INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Desenvolvimento de solução de Telemetria (Status Monitor)

DIOGO FILIPE FREITAS RIBEIRO
Junho de 2023

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Desenvolvimento de solução de Telemetria (Status Monitor)

Diogo Filipe Freitas Ribeiro

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Computacionais

Orientador: Dr. Nuno Silva
Supervisor: João Areias

Porto, 30 de junho de 2023

Declaração de Integridade

Declaro ter conduzido este trabalho académico com integridade.

Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.

Portanto, o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.PORTO.

ISEP, Porto, 30 de junho de 2023

I don't know which option you should choose and I could never advise you on that. No matter what kind of wisdom dictates you the option you pick, no one will be able to tell if it's right or wrong until you arrive to some sort of outcome from your choice.

LEVI ACKERMAN

Resumo

Para qualquer tipo de sistema que exista, ter acesso a diversas informações referentes ao mesmo é sempre benéfico, resultando numa visão e perceção geral do estado do sistema em questão. Isso torna possível, por exemplo, a identificação de erros produzidos pelo mesmo, o que pode ajudar nas suas retificações, evitando possíveis ressurgimentos futuros, e contribuindo, consequentemente, para a qualidade do produto. Para além disso, também é possível capturar tempos de processamento, tendências de desempenho ao longo de um período de tempo, entre outros. Esses tipos de dados podem ser considerados dados de telemetria.

Os dados podem ser apresentados a utilizadores interessados através de sistemas específicos para tal, por forma a ajudá-los a tomar decisões relativas ao sistema monitorizado, consoante esses mesmos dados.

Esta tese apresenta uma plataforma (Status Monitor), já existente, que capta dados de uma outra, mas que possui um baixo grau de observabilidade de dados. Por isso, são propostas algumas funcionalidades com o objetivo de tentar mitigar o problema supracitado, agregando ainda mais valor ao produto existente. Essas adições permitem que os utilizadores que tenham acesso às mesmas possam assumir ações que, consequentemente, podem melhorar a qualidade da aplicação monitorizada.

De todas as funcionalidades propostas, foram adicionadas as que envolvem a criação de *dashboards* que apresentam diversas informações e métricas relativamente a servidores, instalações, e componentes do sistema.

Palavras-chave: Telemetria, Observabilidade, Métricas, Monitorização

Abstract

For any type of system that exists, it's always beneficial to know a lot of information about it, providing an overview and general perception of the state of the system in question. This makes possible, for example, the identification of errors produced by the same, which can help in its rectifications, avoiding possible future resurgences, and contributing, consequently, to the quality of the product. In addiction, it's also possible to capture processing times, performance trends over a period of time, and much more. These types of data can be considered telemetry data.

The data can be presented to interested users through specific systems built for this purpose, for example, in order to help them make decisions regarding the monitored system, depending on the data.

This dissertation presents a platform (Status Monitor), which already exists, that captures data from another, having a low degree of data observability. Therefore, some functionalities are proposed in order to try to mitigate the aforementioned problem, adding even more value to the existing product. These adictions allow users who have access to the data to take actions that, consequently, can improve the quality of the monitored application.

Of all the functionalities, those involving the creation of *dashboards* that display various information and metrics regarding servers, installations, and system components were implemented.

Agradecimentos

Em primeiro lugar, gostaria de agradecer ao meu orientador, o Professor Doutor Nuno Silva, por todo o suporte, disponibilidade e envolvimento no desenvolvimento deste projeto. Também quero agradecer ao supervisor da empresa, João Areias, por toda a disponibilidade e ajuda fornecida ao longo do tempo disponível para a realização do trabalho. Por fim, quero deixar um agradecimento muito especial à minha família e amigos próximos, em particular aos meus pais e ao meu melhor amigo André Novo, por todo o apoio incondicional ao longo destes últimos anos, e por me terem aturado, o que não deve ser nada fácil.

A concretização deste trabalho não seria possível sem todos vocês.

Muito obrigado, por tudo!

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Código

Lista de Acrónimos

API	*Application Programming Interface*.
CEO	*Chief Executive Officer*.
CNCF	*Cloud Native Computing Foundation*.
CPU	*Central Processing Unit*.
DBMS	*Database Management System*.
HTTP	*Hyper Text Transfer Protocol*.
IIS	*Internet Information Services*.
ISEP	Instituto Superior de Engenharia do Porto.
JSON	*JavaScript Object Notation*.
MVC	*Model-View-Controller*.
OTLP	*OpenTelemetry Protocol*.
SDK	*Software Development Kit*.
SMS	*Short Message Service*.
SMTP	*Simple Mail Transfer Protocol*.
SQL	*Structured Query Language*.
SSD	*Solid State Drive*.
SUS	*System Usability Scale*.
UI	*User Interface*.
UML	*Unified Modeling Language*.
UWP	*Universal Windows Platform*.
VM	*Virtual Machine*.

Capítulo 1

Introdução

Neste capítulo é apresentado o contexto geral onde o projeto se insere, seguido da identificação do problema existente que tem que ser resolvido, assim como os objetivos da solução a atingir para o mitigar e a abordagem utilizada para os alcançar. Por último, é apresentado o planeamento efetuado e a estrutura do documento.

1.1 Contexto

O presente projeto foi desenvolvido no âmbito da unidade curricular Tese/Dissertação/Estágio (TMDEI), pertencente ao segundo ano do Mestrado em Engenharia Informática, na área de especialização de Sistemas Computacionais, do Instituto Superior de Engenharia do Porto (ISEP). O projeto está inserido num contexto empresarial, tendo sido proposto por uma empresa designada Ciberbit.

A Ciberbit é uma empresa privada fundada em Coimbra, em 1995, por um grupo de engenheiros (Freixo 2021), que projeta e desenvolve "soluções de futuro, providenciando formas de gestão inteligente para diferentes sectores da indústria, comércio, serviços e muito mais"(Ciberbit 2023b). A mesma tem crescido constantemente seguindo os objetivos de "ser moderna, ser diferente, ser melhor"(S.A. 2023), fornecendo diversos serviços, desde consultoria tecnológica, até especialistas de implementação, desenvolvimento de *software*, e soluções *web* (Ciberbit 2023b).

A empresa tem como missão "desenhar e desenvolver soluções e produtos que variam de jogos até aplicações corporativas", sendo que o foco dela, atualmente, encontra-se na saúde e no retalho (S.A. 2023), com o desenvolvimento de dois produtos. Um deles é focado na gestão de cadeias de lojas de retalho, designado por CBRetail. O outro, este já de extremo interesse para o projeto desenvolvido, é designado por I'MTHOM (Freixo 2021).

Este produto consiste, nas palavras do *Chief Executive Officer* (CEO) atual da empresa Américo Amorim Pinto (S.A. 2023), numa "solução para a gestão integrada de unidades de saúde. É um produto sofisticado que utiliza as últimas tecnologias *web*. Tentámos fazer um produto que funcionasse bem numa pequena clínica, mas que, ao mesmo tempo, pudesse ser utilizado num hospital"(Freixo 2021). Por outras palavras, consiste numa plataforma que permite a gestão de várias operações referentes a unidades de saúde, quer sejam de cariz clínico, administrativo ou financeiro (Ciberbit 2023a). É, portanto, uma solução de *software* desenvolvida pela Ciberbit com o objetivo de apoiar o funcionamento de unidades de saúde em várias das suas vertentes.

Para qualquer produto de *software* como o supracitado, torna-se necessária a captação de diversos dados do mesmo, tanto por forma a identificar, por exemplo, origens de vários

erros que afetam o produto, como por forma a captar métricas, ajudando na análise do desempenho e comportamento desse mesmo *software*. A partir deste processo, surge o termo telemetria.

A telemetria consiste num "processo de comunicação altamente automatizado pelo qual os dados são colecionados a partir de instrumentos localizados em pontos remotos ou inacessíveis e transmitidos a equipamentos recetores para medição, monitorização, exibição e registo"(Britannica 2023). Falamos, portanto, de uma tecnologia que permite a medição e análise de dados à distância (Kechagia, Mitropoulos e Spinellis 2015). Esses dados são uma espécie de *feedback* proveniente do produto, sendo possível extrair deles várias informações que denunciam diversos estados do mesmo, cuja análise permite melhorar a capacidade de tomar decisões relativas a diversos aspetos do sistema (Riedesel 2021).

Pela importância atribuída aos dados de telemetria, torna-se necessária a criação de um sistema que seja capaz de receber esses dados da aplicação a ser monitorizada, e que os consiga mostrar aos devidos utilizadores de forma útil.

Surgiu, assim, a Status Monitor, uma solução de telemetria desenvolvida pela Ciberbit, a fim de responder à necessidade de captação de dados de telemetria, com a intenção de possuir módulos de alarmística, sinalização de eventos, *dashboards* de resumo de informação, gráficos representativos, e outros, para a plataforma I'MTHOM.

1.2 Problema

Assim como pode acontecer com qualquer produto de *software* existente, o mesmo poderá estar sujeito a diversas alterações, quer seja em termos de melhorar, modificar ou corrigir as funcionalidades já implementadas, ou implementar novas.

No caso da Status Monitor, não é muito diferente. Atualmente, este *software* apenas recebe certos eventos, como erros, que a aplicação I'MTHOM[1] envia, guardando os dados referentes, e permitindo a sua posterior consulta. É importante mencionar que não é realizado nenhum tipo de processamento ou análise automática sobre a informação recebida. Para além disto, existe um serviço tipo WatchDog que permite capturar eventos a partir de uma *Application Programming Interface* (API) disponibilizada pela aplicação, enviando posteriormente, esses dados para o Status Monitor.

Ou seja, o seu grau de observabilidade de dados é bastante baixo.

1.3 Objetivos

O objetivo principal deste projeto concentra-se no aumento do grau de observabilidade da aplicação. Isto é, pretende-se desenvolver métodos para captar uma maior quantidade de dados que relatam o que está a acontecer na mesma, como métricas sobre o estado de vários componentes.

[1] que a partir deste ponto, e até ao final deste capítulo, é referida como apenas aplicação, por questões de simplicidade

No âmbito deste projeto é possível definir os seguintes objetivos mais específicos:

- Criação de diversos *dashboards*, contendo:
  - Informações referentes a servidores da aplicação, envolvendo a criação de estruturas de dados que permitam suportar essas mesmas informações;
  - Um conjunto de métricas referentes a uma instalação da aplicação;
  - Um conjunto de métricas referentes a um componente da aplicação;
  - Informações e métricas das bases de dados utilizadas pela aplicação.
- Adoção do conceito de alertas com base em regras definidas, envolvendo a criação de componentes para criar e gerir alertas, para analisarem os dados e validarem se algum alerta deve ser ativado, e para enviarem notificações, no caso de algum alerta surgir, por email.

Os objetivos supracitados foram propostos tendo em conta a abordagem selecionada, que é apresentada na próxima secção. Com base na mesma, esta nova solução deve integrar todas as novas funcionalidades na já existente. Desta forma, a solução agrega valor ao cliente, que no caso é a própria empresa.

1.4 Abordagem

No âmbito deste projeto, foram identificadas 2 potenciais abordagens que podem ser adotadas para a implementação da solução, que se encontram descritas da seguinte forma:

- Usar soluções de observabilidade já disponíveis, criando toda uma nova solução com base nelas;
- Usar a solução já existente, adicionando mais informações relacionadas com os dados de telemetria.

Ambas as ideias supracitadas conseguem atender ao objetivo principal proposto, com a diferença de possuirem processos diferentes de desenvolvimento.

A ideia referente ao uso da solução já existente para desenvolver os objetivos foi a escolhida pelo autor desta dissertação, enquanto que a outra abordagem ficou ao encargo de um outro aluno inscrito em TMDEI. Contudo, é importante mencionar que a abordagem escolhida pode não corresponder à melhor.

Por forma a que os objetivos previamente enunciados possam ser concretizados, recorrendo à abordagem selecionada anteriormente, torna-se necessária a criação de uma outra abordagem que deverá de ser seguida. As tarefas a serem realizadas inerentes a essa abordagem são as seguintes:

- Numa primeira fase, é realizado um estudo e levantamento de limitações existentes na solução já desenvolvida, constituindo o problema a ser mitigado. Por forma a tornar isso possível, são definidos os objetivos que terão que ser atingidos por forma a melhorar a solução atual e ir em conta do que é pretendido pela empresa;
- Após a concretização da primeira fase, procede-se ao desenvolvimento do estado da arte, com a análise e pesquisa ampla dos diversos conceitos referentes à solução existente, como conceitos de monitorização, telemetria, entre outros, fundamentais para

uma melhor compreensão dos objetivos pretendidos, e consequentemente, por forma a torná-los atingíveis. Ainda nesta fase, o apuramento de soluções já existentes que podem ajudar na resolução do problema identificado é realizado;

- Com o estado da arte realizado, inicia-se o processo de identificação dos requisitos funcionais do sistema, assim como dos atributos de qualidade. Posteriormente, é realizada uma análise ao contexto de negócio, incluindo a apresentação do modelo de domínio, e a análise do estado atual da Status Monitor;
- Após a fase supracitada, procede-se à conceção dos artefactos de desenho, através da elaboração de diagramas da arquitetura, componentes e sistema, recorrendo à linguagem *Unified Modeling Language* (UML), para diversos graus de granularidade, que espelham a estrutura do projeto;
- Dá-se início à fase de implementação da solução proposta. Este processo tem como base tudo aquilo que foi definido nas fases antecessoras;
- Finalmente, é realizada a avaliação da solução, por forma a avaliar o estado do novo sistema. Esta avaliação pode ser realizada através do preenchimento de inquéritos a serem respondidos por indivíduos que têm acesso ao produto, que têm em consideração um conjunto de métricas a serem avaliadas, ou através da realização de testes de desempenho, são importantes como forma de avaliar a qualidade do *software*.

1.5 Planeamento

No que toca ao planeamento do projeto, foram realizadas várias reuniões com o supervisor da empresa, e com o orientador desta dissertação, tanto para a definição dos objetivos a serem concretizados, como para a verificação e acompanhamento dos desenvolvimentos efetuados.

Deste modo, foi definido um processo de desenvolvimento com base na metodologia Scrum, que consiste na realização de um processo de controlo e revisão de uma tarefa definida no planeamento após o término da mesma. Neste processo, tanto o supervisor como o orientador fornecem o seu *feedback* e opinião, apontando falhas existentes, procedendo-se, em caso necessário, às correções/melhorias sugeridas, promovendo uma melhoria contínua do trabalho realizado. Após isto, são definidas as novas tarefas a desenvolver, assim como os seus requisitos e prioridades.

Esta metodologia assenta em quatro pilares fundamentais que a área de gestão e controlo possui: Planear, Executar, Controlar/Verificar e Agir/Ajustar.

1.6 Estrutura do Documento

No primeiro capítulo é realizada uma contextualização do tema do documento, contendo uma breve apresentação da empresa Ciberbit e do produto I'MTHOM, que é utilizado como base para a solução a desenvolver. Para além disso, o problema proposto, assim como os objetivos do projeto, e a respetiva abordagem, por forma a se proceder à concretização dos mesmos, são definidos.

No segundo capítulo é realizado um levantamento do estado da arte atual referente a conceitos importantes relacionados com o tema do documento, tais como monitorização, telemetria, entre outros. Para além disto, é realizada uma análise a diversas soluções existentes no mercado que podem ajudar na resolução do problema identificado.

No terceiro capítulo são apresentados os requisitos funcionais e atributos de qualidade propostos para o desenvolvimento da solução.

No quarto capítulo é apresentado o contexto do negócio através da análise do mesmo e da solução já existente.

No quinto capítulo é definido o *design* arquitetural da solução, recorrendo ao uso de diagramas em notação UML.

No sexto capítulo é apresentado o trabalho de implementação do *software* descrito no capítulo anterior, com inclusão de algumas descrições técnicas dos componentes implementados.

No sétimo capítulo é definido o processo e métodos de avaliação a serem aplicados no projeto. Assim sendo, a avaliação da solução é realizada, assim como são apresentados os resultados da mesma.

Por último, no oitavo capítulo, é realizado um resumo do projeto implementado, sendo identificados os objetivos alcançados, assim como os que não foram concretizados, desenvolvimentos que podem ser considerados futuramente para a solução, além de uma apreciação final relativa ao desenvolvimento da mesma.

