.com/ (accessed Aug. 02, 2022).

[41] “Cloud Object Storage – Amazon S3 – Amazon Web Services.” https://aws.amazon.com/s3/ (accessed Aug. 02, 2022).

[42] “What is Security Assertion Markup Language (SAML)? | Oracle.” https://www.oracle.com/security/cloud-security/what-is-saml/ (accessed Aug. 02, 2022).

[43] “SAML Overview: How SAML Authentication Works | OneLogin Developers.” https://developers.onelogin.com/saml (accessed Aug. 02, 2022).

[44] “React – A JavaScript library for building user interfaces.” https://reactjs.org/ (accessed Aug. 02, 2022).

[45] “TypeScript: JavaScript With Syntax For Types.” https://www.typescriptlang.org/ (accessed Aug. 02, 2022).

[46] “Material UI - Overview - Material UI.” https://mui.com/pt/material-ui/getting-started/overview/ (accessed Aug. 02, 2022).

[47] “XRmeet: AR Remote Assistance & AR Object Detection Software.” https://www.xrmeet.io/ (accessed Aug. 05, 2022).

[48] “TeamViewer Assist AR - Bring Augmented Reality Support to a New Level.” https://www.teamviewer.com/en/augmented-reality/ (accessed Aug. 05, 2022).

[49] “Remote Assist | Microsoft Dynamics 365.” https://dynamics.microsoft.com/en-us/mixed-reality/remote-assist/ (accessed Aug. 05, 2022).

[50] “Enterprise Augmented Reality Remote Assistance & Support Solutions.” https://www.scopear.com/ (accessed Aug. 05, 2022).

[51] “Vuforia Chalk Augmented Reality (AR) Remote Assistance | PTC.” https://www.ptc.com/en/products/vuforia/vuforia-chalk (accessed Aug. 05, 2022).

[52] “REFLEKT Remote Augmented Reality Tool for Service & Support.” https://www.re-flekt.com/reflekt-remote (accessed Aug. 05, 2022).

[53] “AR Enabled Remote Assistance | Atheer.” https://www.atheerair.com/remote-assist/ (accessed Aug. 05, 2022).

[54] “Remote Access Software | Free Remote Support Software - Zoho Assist.” https://www.zoho.com/assist/ (accessed Aug. 05, 2022).

[55] “Augmented Reality software for remote support – Acty.” https://www.acty.com/ (accessed Aug. 05, 2022).

[56] “JoinPad - Augmented Reality for Enterprises.” https://www.joinpad.net/ (accessed Aug. 05, 2022).

[57] “VSight Remote | VSight.” https://vsight.io/vsight-remote/ (accessed Aug. 05, 2022).