Capítulo 2

Estado da Arte

O presente capítulo introduz diversos conceitos relacionados com o projeto desenvolvido, focando-se, numa primeira fase, num enquadramento teórico, e finalizando com a apresentação de diversas soluções existentes atualmente no mercado, e as suas principais caraterísticas, por forma a entender como as mesmas podem ser utilizadas ou como podem contribuir para a implementação da solução.

2.1 Enquadramento teórico

Nesta secção são abordados conceitos teóricos importantes e adjacentes ao tema do projeto e que são necessários para a elaboração e entendimento do mesmo.

2.1.1 Importância da Monitorização

A monitorização de um sistema consiste num processo que permite a visibilidade e perceção significativa do estado desse mesmo sistema, ajudando na identificação de causas de vários comportamentos anormais, e consequente retificação desses problemas, por forma a evitar que os mesmos possam resurgir futuramente (Chakraborty e Kundan 2021). Nas palavras de Rob Ewaschuk, "monitorização significa colecionar, processar, agregar e apresentar dados quantitativos em tempo-real relativos a um sistema, como contagens e tipos de *queries*, contagens e tipos de erros, tempos de processamento, (...)"(Beyer et al. 2016).

Este processo é muito importante para identificar, por exemplo, tendências de desempenho ao longo de um período de tempo. Também ajuda na "procura da utilização histórica de recursos", e na tomada de decisões, podendo também servir para um possível redesenho da própria arquitetura do sistema, e, mais importante, no fornecimento de diversas informações sobre o mesmo. Também é bastante útil para descobrir comportamentos anormais antes que os mesmos se tornem problemas mais complicados, contribuindo para a manutenção da disponibilidade e qualidade dos serviços (Chakraborty e Kundan 2021).

Os sistemas que estão a ser monitorizados podem emitir dados em intervalos de tempo regulares, de forma automática, ou aquando da ocorrência de um determinado evento. Com recurso a sistemas de monitorização, torna-se possível, de acordo com (Chakraborty e Kundan 2021):

- A obtenção de informações referentes a potenciais problemas do sistema, de forma proativa;
- Uma análise rápida de um problema que já ocorreu e sua respetiva correção;
- Determinar a saúde geral do sistema.

Por isso, para qualquer sistema de *software*, torna-se importante a captura deste tipo de dados. As técnicas que são utilizadas para monitorização nos tempos atuais são variadas e transversais às áreas de processamento de dados em tempo-real e análise estatística de dados. A visualização de dados também possui um papel bastante importante, uma vez que os dados processados devem ser apresentados de forma significativa para os utilizadores, num formato legível (Chakraborty e Kundan 2021).

2.1.2 Sistemas de Telemetria

Como previamente mencionado, em qualquer sistema de *software*, saber o que se passa dentro do mesmo torna-se extremamente benéfico para questões de manutenção, análise e *debugging*.

Assim, surge o termo telemetria. Como mencionado na secção 1.1, este refere-se ao *feedback* proveniente de sistemas de produção que denuncia diversos estados desses mesmos sistemas. Esses dados são capturados e podem ser mostrados aos utilizadores interessados através de outros sistemas, próprios para tal, por forma a ajudá-los a tomar decisões relativas aos sistemas em monitorização, melhorando, de certo modo, a sua capacidade de decisão (Riedesel 2021).

Entre os anos 2010 e 2020, a indústria presenciou o surgimento de métricas e *distributed tracing*, que, combinados com *logs*, formaram os conhecidos três pilares da observabilidade, que são explorados nas secções seguintes, com exceção de um deles, referente a *distributed traces*, uma vez que os mesmos não são aplicados ao âmbito do projeto (Riedesel 2021). A observabilidade é um conceito que descreve um mecanismo que permite a captação de qualquer tipo de informação relativa a um determinado sistema, por forma a permitir a construção de uma imagem completa do estado interno de um sistema, em tempo real. Esses dados podem ajudar os utilizadores a ganharem visibilidade em falhas não muito explícitas, por exemplo (Chakraborty e Kundan 2021).

2.1.3 Métricas

As métricas dizem respeito a informações númericas, no caso, valores que correspondem a um determinado parâmetro em observação (Gatev 2021), que pode corresponder, por exemplo, à utilização do processador de uma máquina virtual, ao espaço de armazenamento livre, ou até mesmo ao tráfego de entrada de uma rede virtual (Microsoft 2023a). Estas são medidas ao longo do tempo, em intervalos regulares e fixos, que podem corresponder a um segundo, ou vários minutos, conhecidos como granularidade ou resolução (Chakraborty e Kundan 2021).

Para observar o comportamento de um sistema de forma eficiente, torna-se importante encontrar a granularidade necessária. Por exemplo, caso a mesma seja muito pequena, isso irá impactar negativamente o desempenho do sistema, uma vez que o mesmo será acedido de forma bastante frequente, para além de poder resultar numa falta de variação significativa dos dados. Em contrapartida, se a granularidade for grande demais, informações valiosas poderão ser perdidas, eliminando o propósito da captura das métricas. Por culpa desses cenários, é necessário adotar uma estratégia eficiente no que toca a definir a granularidade. Uma delas consiste na atribuição de diferentes níveis da mesma para diferentes componentes do sistema, por exemplo (Chakraborty e Kundan 2021).

O conjunto das observações efetuadas sobre um parâmetro é designado de *time series*, traduzido do inglês para séries temporais, que podem ajudar na identificação de *outliers*, que seriam valores que, num determinado grupo, apresentam um grande afastamento em relação aos demais (Infopédia 2023), nesse mesmo conjunto, assim como entender o motivo que justifica a presença do mesmo (Chakraborty e Kundan 2021).

Por culpa da natureza numérica das métricas, tornam-se bastante convenientes para qualquer tipo de agregação, sumarização, e correlação, que podem ser apresentadas a partir de *dashboards*, onde é possível explorar algumas tendências relativas à aplicação (Gatev 2021).

Uma sequência de métricas capturas pode ser bastante significativa no que toca a entender o estado de um sistema, fornecendo uma imagem dinâmica e em tempo-real do mesmo. Ao ajudarem na deteção de anomalias e padrões, elas contribuem para a identificação de falhas de uma forma proativa, antes que possa ocorrer um problema maior (Chakraborty e Kundan 2021).

Cada métrica é composta por uma medição, conhecida como *data value*, um *timestamp* do momento em que a medição foi registada, e um conjunto de *tags* e *labels* que descrevem essa métrica. As *tags* permitem a agregação de informações, sem que a forma como as métricas são capturadas seja alterada. Ou seja, correspondem a metadados que complementam uma métrica com informação contextual. Ao associá-las com as métricas, a reconfiguração dos dados colecionados, assim como o agrupamento de todas as métricas de um determinado conjunto de serviços, torna-se mais simples (Chakraborty e Kundan 2021).

2.1.4 Eventos

Para além da captura de métricas, também a captura de eventos, que correspondem a ocorrências como mudanças de código, ou até mesmo alertas internos, torna-se importante, uma vez que podem fornecer informação contextual crucial para entender certas mudanças comportamentais do sistema (Chakraborty e Kundan 2021).

Poderia se pensar em capturar cada evento que acontecesse a qualquer momento, contudo, isso acaba por não ser muito viável, uma vez que cada captura requer sempre algum espaço de armazenamento, o que aumenta os custos. Por culpa disto, faz sentido capturar apenas os eventos que não são muito frequentes, mas que são críticos por natureza (Chakraborty e Kundan 2021).

Como supracitado, os eventos também podem ser utilizados para gerar alertas, até porque alguns deles podem ser extremamente raros, mas críticos, que requerem alguma intervenção humana, como por exemplo, uma falha num *backup* noturno (Chakraborty e Kundan 2021).

Todas as informações referentes a um determinado evento que aconteceu num determinado período de tempo podem ser representadas sob a forma de *logs*, que são explorados na secção seguinte.

2.1.5 Logs

Em relação aos *logs*, os mesmos correpondem a registos imutáveis para um determinado evento que ocorreu num determinado instante temporal (Picoreti et al. 2018).

Eles podem ajudar na recriação de ações passadas, uma vez que correpondem a registos que providenciam dados altamente fidedignos e com um contexto detalhado sobre o evento, podendo ser imprimidos aquando da ocorrência desse mesmo evento que o despoletou, ou

simplesmente armazenados para posterior análise (Chakraborty e Kundan 2021). Dependendo do seu nível de especificidade, poderá ser possível identificar as causas principais de alguns problemas de desempenho, ou até mesmo realizar um *debug* a um problema sem a necessidade de atrelar à tarefa nenhum *debugger* (Gatev 2021).

Normalmente, os *logs* podem ser gerados num dos seguintes três formatos, de acordo com (Gatev 2021):

- **Texto simples:** Consiste na forma mais comum em que eles podem ser apresentados;
- **Estruturado:** Eles podem assumir formatos estruturados, como *JavaScript Object Notation* (JSON), tornando a sua posterior interpretação mais simples;
- **Binário:** Eles são armazenados num formato binário. Eles não são um formato muito comum e, no geral, são bastante úteis apenas para os sistemas que os geram. Um exemplo deste tipo diz respeito aos *logs* de alguns sistemas de gestão de bases de dados relacionais, como o MySQL.

Quando é tomada a decisão de incluir *logs* num produto, é crucial definir quais as informações que devem ser registadas, e como. Por isso, sempre que possível, cada um deles deve conter os seguintes elementos, de acordo com (Dávideková e Gregu Ml 2016):

- Identificação do tipo de ocorrência, que pode ser um aviso, informação, ou erro, assim como os elementos/componentes do sistema afetados por essa ocorrência;
- Presença da data e hora da ocorrência em causa, juntamente com o fuso horário respetivo;
- Identificação do sistema, aplicação ou classe onde ocorreu o evento;
- Identificação do autor responsável por ter despoletado o evento;
- Presença de informações que revelam se a ocorrência afetou apenas o utilizador que a iniciou, ou se afetou mais para além desse;
- Identificação do motivo que justifica a ocorrência do evento, que podem ser diversos, desde introdução incorreta de *passwords*, até opções obrigatórias de preencher num formulário não terem sido efetivamente preenchidas, entre outros.

Considerando o que foi mencionado até ao momento, a ideia de realizar *logs* para qualquer tipo de ação que seja realizada por qualquer tipo de serviço da aplicação parece fazer sentido. Contudo, é preciso ter em conta as implicações de desempenho desse tipo de abordagem. Isto é, dependendo do sistema de *logging* que esteja a ser utilizado, a criação de múltiplos *logs* pode prejudicar o desempenho de um determinado serviço (Gatev 2021), sendo que, quanto maior for a quantidade de *logs* e o seu nível de detalhe, maior será o impacto negativo que os mesmos terão no desempenho do serviço (Picoreti et al. 2018).

Para além disso, o excesso dos mesmos pode ter como consequência negativa altos custos operacionais para a aplicação, uma vez que estes são tipicamente guardados em sistemas externos, por forma a poderem ser analisados posteriormente (Gatev 2021).

2.2 Soluções Existentes

Na presente secção, é apresentada uma análise e descrição de diversas soluções existentes no mercado, referentes ao tema da dissertação, por forma a identificar os contributos

que as mesmas fornecem e que podem ajudar no desenvolvimento da mesma. No caso, é apresentada, numa primeira fase, uma solução que pode complementar a que se pretende desenvolver, fornecendo suporte para a captação de métricas do I'MTHOM. Já na segunda fase, é apresenta uma solução com funcionalidades semelhantes às que se pretendem implementar, fornecendo uma visão geral de como esta funciona. Por último, é apresentado um sumário.

2.2.1 OpenTelemetry

Como já é sabido, por forma a que um sistema possa ser observável, é necessário que o mesmo envie diversos dados de telemetria, como métricas, *logs* e *traces*, para um outro sistema responsável por capturá-los e apresentá-los a utilizadores específicos. Quando um sistema possui código responsável pela emissão destes dados, podemos dizer que o mesmo é instrumentado (OpenTelemetry 2023f).

Há uns anos atrás, a forma como o código era instrumentado variava dependendo do sistema que era utilizado para receber e apresentar os dados. Muitos deles tinham as suas próprias bibliotecas e agentes específicos, que eram utilizados pelo sistema instrumentado. Contudo, se a empresa que utilizasse um determinado sistema de captação mudasse para um outro, o código teria que ser reinstrumentado, e novos agentes teriam que ser configurados, por forma a que os dados de telemetria pudessem ser enviados para a nova ferramenta a ser utilizada. Por forma a resolver este problema, foram criados os seguintes projetos *open-source* (OpenTelemetry 2023f):

- OpenTracing, que consistia num projeto desenvolvido pela *Cloud Native Computing Foundation* (CNCF) que fornece uma API que permite o envio de dados de telemetria para um determinado sistema;
- OpenCensus, que consistia num projeto desenvolvido pela Google Open Source Community, que fornece um conjunto de bibliotecas para uma determinada linguagem de programação específica, e que poderia ser utilizada por desenvolvedores para capturar diversos dados telemétricos e os enviar para algum sistema suportado.

Em meados de 2019, os dois projetos supracitados foram combinados num único, dando origem ao OpenTelemetry, que combina o melhor dos dois, e muito mais (OpenTelemetry 2023f). O seu objetivo consiste em fornecer, para cada linguagem de programação que suporta, uma API, que define tipos de dados e operações para gerar dados de *tracing*, métricas e *logging*, um *Software Development Kit* (SDK), que consiste na implementação da API nessa mesma linguagem (OpenTelemetry 2023d), e ferramentas para capturar, transformar e enviar dados de telemetria para sistemas especializados que utilizam esses dados para diversos propósitos. De uma forma mais detalhada, este produto fornece aos seus utilizadores, tornando a sua adotação bastante simples, o seguinte (OpenTelemetry 2023f):

- Uma biblioteca de instrumentação para diversas linguagens de programação que suportam instrumentação manual e automática. A instrumentação automática permite a captura de dados de telemetria sem que o código fonte precise de ser alterado, já a manual necessita que o mesmo tenha que ser modificado (OpenTelemetry 2023e);
- Um *collector*, que consiste num simples binário e que pode ser implantado de diversas formas diferentes. Ele pode receber, processar e exportar dados de telemetria para diversos sistemas. Esta ferramenta não necessita de ser obrigatoriamente utilizada, podendo o envio dos dados ser direto para os sistemas. Por mais que o envio direto se

traduza numa forma de obter valores rapidamente, e que os resultados sejam decentes em ambientes de desenvolvimento ou de pequena escala, de forma geral, o seu uso é recomendado, uma vez que permite que um serviço envie dados de forma rápida e o mesmo pode tratar dos mecanismos de gestão adicional, como filtragem de dados sensíveis, ou até mesmo criptografia. Um outro ponto a considerar é a facilidade de configuração do mesmo (OpenTelemetry 2023a). Ele possui dois métodos de implantação primários, de acordo com (OpenTelemetry 2023c):

  - Agente: Consiste numa instância do collector a executar com a aplicação, ou no mesmo *host* que ela;
  - *Gateway*: Consiste na execução de uma ou mais instâncias do *collector* como um serviço *standalone*, por exemplo, num *container*.

  Relativamente à sua constituição, o *collector* é composto por três componentes que acedem a dados de telemetria, a saber (OpenTelemetry 2023b):

  - *Receivers*: Descrevem mecanismos de como os dados chegam ao *collector*. Por outras palavras, são componentes responsáveis por agregar e receber os dados de telemetria de um sistema instrumentado (Grafana 2023);
  - *Processors*: Usam os dados recebidos e tomam decisões sobre os mesmos, como limpá-los, remover ou adicionar atributos, entre muitos outros, enriquecendo e transformando os dados colecionados (Grafana 2023);
  - *Exporters*: Descrevem mecanismos de como os dados são enviados para um ou mais destinos.

  Para além disto, também existem extensões, que fornecem funcionalidades que podem ser adicionadas ao *collector*, como autenticação ou até mesmo *debugging*, mas que não requerem um acesso direto aos dados (OpenTelemetry 2023b).

- Uma implementação *end-to-end* que permite gerar, emitir, capturar, processar e exportar dados de telemetria;
- Controlo completo desses dados, podendo os mesmos serem enviados para múltiplos destinos em paralelo.

Para além disto, o projeto define um protocolo designado *OpenTelemetry Protocol* (OTLP), que tem como propósito transmitir dados de telemetria. Ele pode ser utilizado para transmiti-los a partir do SDK até ao *collector*, assim como do próprio *collector* até ao sistema de telemetria. A sua especificação define mecanismos de codificação, transporte e entrega de dados. Contudo, é possível que o protocolo não seja utilizado, caso estejam a ser utilizadas ferramentas e *frameworks* que já proveem de mecanismos de instrumentação. O *collector* consegue receber dados a partir de outros protocolos, usando um *receiver* adequado para o efeito (Logz.io 2023). A figura 2.1 representa uma visão geral do processo de funcionamento do OpenTelemetry, usando o *collector*.

É importante ressaltar que o OpenTelemetry não consiste num sistema de telemetria, mas sim uma ferramenta que permite o envio de dados de telemetria para uma grande variedade de sistemas diferentes (OpenTelemetry 2023f).

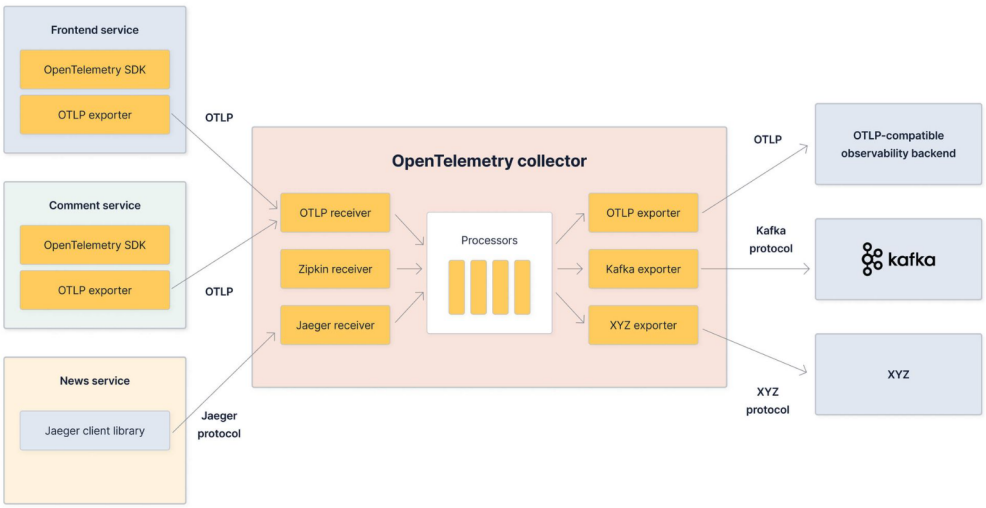

Figura 2.1: Processo de funcionamento do OpenTelemetry. Fonte: (Timescale 2021).

Como previamente mencionado, o *collector* consegue receber dados de telemetria a partir de outros protocolos, ou formatos, por forma a que possam ser enviados, após o seu processamento, para sistemas que suportem essa mesma estrutura. Um sistema desse tipo é analisado na secção seguinte.

2.2.2 New Relic

A New Relic consiste numa solução de observabilidade que permite melhorar a qualidade do *software* a ser monitorizado, recebendo dados provenientes do mesmo, por forma a permitir uma melhor "compreensão do sistema, análise desses dados de forma eficiente, e resposta a incidentes antes que os mesmos se tornem problemas"(NewRelic 2023d).

Estes dados podem ser apresentados para os utilizadores através de *dashboards*, que os agrupam. Para além disto, a plataforma fornece suporte nativo para o OpenTelemetry, mencionado na secção anterior, e permite integração com diversas tecnologias, como NodeJS, MySQL, .NET Core, Kafka, entre outras (NewRelic 2023b).

A solução permite monitorizar diversos dados referentes ao desempenho e erros da aplicação monitorizada, infraestruturas, como bases de dados e servidores, permitindo visualizar o estado dos mesmos, logs, e até mesmo aplicações móveis (NewRelic 2023c).

Para além do que foi supracitado, também é possível gerar alertas a partir desta plataforma. Um utilizador pode criar uma condição estática completamente customizável que gera um alerta caso o comportamento de algum atributo do sistema, como latência, exceda algum valor pré-definido, por exemplo. Eles possuem um sistema de deteção de anomalias, que recorre a técnicas de *machine learning*, que permite "monitorar padrões e tendências nos dados", notificando a equipa assim que ocorrer um desvio do comportamento expectável (NewRelic 2023a).

Após a deteção de algum comportamento irregular, é aplicado um processo de correlação lógico, que tenta comparar todos os eventos provenientes da aplicação monitorizada com

base em alguns atributos dos mesmos, como tempo, contexto, ou fonte de origem. Assim, se os mesmos possuirem relações neste sentido, é gerado apenas um alerta para esses eventos, em vez de múltiplos redundantes (NewRelic 2023a).

Após os processos mencionados, os utilizadores devidos são notificados da geração do alerta (NewRelic 2023a).

2.2.3 Sumário

Esta secção apresentou algumas soluções que permitem fornecer um exemplo de um possível fluxo de como dados telemétricos seriam apresentados para utilizadores, desde mecanismos de captação, até à posterior apresentação dos mesmos, aplicando conceitos de alertas sempre que necessário. No caso, nenhuma das soluções é utilizada na implementação da solução. A empresa tem intenções de realizar toda a parte da instrumentação do código do I'MTHOM fazendo uso dos seus próprios mecanismos, sem que o autor tenha que se preocupar com essa parte do processo.

Capítulo 3

Engenharia de Requisitos

O presente capítulo tem como objetivo eliciar, sistematizar e representar os requisitos do projeto em desenvolvimento, tanto funcionais como não funcionais, num processo designado de Engenharia de Requisitos. A primeira secção apresenta os casos de uso definidos para o projeto, assim como o respetivo diagrama de casos de uso em notação UML. Já a segunda secção aborda todos os atributos de qualidade, que contém os requisitos não funcionais definidos, recorrendo ao modelo FURPS+.

3.1 Requisitos funcionais

De acordo com (Malan, Bredemeyer et al. 2001), os requisitos funcionais "capturam o comportamento esperado de um sistema. Esse comportamento pode ser expresso como serviços, tarefas ou funções que o sistema necessita para funcionar". Este tipo de requisitos podem ser expressos em casos de uso, que definem um conjunto de interações orientadas a um determinado objetivo entre atores externos ao sistema, que pretendem atingir um determinado objetivo, e o sistema em si. Um ator pode representar tanto um conjunto de utilizadores, como papéis que eles podem assumir, ou até mesmo outros sistemas. Um conjunto completo de todos os casos de uso especifica as diversas formas de interação com o sistema, assim como o comportamento esperado do mesmo para responder às mesmas.

Este conjunto pode ser representado graficamente sob a forma de um diagrama de casos de uso (Malan, Bredemeyer et al. 2001). A figura 3.1 apresenta esse mesmo diagrama, aplicado ao tema desta dissertação, cujos casos de uso se encontram alinhados com os objetivos definidos na secção 1.3. De momento, apenas é considerada a existência de um único ator, denominado de "Administrador".

De uma forma mais detalhada, o administrador pode realizar as seguintes tarefas, que se encontram organizadas pelo seu grau de prioridade. Ou seja, a tarefa com maior prioridade é apresentada no primeiro ponto, e assim sucessivamente:

- Visualizar diversas informações referentes a uma instalação do I'MTHOM, como número total de sessões, de exceções, quantidade de servidores com determinado estado, uma listagem dos eventos mais recentes, número total de exceções do último mês, e do presente dia, entre outros;
- Visualizar diversos dados referentes aos servidores onde uma determinada instalação do I'MTHOM se encontra a executar. Esses dados são referentes a IP do servidor, versão das plataformas que são utilizadas, *uptime* e estado, sendo que este último corresponde a uma métrica que se pretende captar;

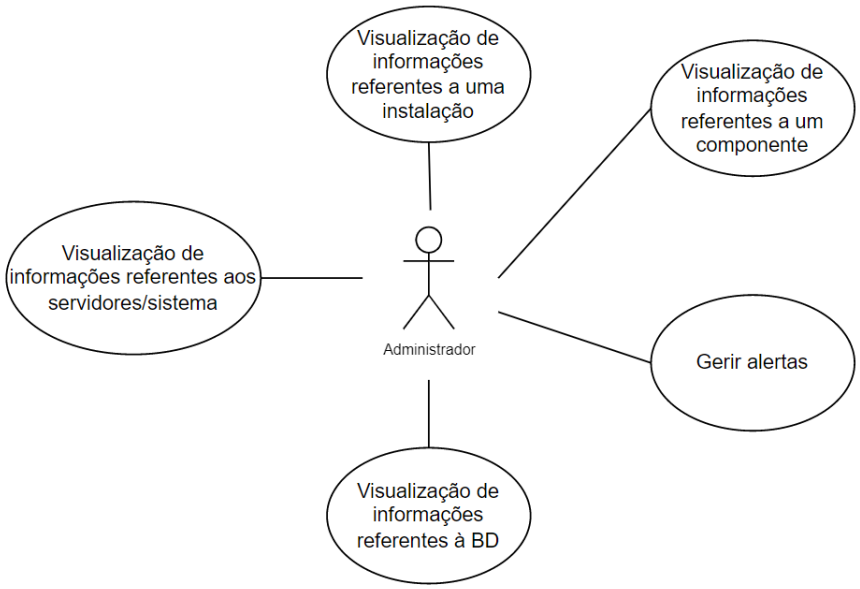

Figura 3.1: Diagrama de casos de uso.

- Visualizar diversos dados respetivos a um componente de uma determinada instalação do I'MTHOM, como número de exceções nesse componente na última hora, tempo médio de resposta de um *ping*, e gráficos que apresentam informações referentes a eventos, exceções, e desempenho semanais registadas;
- Visualizar métricas provenientes das diversas bases de dados utilizadas pelo I'MTHOM, que correspondem a listas de queries que mais *Central Processing Unit* (CPU) consomem, que mais tempo demoram a ser executadas, que mais vezes são executadas num determinado intervalo de tempo, e que mais leituras efetuam;
- Gerir alertas, por forma a que os mesmos sejam ativados aquando da concretização de uma determinada condição, como estados de determinados servidores, estatísticas da base de dados desatualizadas, entre outras. Aquando da ocorrência de um alerta, os utilizadores devidos devem ser notificados via email.

3.2 Atributos de qualidade

Os atributos de qualidade, também conhecidos como requisitos de qualidade, "declaram propriedades adicionais relativas à qualidade que os efeitos funcionais do software deveria ter"(Lamsweerde 2009). Existem diversos modelos de qualidade de *software* que podem ser utilizados nomeadamente o modelo FURPS+, que foi o escolhido para este projeto. Este modelo classifica os atributos de qualidade em requisitos funcionais e não-funcionais (Janocová 2012). Os requisitos não-funcionais definem restrições na forma como o produto a ser desenvolvido deve satisfazer os seus requisitos funcionais, ou na forma como deve ser desenvolvido (Lamsweerde 2009). A categoria "funcionalidade"diz respeito à definição de requisitos funcionais, enquanto que as restantes a não-funcionais, como usabilidade, confiabilidade, desempenho, suportabilidade, e restrições de desenho, implementação, *interface* e físicas.

Os requisitos de cada uma das categorias do modelo encontram-se presentes nas secções seguintes.

3.2.1 Atributos de Funcionalidade

Esta categoria apresenta as principais funcionalidades do produto, não tendo que ser unicamente específicas do domínio. Algumas das preocupações que se encaixam nesta categoria são a auditoria, localização, segurança, email, entre outras (Eeles 2005). No âmbito deste projeto, foram identificadas as seguintes:

- Um utilizador que queira aceder ao sistema deve estar devidamente autenticado no mesmo;
- O sistema deve notificar todos os clientes interessados em algum surgimento de um alerta, espoletado por condições pré-definidas.

3.2.2 Atributos de Usabilidade

A presente categoria é focada em caraterísticas relacionadas com a estética e consistência da interface do utilizador do sistema (Eeles 2005). No contexto deste projeto, foi identificado o seguinte requisito:

- O sistema deve apresentar as informações referentes a dados de telemetria aos utilizadores, de uma forma simples e prática, recorrendo a *dashboards*.

3.2.3 Atributos de Confiabilidade

Esta categoria concentra-se na captação de caraterísticas como a disponibilidade, e capacidade do sistema em recuperar de alguma falha (Eeles 2005). Também se refere à integridade e conformidade do *software* (QualidadeBR 2008). A mesma possui os seguintes requisitos:

- O sistema deve validar todos os dados recebidos, rejeitando injeções de código, de acordo com o OWASP;
- A integridade dos dados de telemetria deve ser garantida aquando do seu transporte;
- A autenticidade dos dados de telemetria deve ser garantida aquando do seu transporte.

3.2.4 Atributos de Desempenho

Esta categoria é focada em caraterísticas relativas ao desempenho do sistema, como taxa de transferência de informação, tempo de resposta, consumo de mémoria, utilização da CPU, entre outros (Eeles 2005; QualidadeBR 2008). Os requisitos especificados para esta categoria são:

- Um utilizador deve conseguir aceder aos dados das *dashboards* num período de 2 segundos em 90% dos casos;
- O sistema deve ser capaz de ser utilizado por vários utilizadores em simultâneo;
- Quando um alerta é espoletado, o sistema deve enviar notificação aos devidos utilizados em cerca de 10 segundos em 90% dos casos.

3.2.5 Atributos de Suportabilidade

Estes requisitos dizem respeito a caraterísticas relacionadas com a testabilidade, adaptabilidade, manutibilidade, compatibilidade, configurabilidade, escalabilidade, entre outros (Eeles 2005). Para esta categoria, foi identificado o seguinte requisito:

- O sistema deve suportar diversos navegadores *web*, como Chrome, Safari, Edge, entre outros.

3.2.6 Restrições de Desenho

Esta categoria especifica determinadas opções ou restrições para desenho do sistema (Eeles 2005). Nenhum requisito foi proposto para esta categoria.

3.2.7 Restrições de Implementação

Esta secção "especifica ou restringe a codificação ou construção do sistema"(Eeles 2005). Nenhum requisito foi proposto para esta categoria.

3.2.8 Restrições de Interface

Estas restrições referem-se a sistemas externos com os quais o sistema interage, ou até mesmo restrições ou formatos usados nessas mesmas interações (Eeles 2005). Foram identificadas as seguintes restrições:

- Todas as comunicações efetuadas entre os clientes e o sistema devem fazer uso de protocolos seguros, como HTTPS;
- O sistema deve expor uma *interface* de comunicação com o exterior, nomeadamente uma RESTful;
- O sistema deve suportar o envio de emails recorrendo ao *Simple Mail Transfer Protocol* (SMTP).

3.2.9 Restrições Físicas

Esta secção especifica restrições físicas impostas pelo *hardware* onde o sistema se encontra (Eeles 2005). Nenhum requisito foi proposto para esta categoria.

Capítulo 4

Análise

Este capítulo descreve o contexto do negócio através da análise do mesmo e da solução já existente. Numa primeira fase, são apresentados e analisados os diversos conceitos de negócio existentes numa perspetiva de engenharia de *software*, o que permite uma melhor compreensão do projeto e dos requisitos definidos. Esta análise é feita recorrendo a diagramas UML, no caso, um modelo de domínio. Já numa segunda fase, é realizada uma análise da solução existente, por forma a fornecer ao leitor mais detalhe do que já se encontra implementado, e respetivas limitações do sistema.

4.1 Análise de negócio

Esta secção descreve e analisa o domínio do problema. Para tal, é apresentado o modelo de domínio desenvolvido, que evidencia as entidades de negócio relevantes para o projeto, e respetivas definições.

4.1.1 Modelo de domínio

No contexto deste projeto, foi desenvolvido o modelo de domínio que se encontra presente na figura 4.1, que representa as principais entidades de negócio identificadas, através de classes conceptuais, e respetivos atributos e associações entre elas.