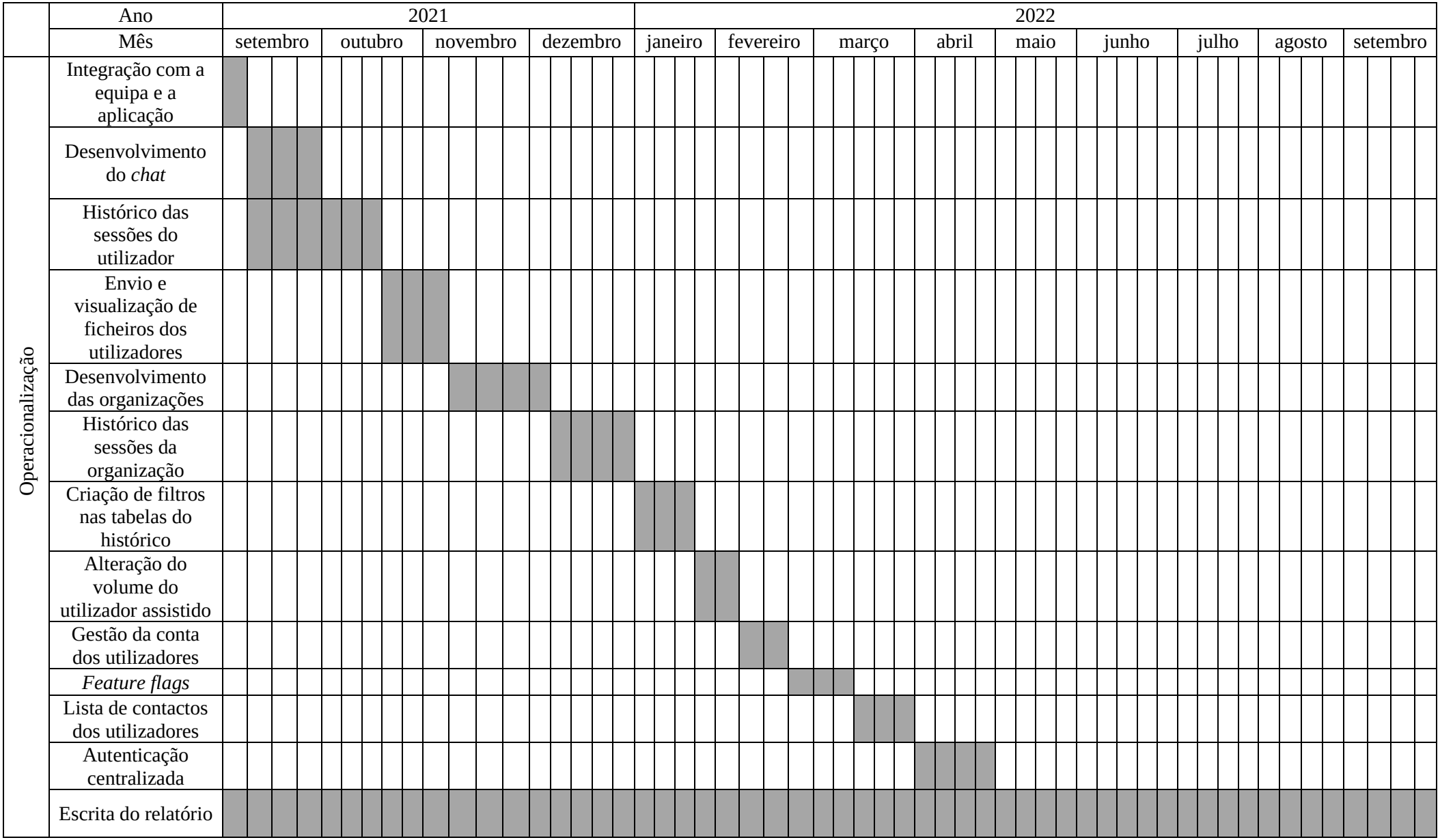
[58] “Houston - Workforce Knowledge Software | Resco.net.” https://www.resco.net/houston/ (accessed Aug. 05, 2022).

[59] “AR Remote Support (KoalaAR) - Arimars Technologies.” https://www.arimars.com/augmented-reality/augmented-remote-support-koala/ (accessed Aug. 05, 2022).

Esta página foi intencionalmente deixada em branco

Apêndices

Aplicações / Features	Aplicação web	Aplicação desktop	Assistência com Multiutilizadores	Autenticação centralizada	*Chat*	Transferência e partilha de ficheiros	Histórico	Filtros	Gestão de equipas/ organizações	Lista de contactos
XRmeet [47]	X		X		X		X	X		X
TeamViewer - Assist AR [48]	X	X	X	X		X		-	X	X
Dynamics 365 - Remote Assist [49]	X	X	X	X	X	X	X	X	X	X
Scope AR [50]		X				X		-		X
Vuforia Chalk [51]	X		X	X				-	X	X
Re'flekt Remote [52]		X	X		X	X		-	X	X
Atheer – Remote Assist [53]	X	X	X		X	X		-		
Zoho – Assist [54]	X	X		X	X	X	X	X	X	X
Acty [55]	X		X	X				-	X	
Joinpad [56]	X		X			X	X		X	X
Vsight Remote [57]	X		X	X	X	X	X		X	
resco.Houston [58]	X					X	X	X	X	X
KoalaAR [59]			X					-		


Ano
2021
2022
Mês
setembro
outubro
novembro
dezembro
janeiro
fevereiro
março
abril
maio
junho
julho
agosto
setembro
Operacionalização
Integração com a equipa e a aplicação
Desenvolvimento do *chat*
Histórico das sessões do utilizador
Envio e visualização de ficheiros dos utilizadores
Desenvolvimento das organizações
Histórico das sessões da organização
Criação de filtros nas tabelas do histórico
Alteração do volume do utilizador assistido
Gestão da conta dos utilizadores
Feature flags
Lista de contactos dos utilizadores
Autenticação centralizada
Escrita do relatório

Esta página foi intencionalmente deixada em branco