A partir da análise do modelo de domínio, é possível identificar os seguintes conceitos, que se encontram descritos da seguinte forma:

- **Cliente:** Representa uma organização/empresa ou pessoa que usufrui de algum produto da empresa;
- **Produto:** Diz respeito a um produto desenvolvido pela empresa e disponibilizado para os seus clientes;
- **Instalação:** Corresponde a um ambiente de execução, isto é, uma infraestrutura física ou virtual que suporta a execução de uma determinada instância de um produto;
- **Servidor:** Corresponde a um servidor de uma determinada instalação;
- **PapelServidor:** Descreve a função que um determinado servidor desempenha;
- **TipoServidor:** Descreve o tipo do servidor;
- **Evento:** Representa o registo de um acontecimento que ocorre ao nível de um determinado servidor, acompanhado com a data de captura do mesmo e alguns metadados. No contexto deste projeto, um evento corresponde a um tipo de *log*;

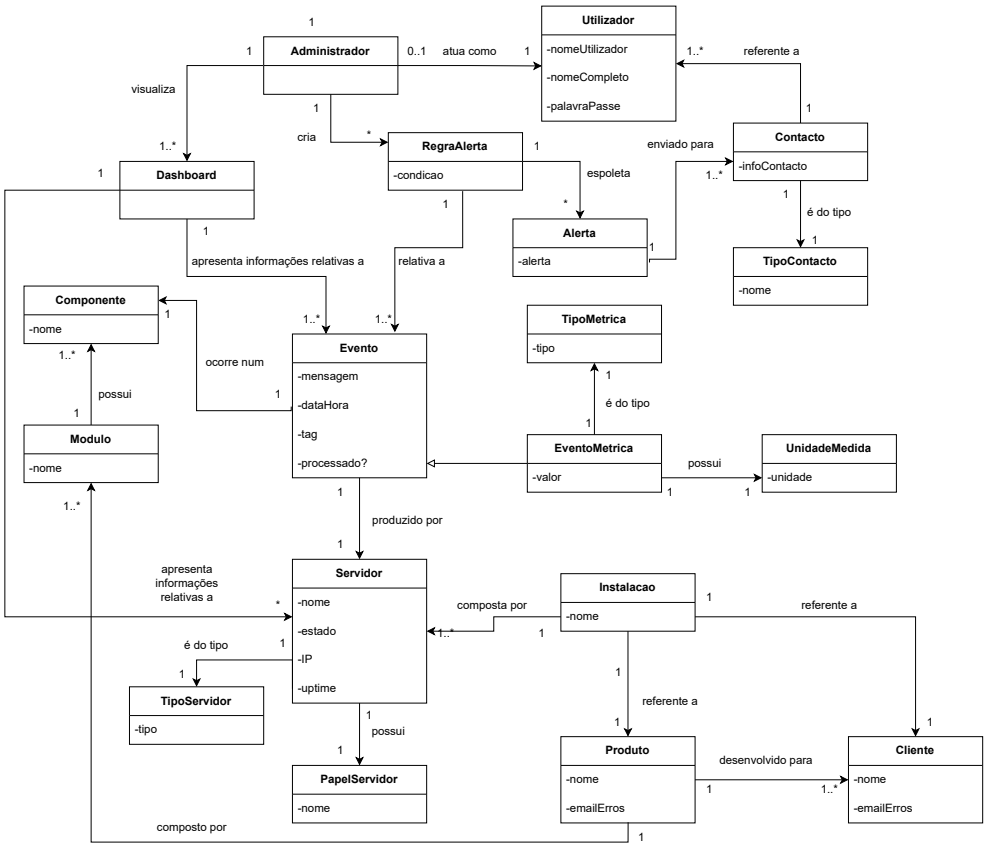

Figura 4.1: Modelo de domínio.

- **EventoMétrica:** Corresponde a um tipo de evento que regista informações referentes a métricas captadas num servidor;
- **TipoMétrica:** Corresponde ao tipo de métrica de um evento de métrica;
- **UnidadeMedida:** Representa a unidade de medida de um valor de uma métrica;
- **Módulo:** Coresponde a uma pequena área de *software* de um determinado produto;
- **Componente:** Corresponde a uma parte de um módulo;
- **Utilizador:** Representa um utilizador registado na plataforma, e que consegue aceder ao sistema;
- **Administrador:** Representa um tipo de utilizador responsável por visualizar diversos tipos de *dashboards*, assim como criar regras para espoletar alertas, e ser notificado dos mesmos, aquando da sua ocorrência;
- **Dashboard:** Painel responsável por apresentar diversas informações de servidores, ou diversos registos de métricas, ou até mesmo métricas obtidas a partir de processos aplicados a esses mesmos registos;
- **RegraAlerta:** Corresponde a uma regra que define um determinado comportamento do sistema, que é utilizado aquando da captação de eventos, por forma a verificar se o mesmo ocorreu ou não;

- **Alerta:** Corresponde a uma notificação que é enviada no caso de alguma regra ser ativada aquando da captação de eventos;
- **Contacto:** Representa uma informação que permite estabelecer uma comunicação com um determinado utilizador ou grupo de utilizadores do sistema;
- **TipoContacto:** Descreve o tipo de contacto de um determinado utilizador.

Dos conceitos e relações supracitados, alguns já se encontravam presentes na versão inicial da Status Monitor, enquanto outros foram adicionados por forma a resolver o problema desta dissertação, e, consequentemente, dar reposta aos requisitos enunciados. Os conceitos adicionados são os referentes a **Servidor**, **PapelServidor**, **TipoServidor**, **EventoMetrica**, **TipoMetrica**, **UnidadeMedida**, **RegraAlerta**, e **Alerta**.

Para além disto, no modelo apresentado, não são representados todos os tipos de eventos existentes na plataforma, uma vez que os mesmos não são relevantes para o âmbito deste projeto.

Relativamente aos atributos dos servidores, o *uptime* e o estado são calculados a partir de métricas específicas que são capturadas. No caso, falamos de uma métrica que permite determinar o estado do servidor, que por enquanto, apenas pode ser um dos seguintes: (i) OK, (ii) Warning, e (ii) NOK. Caso o valor recebido corresponda ao primeiro, e o último estado registado seja o segundo ou terceiro, o estado é modificado para o novo, e o *uptime* é iniciado (no caso do último ser o terceiro), terminando caso o servidor assuma um estado NOK. Relativamente ao significado dos estados, no contexto deste projeto, o primeiro significa que o servidor se encontra a correr sem nenhum tipo de problema. O segundo estado avisa os utilizadores que existem eventos do tipo exceção ou operação mal-sucedida que ainda não foram resolvidos. Já o terceiro é atingido quando um servidor deixa de responder a pedidos.

4.2 Análise da solução existente

A Status Monitor representa a plataforma responsável por receber diversos tipos de eventos provenientes de uma plataforma chamada I'MTHOM, desenvolvida pela empresa. Esses eventos são armazenados numa base de dados, e apresentados aos seus utilizadores. Contudo, por forma a um utilizador conseguir ter acesso ao menu principal, é necessário que o mesmo se autentique. A figura 4.2 apresenta a respetiva página. Atualmente, não é possível registar novos utilizadores na plataforma, sendo que, para adicionar um novo, é necessária uma inserção direta na base de dados.

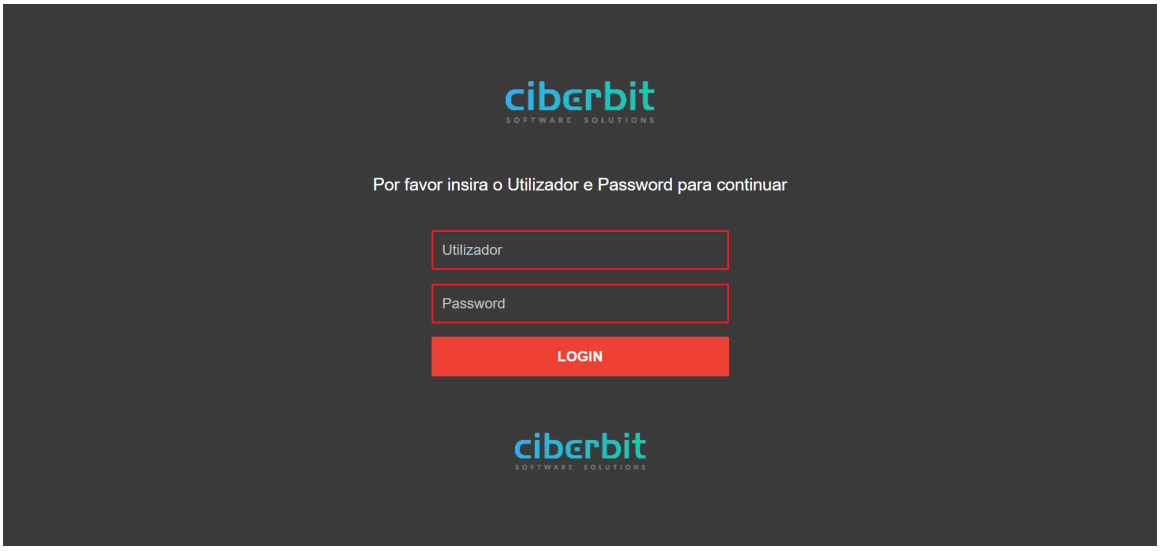

Figura 4.2: Página de autenticação.

Após o processo de autenticação é apresentada a página principal da aplicação para o utilizador, com base no seu tipo (como administrador). Dependendo do tipo, opções diferentes poderão aparecer, de acordo com as permissões. Uma imagem representativa desse menu consta na figura 4.3, que no caso, corresponde a uma página que é apresentada para um utilizador com permissões de administrador, tendo, portanto, acesso a todas as funcionalidades do sistema.

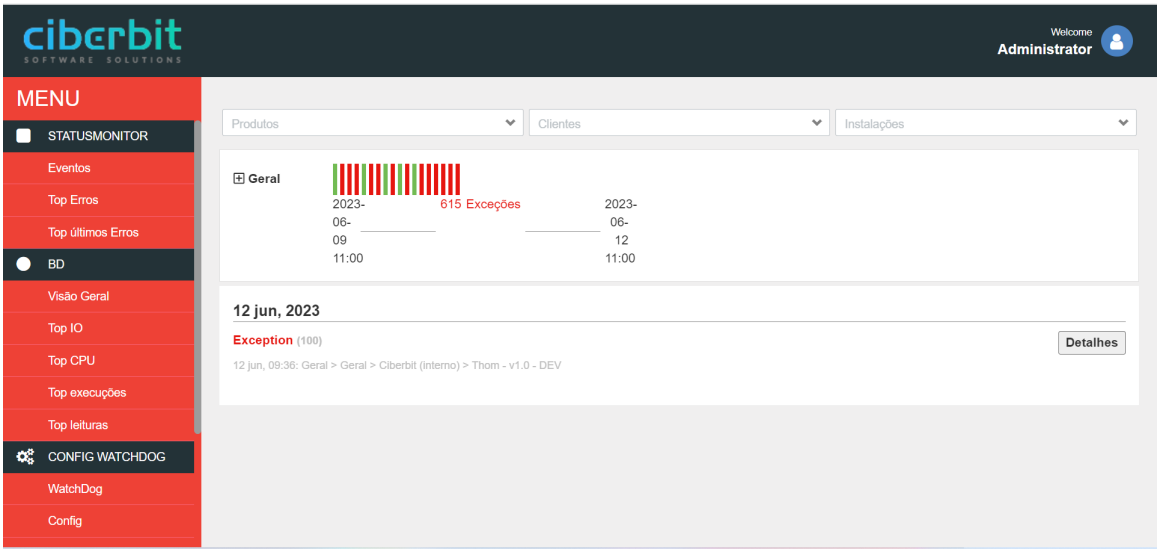

Figura 4.3: Página principal de um administrador.

Como se pode observar, esse menu corresponde a uma *dashboard* que apresenta algumas dados do I'MTHOM, referentes a exceções. Ela apresenta o número total de exceções que ocorreram nos últimos 3 dias anteriores à data atual, para além de uma lista contendo os

últimos 100 eventos espoletados pelo I'MTHOM. A todos os dados mencionados podem ser aplicados os filtros apresentados, no caso, por produto, cliente e instalação.

Para além disso, no canto superior direito da figura, é apresentado um ícone que, ao pressionado, apresenta ao utilizador algumas opções, como alterar a palavra-passe atual, e terminar a sessão.

Já o menu lateral encontra-se dividido em diversos grupos, cada um deles com diversas opções. Existem os seguintes 4: (i) StatusMonitor, (ii) BD, (iii) ConfigWatchDog, e (iv) Configuração.

O primeiro grupo permite visualizar diversos detalhes dos mais variados tipos de eventos reportados pelo I'MTHOM, como exceções e operações mal-sucedidas. Uma parte dessa lista é apresentada na figura 4.4.

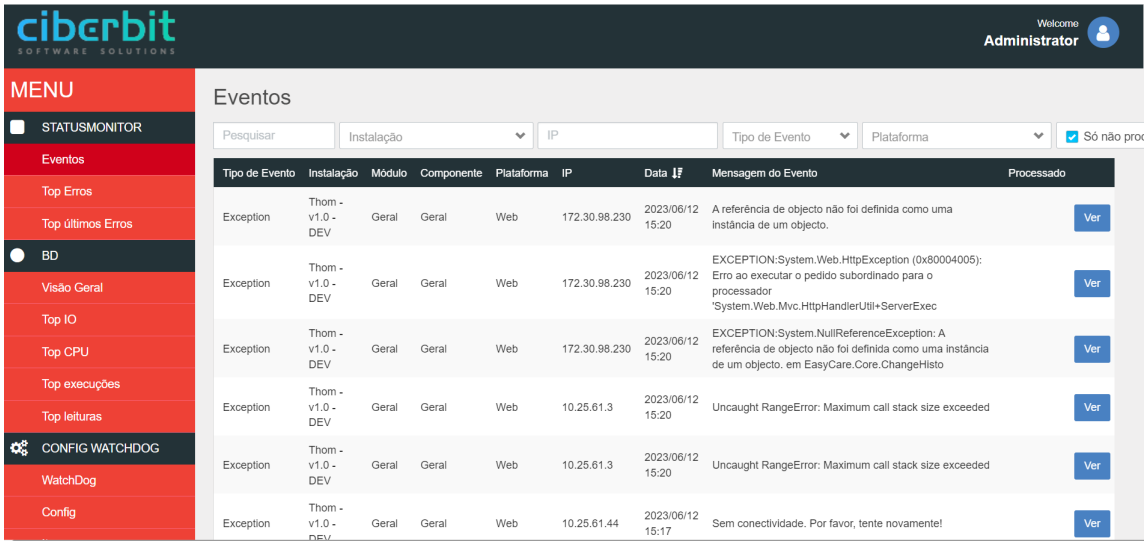

Figura 4.4: Lista de eventos.

A esta lista, que segue um esquema de paginação, é possível aplicar um conjunto de filtros, nomeadamente pesquisar por algum tipo de atributo caraterístico de cada evento, como IP, instalação, data de início, de fim, entre outros, assim como permite pesquisar manualmente por conteúdo presente nas mensagens dos eventos. Para além disto, também é possível exportar o conteúdo da lista para um ficheiro Excel.

Ainda ao nível do primeiro grupo, existem outras 2 opções que, atualmente, não se encontram implementadas, existindo a intenção de as implementar futuramente. Elas mostrariam listas contendo os eventos que manifestam algum tipo de erro, como exceções e operações mal-sucedidas, mais frequentes.

Relativamente ao segundo grupo, assim como acontece com as opções supracitadas, não existe nenhuma funcionalidade que permita que os utilizadores consigam ter acesso a esse tipo de informação. No caso, o objetivo seria apresentar diversas métricas referentes às diversas instâncias de bases de dados utilizadas pelas múltiplas instâncias do I'MTHOM, como listas de queries que mais CPU consomem, que mais tempo demoram a ser executadas, que mais vezes são executadas num determinado intervalo de tempo, que mais leituras efetuam, entre muitas outras. Este objetivo corresponde a um dos propostos na secção 1.3.

No que toca ao terceiro grupo, é possível consultar a lista de todos os serviços do tipo watchdog existentes, para além das respetivas configurações. Estes serviços são executados em servidores específicos, e, numa primeira fase, comunicam com o Status Monitor por forma a obter os respetivos dados de configuração, que também são listados neste grupo. Cada serviço pode ter diversas configurações, sendo que cada uma se encontra associada a ums determinada instalação, e possui diversas funções que permitem captar dados de uma API disponibilizada pelo próprio I'MTHOM. Esses dados, capturados pelo serviço, são enviados, posteriormente, para a API que o Status Monitor disponibiliza. Em suma, este tipo de serviço permite que sejam obtidas métricas sem que a aplicação monitorizada tenha que os enviar diretamente para o Status Monitor.

Por último, no quarto grupo, o utilizador consegue ter acesso a diversas listas de diversos objetos do sistema, como clientes, produtos, instalações, módulos e componentes, conceitos presentes no modelo de domínio apresentado na secção 4.1.1. A figura 4.5 apresenta uma dessas listagens, no caso a dos clientes, sendo que todas as outras possuem a mesma estrutura.

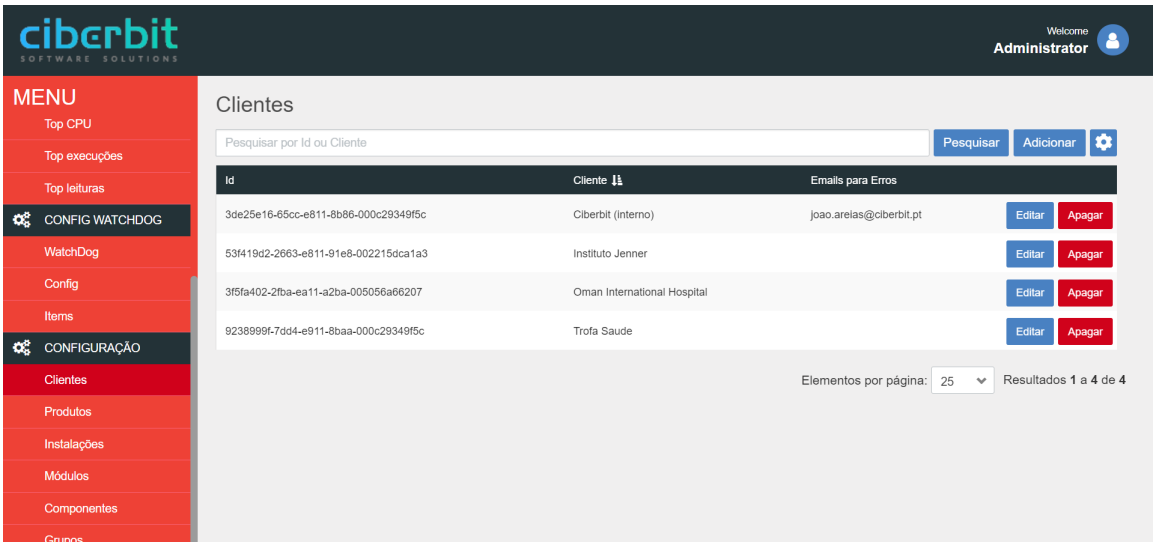

Figura 4.5: Lista de clientes.

Como é possível constatar, para além de ser possível realizar pesquisas manuais, também é possível editar ou eliminar os elementos presentes, assim como adicionar novos, sendo que todas essas alterações são persistidas na base de dados. Este processo também é aplicado às listagens presentes no terceiro grupo.

Com base no que foi mencionado previamente, é pode-se concluir que, atualmente, a aplicação não capta uma grande variedade de dados do I'MTHOM, existindo, portanto, uma necessidade em aumentar o grau de observabilidade dos dados, captando e apresentando ainda mais para os utilizadores, como mencionado na secção 1.3. Para além disso, como já mencionado, não é efetuado nenhum processamento ou análise automática dos dados que são enviados, algo que também necessita de ser corrigido com a introdução do conceito de alertas.

Capítulo 5

Desenho

Neste capítulo é realizada uma descrição do *design* arquitetural. Por forma a tornar isso possível, é utilizada uma combinação entre dois modelos de representação arquitetural: C4 (Brown 2023) e 4+1 (Kruchten 1995).

O modelo C4 é utilizado para descrever a arquitetura de um *software* ao longo de 4 níveis de abstração. Cada um destes níveis adota uma certa granularidade mais fina do que a do nível que o precede, apresentando mais detalhes de uma parte mais pequena do sistema, como se tratasse de um mapa do código. Uma vez que possui um conjunto bastante pequeno de níveis, o modelo torna-se fácil de usar. É importante ressaltar que não é necessário representar todos os níveis, mas sim apenas aqueles que agregam valor ao projeto (Brown 2023; Silva 2020).

Relativamente aos níveis de abstração, os mesmos encontram-se definidos da seguinte forma (Brown 2023; Silva 2020):

- **Nível 1 (contexto):** Nível mais alto de abstração, que descreve o sistema como um todo, incluindo, também, outros sistemas dos quais o principal depende, ou vice-versa;
- **Nível 2 (contentores):** Descreve diversas partes do sistema, também conhecidas como contentores, podendo corresponder a aplicações ou servidores para armazenar dados, fulcrais para o funcionamento do sistema como um todo, e que se interligam entre si;
- **Nível 3 (componentes):** Descreve os diversos componentes presentes em cada um dos contentores apresentados no nível anterior;
- **Nível 4 (código):** Nível mais baixo de abstração, que descreve o código ou partes mais pequenas de cada um dos componentes.

O modelo de vistas 4+1 tem como objetivo descrever o sistema a partir de um conjunto de diversas vistas complementares entre si, que permitem, por sua vez, a análise de diversos requisitos dos vários *stakeholders* do *software*, de forma separada, tais como utilizadores, administradores de sistemas, programadores, entre outros (Kruchten 1995; Silva 2020).

As vistas apresentadas neste modelo são as seguintes (Kruchten 1995; Silva 2020):

- **Vista lógica:** Relativa a conceitos de negócio e partes lógicas do sistema;
- **Vista de processos:** Relativa ao fluxo simplificado da comunicação entre as diversas partes ou interações que constituem o sistema, por forma a realizarem os processos de negócio;
- **Vista de implementação:** Referente à forma como o *software* se encontra organizado;

- **Vista física:** Apresenta os vários componentes físicos do sistema, ou seja, os componentes do *software* em *hardware*;
- **Vista de cenários:** Referente à associação de processos de negócio com autores que são capazes de os iniciar.

A combinação destes dois modelos torna possível a representação do sistema a partir de várias perspetivas, cada uma delas com diferentes níveis de detalhe.

Ao longo deste capítulo, podem ser encontrados diversos diagramas estruturados da forma supracitada, recorrendo a UML. Para este projeto, apenas algumas combinações de perspetivas e níveis de detalhe da arquitetura que se consideram relevantes são apresentadas. No caso, importa mencionar que o nível 4 do modelo C4 não é apresentado para nenhuma das vistas, por já não corresponder ao desenho arquitetural.

5.1 Nível 1 - Contexto

Nesta secção são apresentadas as vistas de nível 1 mais relevantes, que representam o sistema como um todo, num nível de granularidade mais grossa. Para isso, são apresentados todos os sistemas externos e utilizadores que comunicam ou interagem com o sistema. As vistas de implementação e física não são representadas na presente secção, devido ao facto de não serem relevantes para o nível de abstração em questão.

5.1.1 Vista lógica

A vista lógica do sistema encontra-se descrita na figura 5.1, por forma a responder aos casos de uso e requisitos propostos e enunciados no capítulo 3.

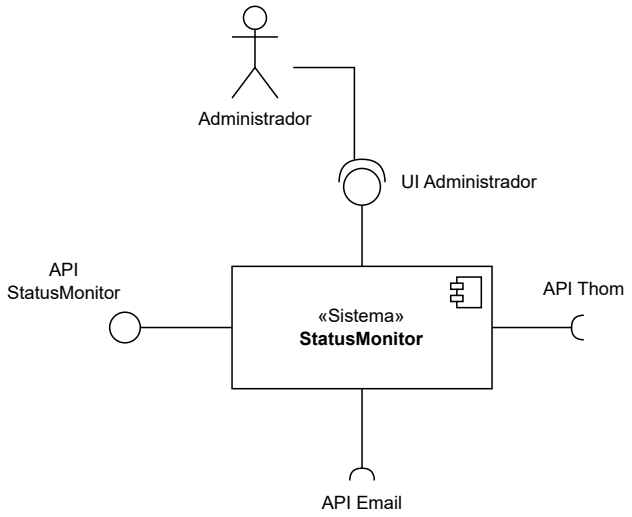

Figura 5.1: Vista lógica - nível 1.

Como mencionado na secção 3.2.8, o sistema deve expor uma *interface* de comunicação com o exterior. No caso, ela é representada no diagrama como API StatusMonitor. Esta API é responsável por receber diversos dados de telemetria de outros sistemas, por forma a armazenar esses dados e apresentá-los para os utilizadores devidos. No âmbito deste

projeto, como já mencionado, esse sistema externo corresponde ao I'MTHOM. Para além disto, também é possível o próprio sistema consumir uma API proveniente desse sistema externo.

O Status Monitor disponibiliza uma *User Interface* (UI), que permite que os utilizadores consigam aceder às diversas informações enunciadas na secção 3.1. Como também consta nessa secção, só existe um único tipo de utilizador no sistema.

Para além disto, nessa mesma secção, também é mencionada a necessidade de integrar serviços de email que permitam o envio de notificações para os utilizadores aquando da ocorrência de algum alerta.

5.1.2 Vista de processos

Nesta secção, alguns casos de uso, que foram identificados na secção 3.1, são apresentados através de diagramas de sequência, por forma a denotar as diversas interações possíveis que ocorrem no sistema. No caso, são apresentados 3 fluxos: (i) visualização de *dashboards* na plataforma (ii) criação de uma regra para espoletar um determinado alerta (iii) envio de notificação para utilizador aquando da ocorrência de um alerta.

A visualização de *dashboards* na plataforma, apresentado na figura 5.2, é referente à apresentação de diversos conteúdos respetivos à *dashboard* que está a ser acedida por um administrador. Cada tipo de utilizador tem acesso a um conjunto de dashboards que diferem de acordo com as suas permissões, sendo apresentada a *interface* mais adequada após a autenticação.

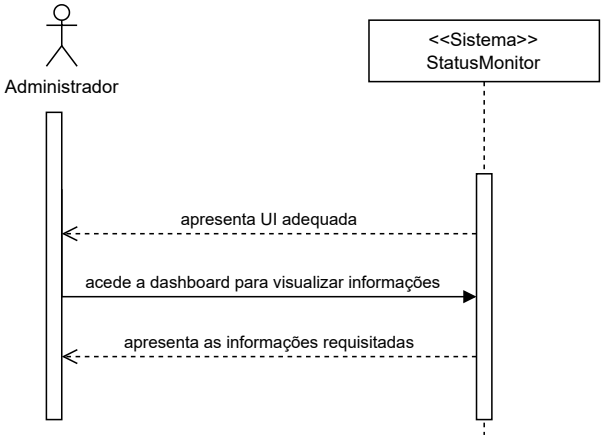

Figura 5.2: Visualização de dashboards na plataforma - vista de processos - nível 1.

Por forma a tornar possível espoletar alertas por parte do sistema, é necessário que existam regras que possam ser utilizadas aquando da captura dos dados, para que os mesmos possam ser devidamente validados. Para isto, é necessário que as mesmas sejam criadas, de acordo com o fluxo apresentado na figura 5.3.

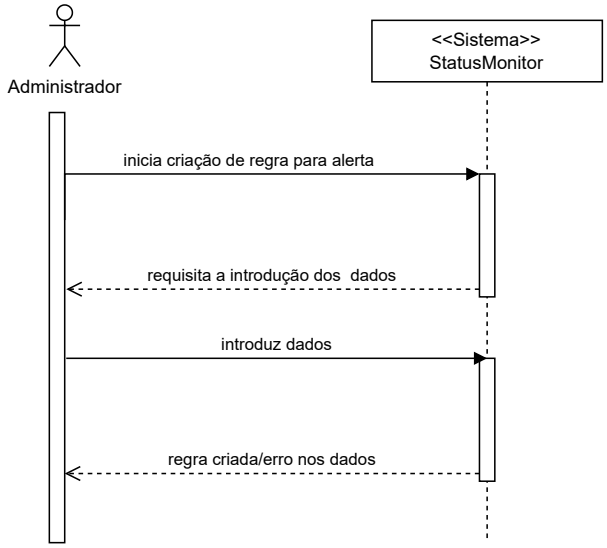

Figura 5.3: Criação de uma regra para espoletar alerta - vista de processos - nível 1.

Assim que um sistema deteta o cumprimento de alguma das regras criadas, ao validar os dados de telemetria provenientes do I'MTHOM, ele cria um alerta, que notifica todos os utilizadores registados para receberem notificação daquele mesmo alerta por email. Este fluxo encontra-se descrito na figura 5.4. É importante mencionar que esse fluxo apresenta o caso em que os dados são enviados diretamente pelo I'MTHOM para a API do Status Monitor, e não pelo WatchDog. Neste último caso, a diferença reside no facto dos dados serem enviados pelo WatchDog, e não diretamente pelo Status Monitor.

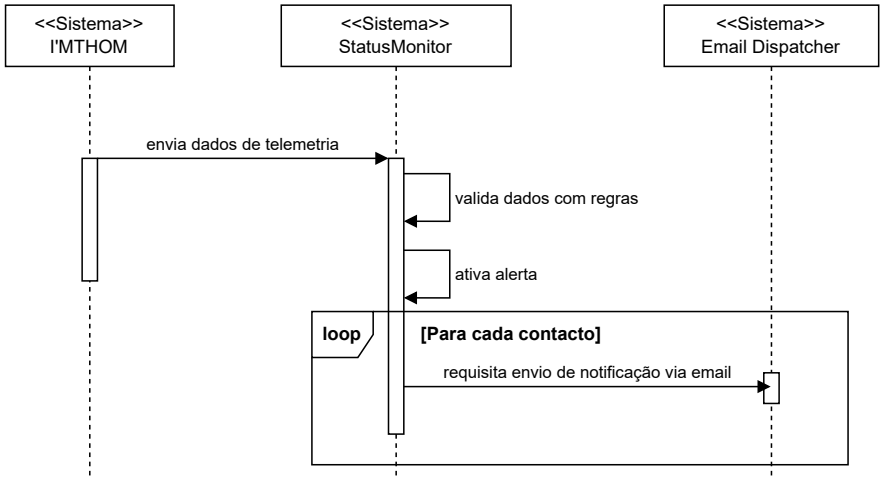

Figura 5.4: Envio de notificação para utilizador - vista de processos - nível 1.

5.2 Nível 2 - Contentores

A presente secção explora as diversas partes que constituem o Status Monitor, a partir de uma perspetiva arquitetural, apresentando, portanto, os diversos contentores que o compõem, num nível de granularidade mais fino que o da secção anterior.

5.2.1 Vista lógica

O diagrama que consta na figura 5.5 apresenta a vista lógica da Status Monitor, com os respetivos contentores. Um deles corresponde a uma base de dados, que não foi desenvolvida, mas apenas configurada para ser utilizada, por forma a permitir a persistência e fornecimento dos diversos dados capturados. Já os restantes dizem respeito a servidores de aplicação *web*.

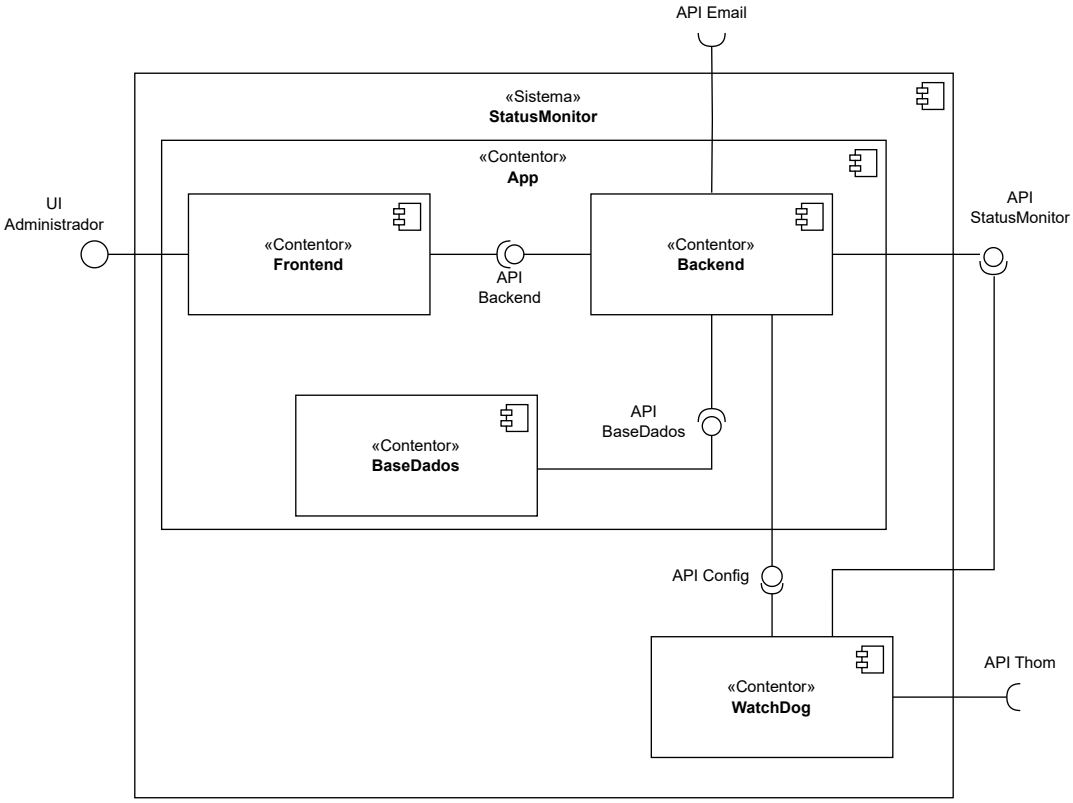

Figura 5.5: Vista lógica - nível 2.

Como se pode constatar, o sistema possui uma camada designada por App, que adota uma arquitetura em camadas, mais especificamente, 3 camadas, a saber (IBM 2023b):

- **Camada de apresentação:** Corresponde à *interface* do utilizador, onde o utilizador interage com o sistema. Tem como principal propósito apresentar dados para um utilizador, assim como capturar dados introduzidos pelo mesmo;
- **Camada de aplicação:** É referente à camada de negócio da aplicação, onde os dados provenientes da camada supracitada são processados e enviados para a camada de dados;

- **Camada de dados:** Diz respeito à camada de infraestrutura da aplicação, onde os dados são armazenados e fornecidos aquando de algum pedido.

Desta forma, pode-se concluir as seguintes caraterísticas dos contentores apresentados para essa camada:

- O contentor Frontend corresponde à camada de apresentação e expõe uma UI para os utilizadores poderem comunicar com o sistema;
- O contentor Backend corresponde à camada de aplicação e comunica com aplicações externas que disponibilizam dados de telemetria (neste caso, o I'MTHOM) através de uma API, consumindo-os. Para além disto, também expõe uma API que é consumida pelo contentor Frontend, e consome uma API responsável pelo envio de notificações via email. Este contentor comunica com um outro, que é detalhado posteriormente;
- Por último, o contentor BaseDados corresponde à camada de dados.

Para além dos contentores supracitados, existe um outro que comunica com a Backend, designado por WatchDog, que corresponde a um serviço que permite obter dados de telemetria provenientes do I'MTHOM. Como previamente mencionado na secção 4.2, aplicando ao contexto atual, o WatchDog comunica com a Backend por forma a obter informações de configuração e para enviar os dados que recebe do I'MTHOM, que expõe uma API que permite a obtenção desses mesmos dados.

5.2.2 Vista de processos

Na presente secção, alguns diagramas de sequência referentes a diversas funcionalidades do sistema são apresentados. Esses diagramas correspondem a versões mais detalhadas dos apresentados na secção 5.1.2.

A figura 5.6 apresenta uma versão mais detalhada da visualização de *dashboards* no sistema, por parte de um administrador.

Aquando do processo de autenticação de um utilizador na plataforma, é gerado um *token*, que possui um tempo de vida limitado, e que permite que o mesmo consiga aceder apenas às informações às quais o utilizador está autorizado a aceder, de acordo com o seu tipo, como se pode constatar. Todas as vezes que um utilizador aceder a uma opção da *interface*, como uma *dashboard* específica, o *token* é novamente validado, assim como as suas permissões, ao nível da Backend, por forma a garantir que o sistema é protegido contra ataques maliciosos.

O processo mais detalhado da criação de regras no sistema é representado na figura 5.7. Neste cenário, considera-se que o utilizador já se encontra autenticado, e com acesso à *interface* mais adequada para o mesmo, assim como um cenário de sucesso onde o mesmo consegue registar uma regra sem que nenhum erro tenha ocorrido. No caso, o utilizador introduz dados como nome, descrição, tipo de evento ao qual aplicar a regra, tipo de métrica, caso o tipo de evento escolhido tenha sido evento de métrica, servidor origem dos dados para os quais a regra será aplicada, condição para espoletar um alerta, grau de severidade da regra, e os diversos contactos para os quais o alerta gerado pela regra irá enviar uma notificação.

O diagrama presente na figura 5.8 apresenta a forma como um utilizador é notificado via email da ocorrência de um alerta, por cumprimento de alguma das regras previamente criadas. Esse alerta é criado pelo próprio sistema, armazenado numa base de dados, para uso

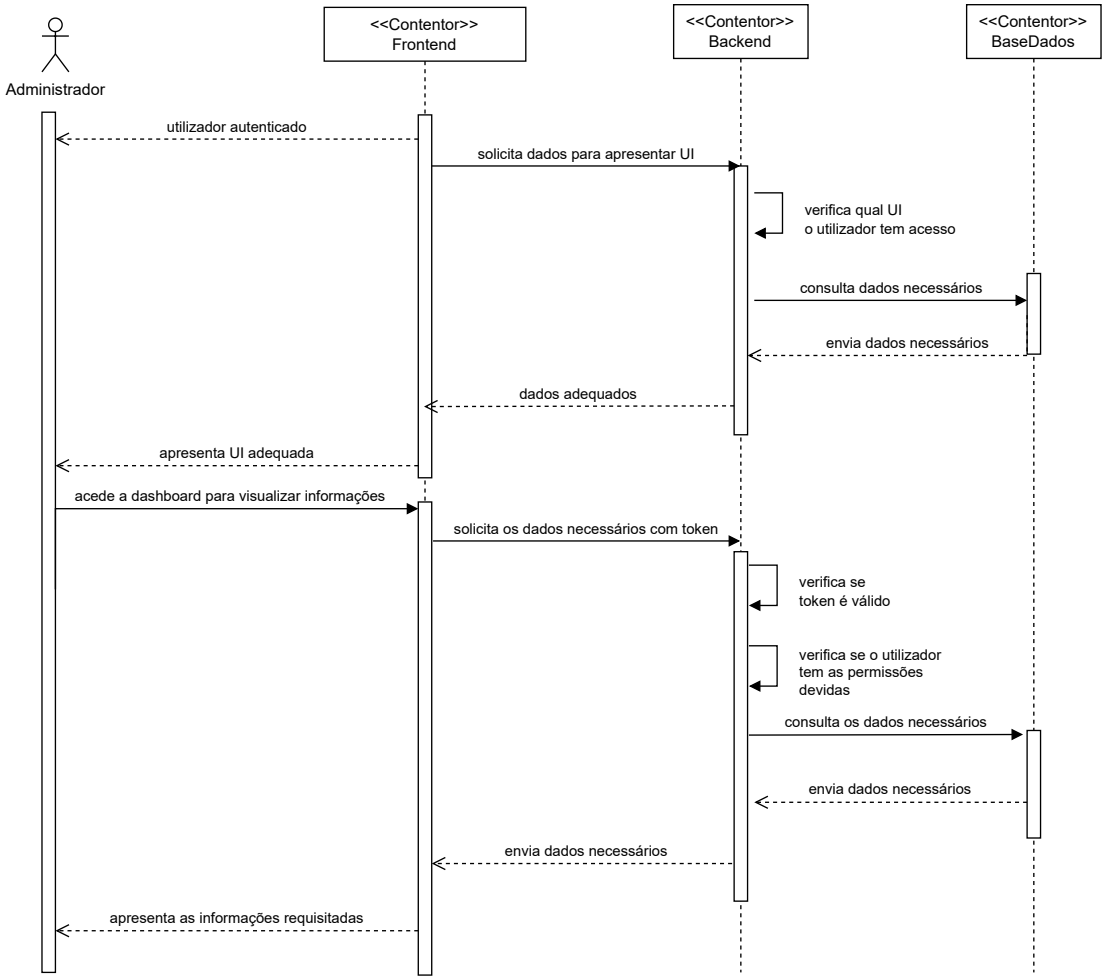

Figura 5.6: Visualização de *dashboards* na plataforma - vista de processos - nível 2.

posterior, e enviado sob a forma de um email para os utilizadores cujos contactos estão associados àquele mesmo alerta.

5.2.3 Vista de implementação

Cada um dos contentores mencionados na secção 5.2.1 é desenvolvido dentro de uma mesma pasta, designada project. O diagrama representado na figura 5.9 apresenta a forma como eles se encontram distribuídos dentro dessa mesma pasta.

Como se pode constatar, apenas são representadas duas pastas dentro da principal. A primeira, designada App, contém outras duas pastas, uma referente ao componente de frontend, e outra ao de backend, enquanto que a WatchDog contém todos os referentes ao respetivo contentor.

Todas as tecnologias preconizadas são exploradas na secção 6.1.

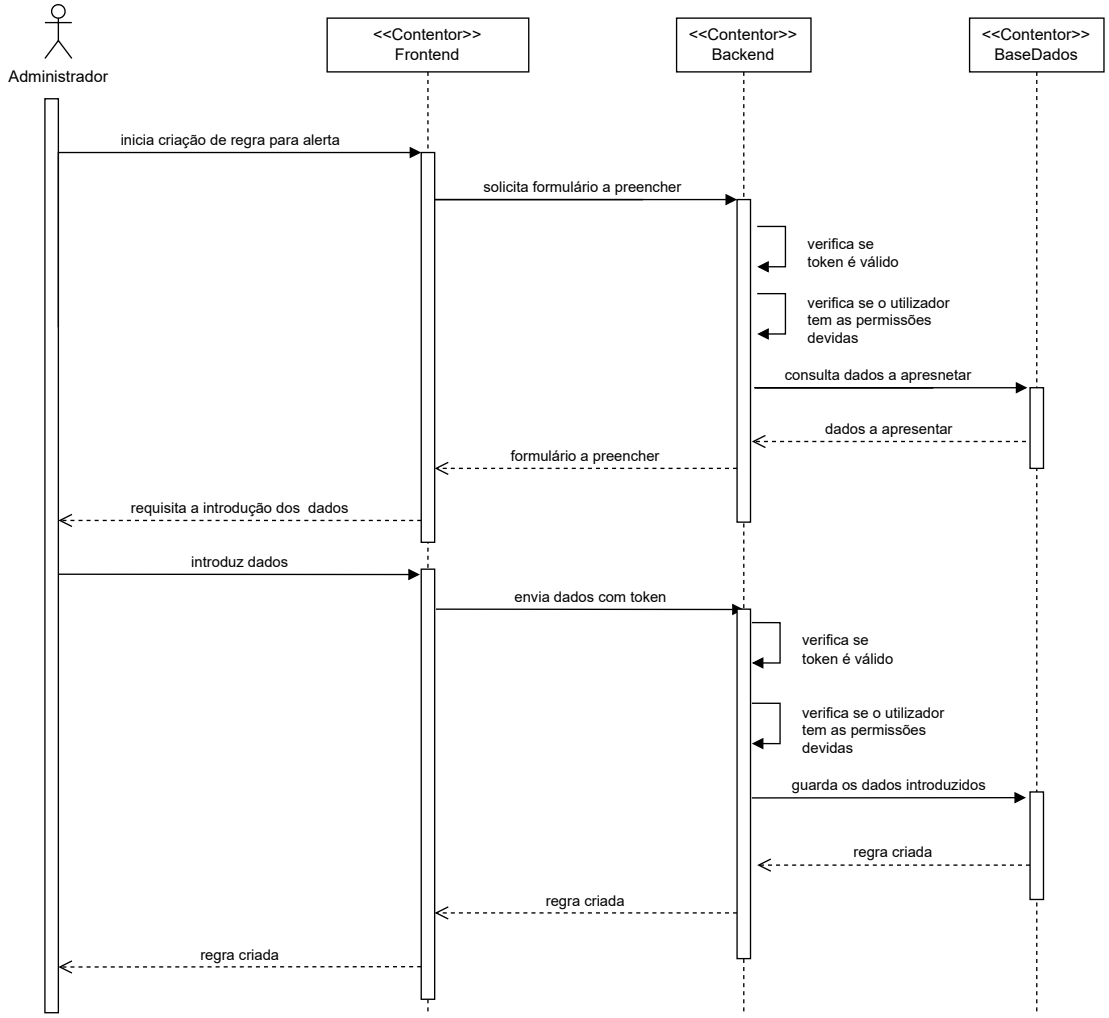

Figura 5.7: Criação de uma regra para espoletar alerta - vista de processos - nível 2.

5.2.4 Vista física

Nesta secção é apresentada a vista física do sistema, que apresenta os componentes físicos do mesmo, como se pode constatar na figura 5.10. Cada contentor declarado na secção 5.2.1 corresponde a um contentor dentro de um determinado dispositivo.

Os 3 dispositivos presentes correspondem: (i) à maquina do utilizador (*localhost*), que disponibiliza a *interface* da aplicação através de um *browser* que pode ser acedido pelo mesmo através de um computador ou dispositivo móvel, (ii) a um servidor Windows disponibilizado pela própria empresa, e (iii) a um servidor *Structured Query Language* (SQL), onde a base de dados se encontra hospedada.

Todas as comunicações entre os utilizadores e a plataforma são seguras e conduzidas por um servidor *Internet Information Services* (IIS), implantado no servidor da empresa, que também executa o contentor da Backend, uma vez que a Status Monitor atual encontra-se implementada dessa forma, e o contentor do WatchDog.

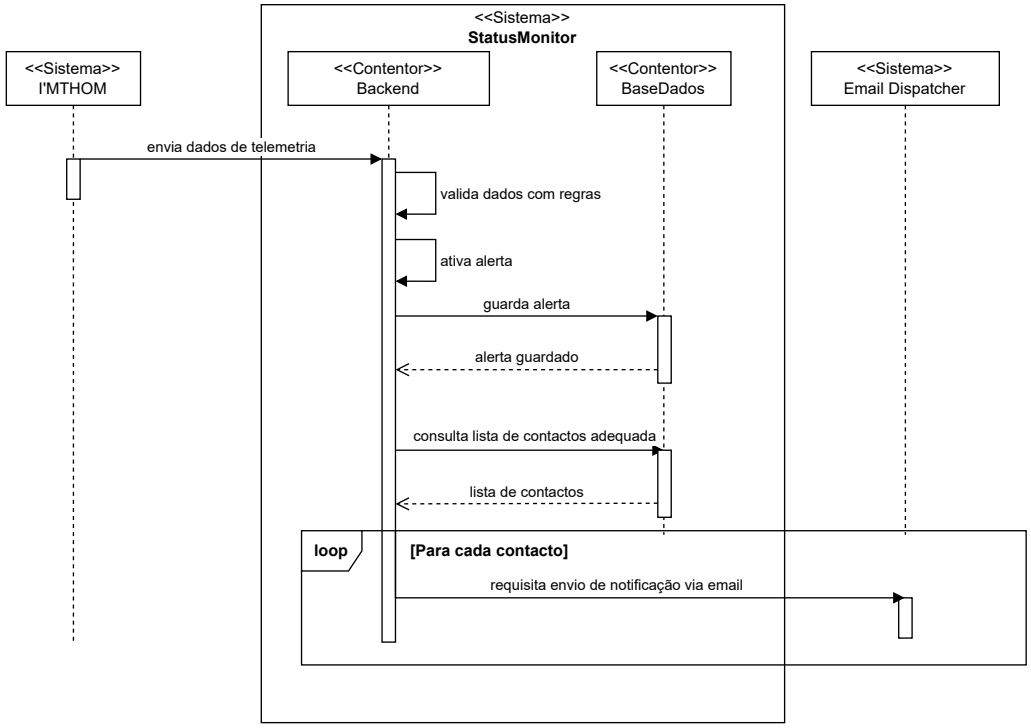

Figura 5.8: Envio de notificação para utilizador - vista de processos - nível 2.

Assim como na secção anterior, todas as tecnologias mencionadas no diagrama são exploradas com maior detalhe na secção 6.1.

5.3 Nível 3 - Componentes

A presente secção descreve os componentes internos que fazem parte de cada um dos contentores relevantes que foram identificados anteriormente, com responsabilidades bem definidas. Estes componentes, assim como as suas dependências, encontram-se detalhadas, assim como nas secções anteriores, sob a forma de diagramas. A vista física não será descrita, uma vez que a apresentada na secção 5.2.4 já contém todos os detalhes relevantes.

5.3.1 Vista lógica

Ao nível desta vista, o contentor WatchDog não é abordado, uma vez que corresponde a um simples serviço. A arquitetura utilizada nos restantes contentores desenvolvidos é baseada no padrão *Model-View-Controller* (MVC). Este padrão ajuda a atingir a separação de camadas, ao dividir uma aplicação em 3 grupos de componentes principais (DEI-ISEP 2021; Microsoft 2022b):

- **Modelo:** Corresponde à parte da aplicação que é responsável pela implementação da lógica da mesma, das regras de negócio, e do modelo de domínio;
- **Vista:** Diz respeito ao componente que apresentam a *interface* da aplicação para um determinado utilizador. Normalmente ela é criada a partir das informações presentes no modelo;

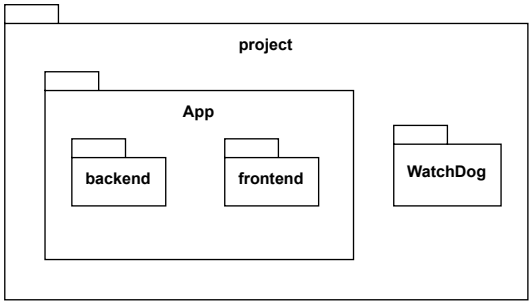

Figura 5.9: Vista de implementação - nível 2.

- **Controlador:** É referente ao componente que lida com a interação com o utilizador, processando e respondendo aos pedidos enviados pelo mesmo. Para além disto, ele trabalha com o modelo e seleciona uma vista para ser apresentada na *interface* do utilizador.

No âmbito deste projeto, este padrão foi aplicado tanto ao contentor Frontend como o contentor Backend, de uma forma complementar, isto é, uma parte dele é aplicada ao primeiro (no caso, a Vista), enquanto que o restante (Modelo e Controlador) é aplicado ao segundo. O comportamento da Frontend pode ser observado na figura 5.11. Já o comportamento da Backend encontra-se presente na figura 5.12

A partir dos diagramas apresentados, é possível identificar, para cada contentor, um conjunto de componentes. Começando pelo contentor referente à Frontend, apenas existem 2 componentes, com a seguinte responsabilidade:

- **View:** Responsável por apresentar os dados para um determinado utilizador da aplicação, e por capturar os introduzidos pelo mesmo, retornando respostas apropriadas;
- **Service:** Responsável por executar diversas ações, a pedido da View. Essas ações podem consistir em verificar quais os acessos do utilizador em questão tem acesso, validar o *token*, e comunicar com a API da Backend, através de pedidos *Hyper Text Transfer Protocol* (HTTP).

Em relação à Backend, as responsabilidades dos seus componentes são as seguintes:

- **Controller:** Responsável por tratar dos pedidos enviados pelas APIs disponibilizadas, e enviar as devidas respostas;
- **Service:** Responsável por executar diversas processos de negócio, a pedido do Controller. Estes processos podem consistir em verificar e validar as permissões de um utilizador e respetivo *token*, e tratar dos dados recebidos, reencaminhando-os para os componentes apropriados para os persistir, ou até mesmo os capturar;
- **Model:** Responsável por representar as classes de domínio e lógica de negócio, contendo dados relevantes para apresentação das vistas para os utilizadores;
- **Repository:** Responsável por permitir que os dados possam ser enviados para a base de dados, interagindo com componentes adequados para o efeito;
- **Driver:** Responsável por comunicar com a base de dados;

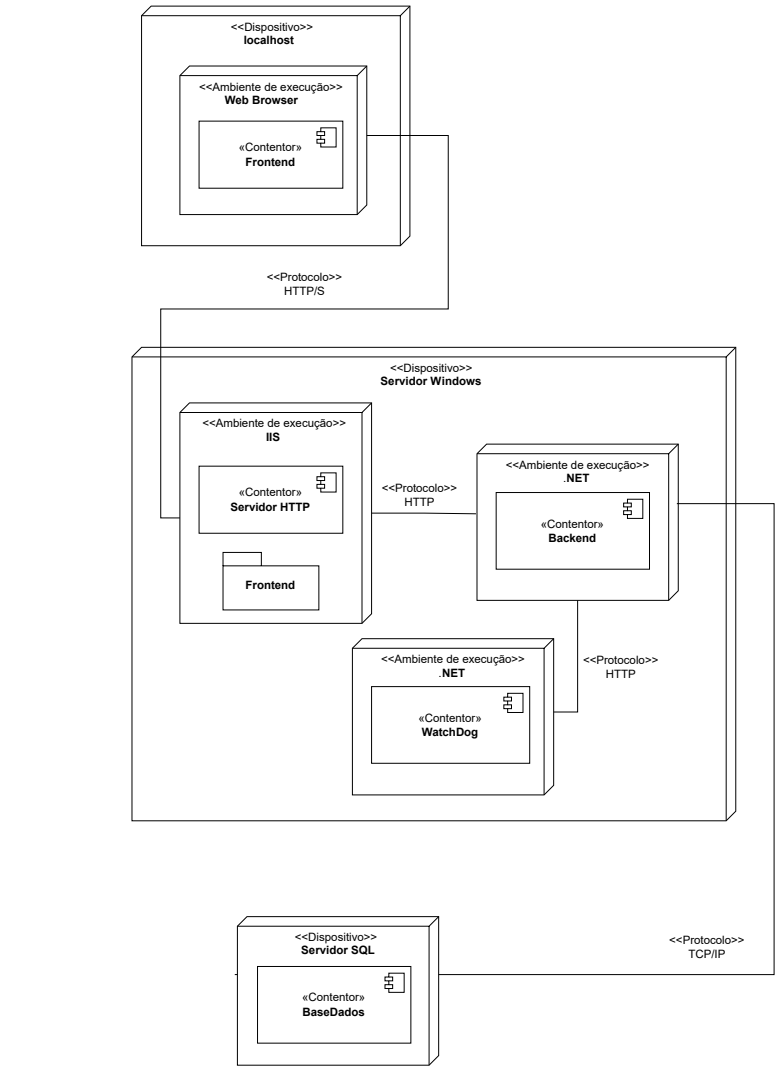

Figura 5.10: Vista física - nível 2.

- **Data Model:** Responsável por representar os dados num formato semelhante aos da base de dados.

5.3.2 Vista de processos

Nesta secção são detalhados alguns processos considerados relevantes, através de diagramas de sequência, por forma a fazer com que o leitor se familiarize com as diversas interações que ocorrem entre os componentes que constituem um determinado contentor.

Os processos que são avaliados correspondem a versões ainda mais detalhadas dos apresentados na secção 5.2.2.

Os seguintes diagramas, que se encontram descritos nas figuras 5.13 e 5.14, detalham ainda mais o processo de visualização de uma determinada dashboard apresentada pelo sistema desenvolvido. No caso, o primeiro mencionado detalha as ações que ocorrem ao nível do contentor da frontend, enquanto que o segundo detalha as da backend.

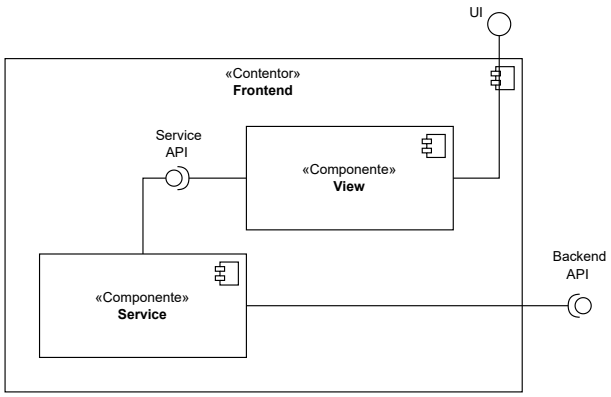

Figura 5.11: Vista lógica - Frontend - nível 3.

Como apresentado nos diagramas, que apresentam um cenário de sucesso do processo:

- Se um utilizador tiver um *token* válido e permissões de acesso a uma determinada *dashboard*, é apresentado para o mesmo o devido conteúdo. Caso contrário, é utilizado um modelo diferente, específico para apresentar para o utilizador uma *interface* que denuncia as presentes falhas;
- Os dados provenientes da classe Driver são do tipo Data Model, sendo, numa fase posterior, convertidos para dados do Model, para serem apresentados para o utilizador.

Os próximos diagramas, apresentados nas figuras 5.15 e 5.16, descrevem, de uma forma mais detalhada, o processo de criação de uma regra no sistema implementado. Este processo corresponde ao cenário de sucesso, onde uma regra é criada de forma bem sucedida pelo utilizador devido. Assim como nos diagramas anteriores, o primeiro mencionado apresentam as ações realizadas ao nível da frontend, enquanto que o segundo da backend.

Por último, o diagrama que detalha o processo em que um utilizador, ou grupo de utilizadores, é notificado através dos emails associados, do espoletamento de um alerta, é apresentado na figura 5.17. Para este caso, optou-se por representar o processo após um alerta ser gerado.

5.3.3 Vista de implementação

O seguinte diagrama, que corresponde à figura 5.18, descreve a vista de implementação deste nível, analisando, com mais detalhe, a organização das pastas referentes à backend e frontend apresentadas na secção 5.3.1. Por corresponder a um serviço bastante simples, a pasta WatchDog não é apresentada.

Uma vez que algumas pastas possuem nomes diferentes dos componentes que representam, a seguinte lista apresentam um mapeamento dessas pastas com os componentes da vista lógica:

- A pasta views da corresponde ao componente view da frontend;
- A pasta services dentro da pasta frontend corresponde ao componente service da frontend;
- A pasta modules corresponde ao componente controller da backend;

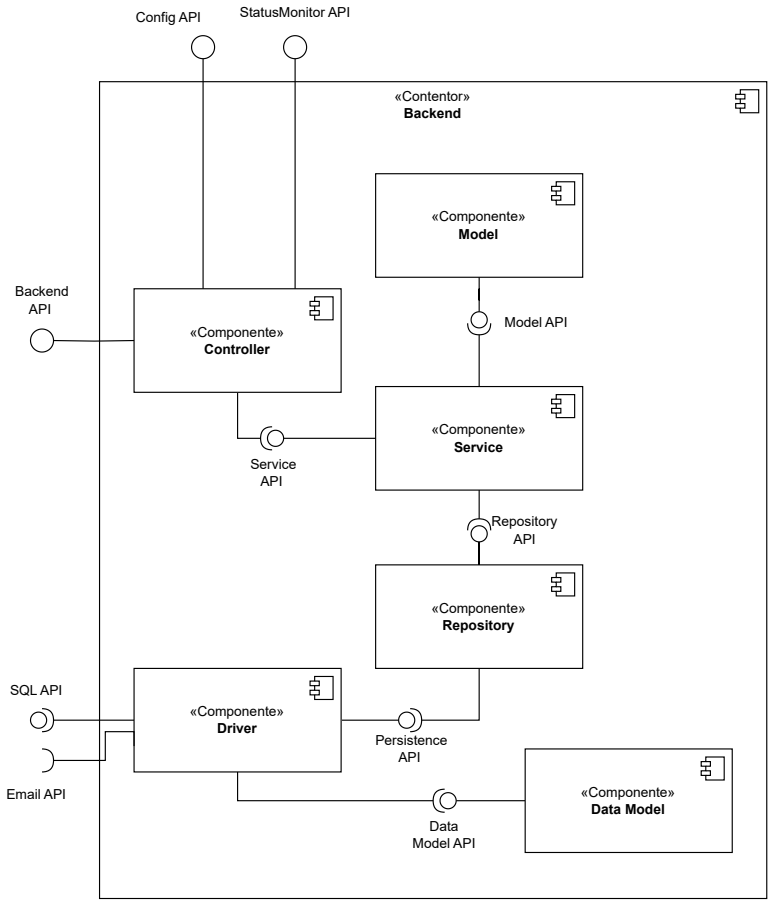

Figura 5.12: Vista lógica - Backend - nível 3.

- A pasta services dentro da pasta core corresponde ao componentes service da backend;
- A pasta repositories corresponde ao componente repository da backend;
- A pasta model corresponde ao componente model da backend;
- A pasta type corresponde ao componente data model da backend.

5.4 Sumário

Esta secção discutiu a arquitetura utilizada para a plataforma. Foram apresentados diversos diagramas em diferentes níveis de granularidade. No capítulo seguinte, a implementação da solução é descrita, com base no que foi apresentado neste capítulo.

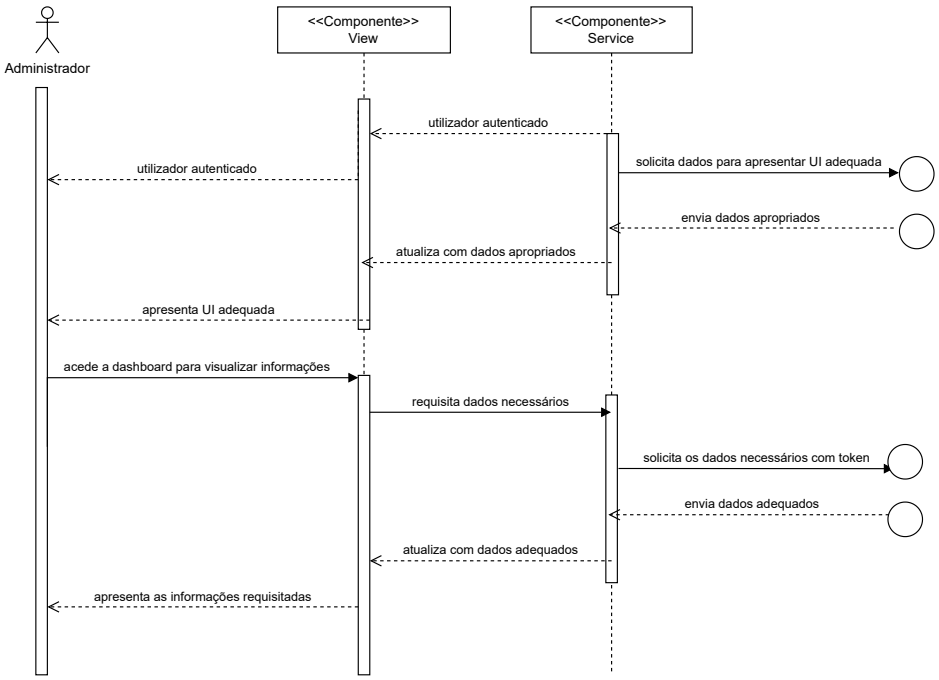

Figura 5.13: Visualização de dashboards na plataforma - vista de processos - nível 3 - frontend.

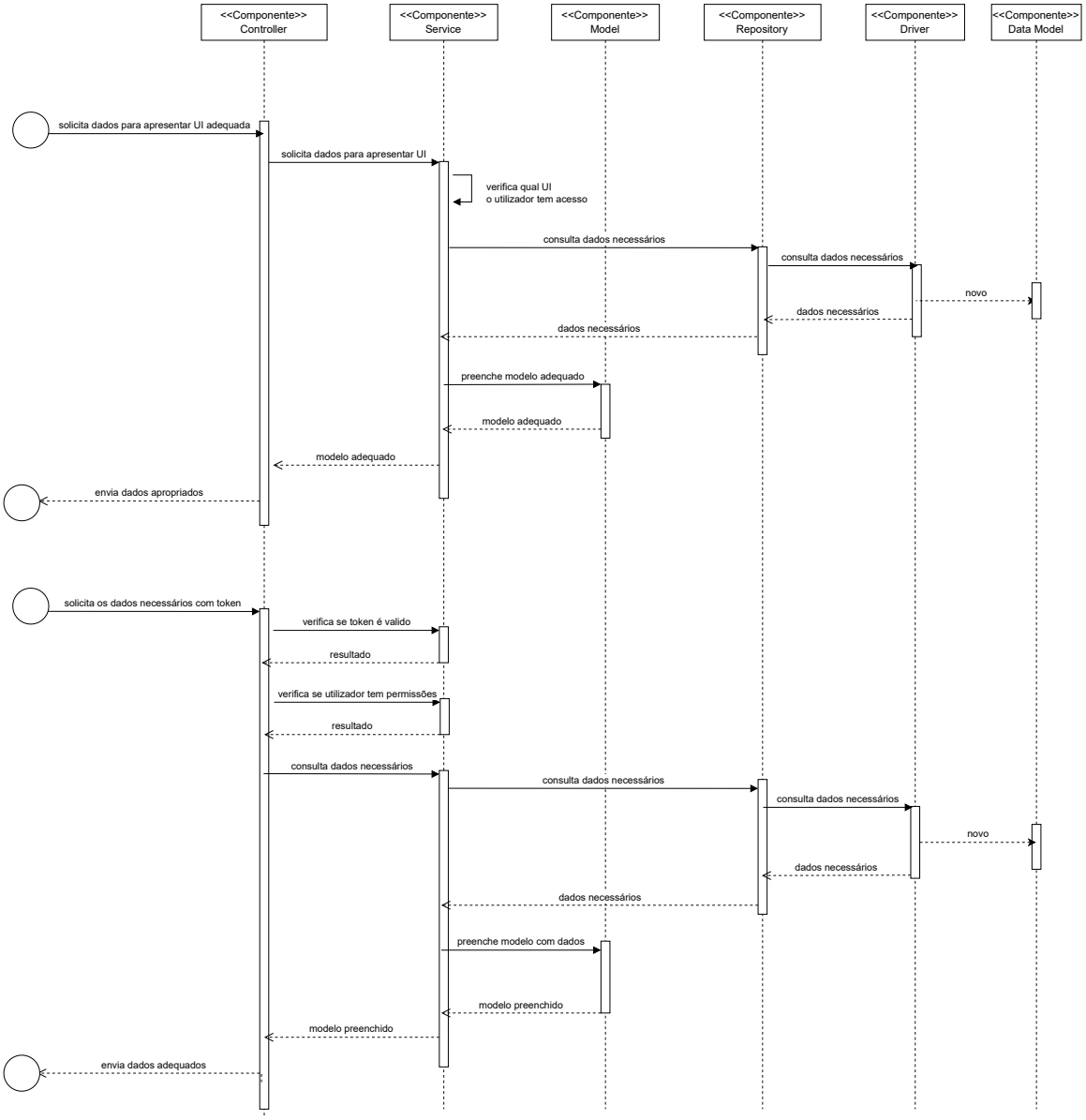

Figura 5.14: Visualização de dashboards na plataforma - vista de processos - nível 3 - backend.

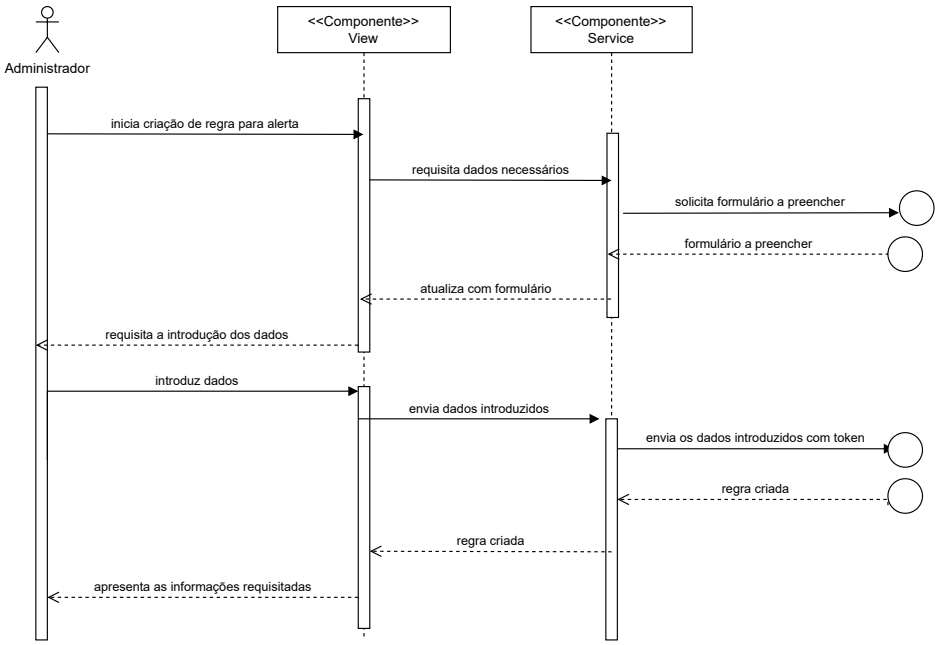

Figura 5.15: Criação de uma regra para espoletar alerta - vista de processos - nível 3 - frontend.

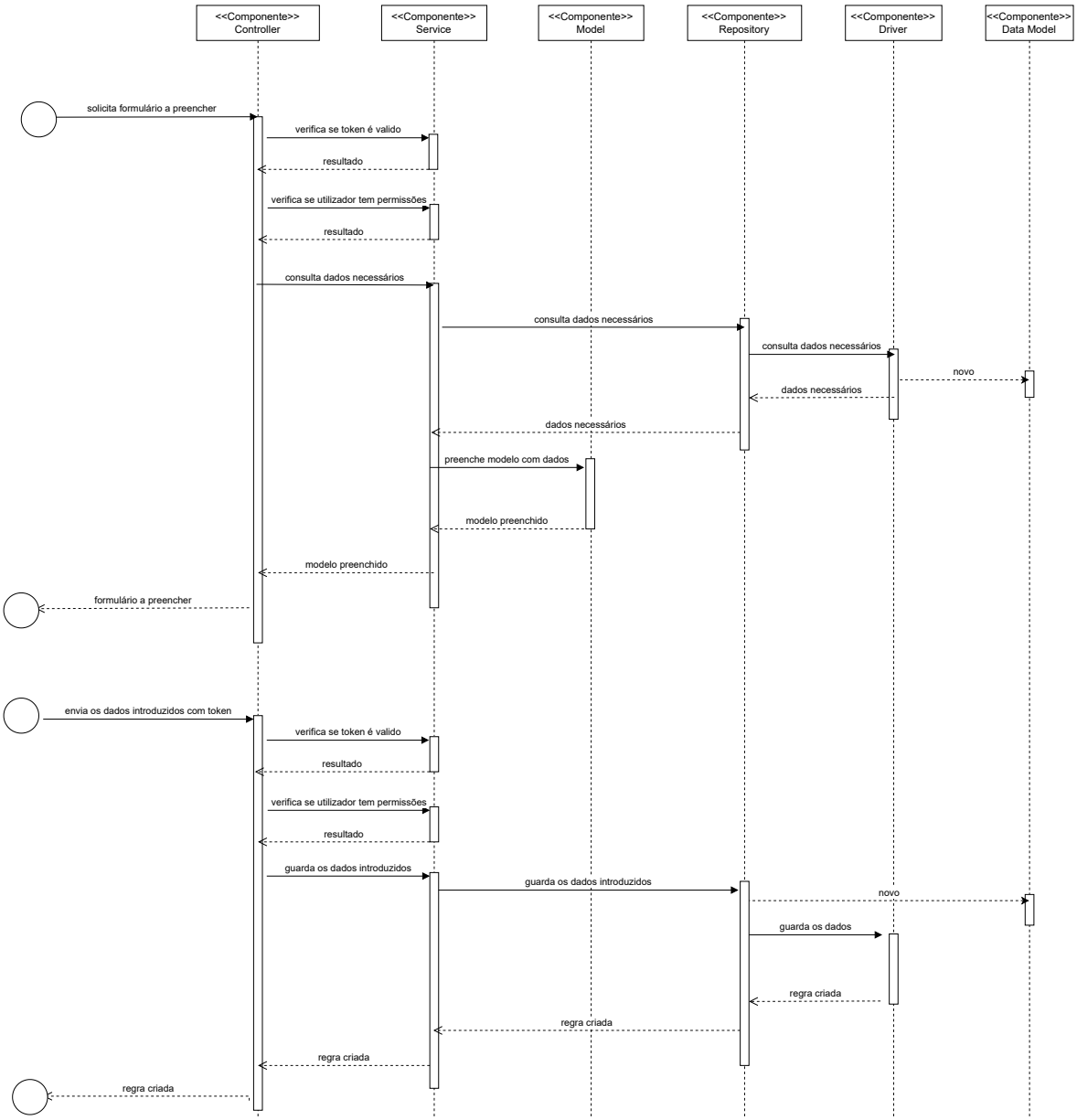

Figura 5.16: Criação de uma regra para espoletar alerta - vista de processos - nível 3 - backend.

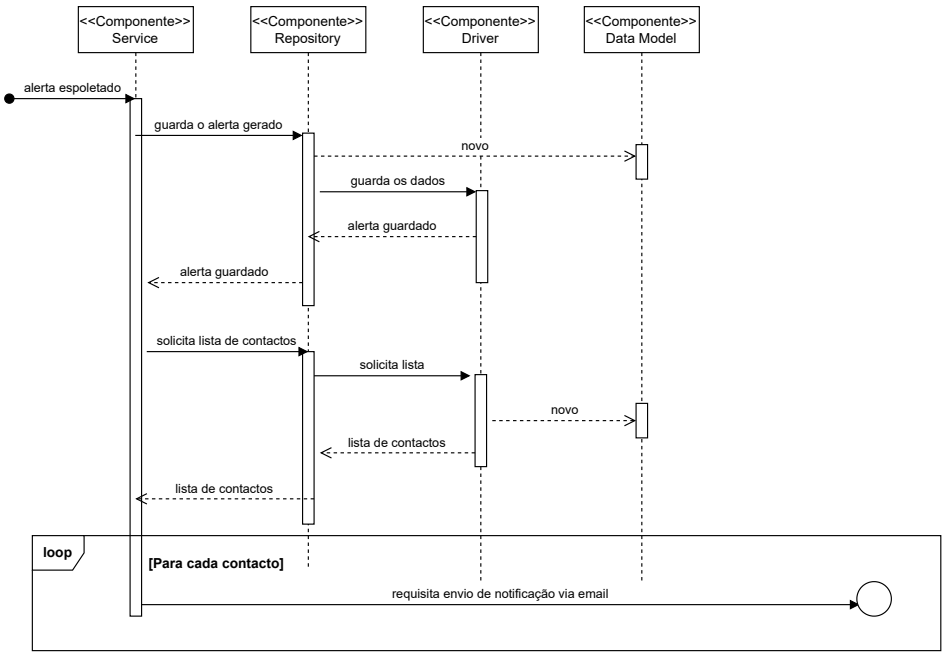

Figura 5.17: Envio de notificação para utilizador - vista de processos - nível 3.

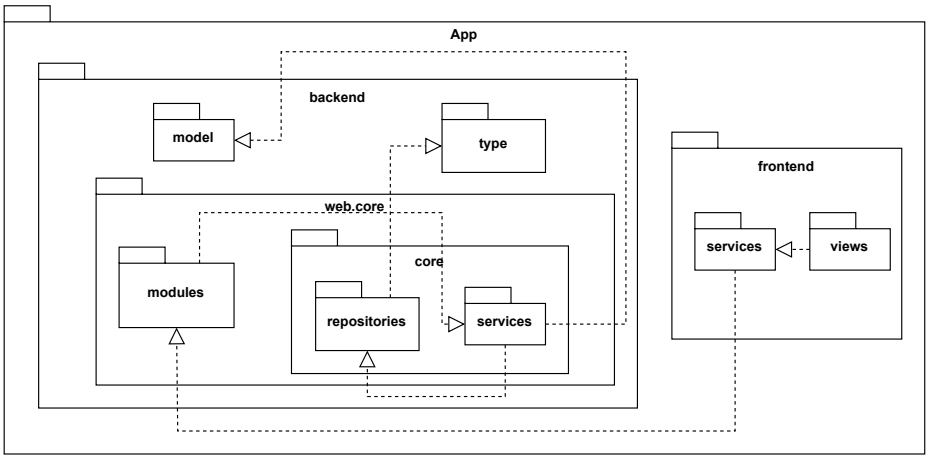

Figura 5.18: Vista de implementação - nível 3.

Capítulo 6

Implementação

Este capítulo detalha a implementação da solução, com base na arquitetura apresentada anteriormente. Num primeiro momento, são apresentadas as decisões técnicas que foram tomadas, detalhando todas as tecnologias utilizadas na implementação. Esta secção é seguida por uma descrição técnica de diversos elementos que constituem o *software* desenvolvido. Por último, é explicada e demonstrada a metodologia utilizada para testar o projeto.

6.1 Decisões técnicas

Esta secção apresenta e descreve as decisões que foram tomadas aquando do desenvolvimento da solução proposta. Uma vez que o projeto consiste na modificação de um já existente, o trabalho sofreu algumas restrições provenientes do mesmo, sendo que algumas decisões foram tomadas com base no que já tinha sido implementado anteriormente. No âmbito deste projeto, as decisões técnicas mais relevantes encontram-se presentes na seguinte lista, sendo que cada uma delas é analisada com maior detalhe nas subsecções seguintes:

- Tecnologias Backend utilizadas;
- Tecnologias Frontend utilizadas;
- Servidor web utilizado e reverse proxy;
- Base de dados utilizada.

6.1.1 Tecnologias Backend utilizadas

Esta subseção apresenta uma breve descrição e justificação das tecnologias utilizadas na parte referente à backend do projeto.

Por forma a construir a solução, foi utilizada a tecnologia .NET, que consiste numa plataforma gratuita, multiplataforma e *open-source*, utilizada para a implementação de diversos tipos de aplicações, sendo possível o uso de múltiplas linguagens de programação, como C#, F#, ou Visual Basic, e bibliotecas aptas para desenvolvimento *web*, *mobile*, *desktop*, entre muitos outros (Microsoft 2023c).

Esta tecnologia foi adotada devido a um conjunto de motivos, nomeadamente por:

- Ser adotada na versão inicial do Status Monitor;
- Ser uma tecnologia segura e confiável, uma vez que é oficialmente suportada pela Microsoft;

- Possuir um largo ecossistema, com cerca de 5.000.000 de desenvolvedores .NET espalhados pelo mundo.

Relativamente à linguagem de programação, a escolhida foi C#, uma vez que corresponde à mesma utilizada na versão inicial do Status Monitor, para além de já existir alguma familiaridade com a mesma por parte do autor.

Existem diversas implementações da tecnologia .NET, como .NET Framework, .NET Core, *Universal Windows Platform* (UWP) , Mono, entre outras. A utilizada é a .NET Framework, que consiste numa biblioteca de classes, que fornece um conjunto de APIs e tipos, como strings, datas e números, por exemplo, para várias funcionalidades, como ler e escrever em ficheiros, conectar a bases de dados, entre outras (Microsoft 2023b).

Assim como no caso da linguagem de programação, esta implementação foi escolhida por ser a utilizada na versão inicial já construída do projeto. Contudo, no futuro, existem intenções, por parte da empresa, de adotar uma outra, nomeadamente o .NET Core.

Para além do que já foi mencionado, existem diversos componentes que estendem a plataforma .NET, permitindo a criação de tipos específicos de aplicações. Um desses componentes consiste no ASP.NET, uma *framework* que fornece diversas ferramentas e bibliotecas específicas para o desenvolvimento de aplicações *web*, incluindo páginas *web*, APIs REST, microserviços, e até mesmo hubs que fornecem conteúdos em tempo-real para os utilizadores conectados (Microsoft 2023d).

Esta *framework* disponibiliza 3 outras *frameworks* que permitem a criação de aplicações *web*: Web Forms, ASP.NET MVC, e ASP.NET Web Pages, sendo que cada uma segue um estilo de desenvolvimento diferente. A escolhida para este projeto foi a ASP.NET MVC, uma vez que, assim como nas escolhas anteriores, é adotada na versão inicial do Status Monitor, e porque adota o padrão arquitetural definido no capítulo anterior, permitindo, dessa forma, uma boa separação da aplicação em camadas (Microsoft 2022a).

Para além disto, a empresa desenvolveu uma *framework* genérica, que permite um conjunto de diversas funcionalidades, recorrendo á *framework* previamente mencionada, como acessos à base de dados, comunicação via *Short Message Service* (SMS), e até mesmo gerar diversas estruturas que podem ser apresentados para o cliente, como tabelas que apresentam diversos dados, com diversos mecanismos de filtragem incorporados, que permitem a adição, edição, ou remoção desses mesmos dados. Ela tem como propósito simplificar a implementação de algumas funcionalidades e, por isso, também foi utilizada no desenvolvimento da backend. Esta *framework* é utilizada ao longo da construção da solução.

6.1.2 Tecnologias Frontend utilizadas

Esta subseção apresenta uma breve descrição e justificação das tecnologias utilizadas na parte referente à frontend do projeto.

Para a parte referente à frontend da aplicação, uma das *frameworks* escolhida corresponde a uma outra mencionada anteriormente e que também é utilizada na backend, no caso, a ASP.NET MVC, que, assim como previamente mencionado, permite a criação de páginas *web* e contribui para o desenvolvimento de aplicações *web*.

Uma das funcionalidades que a biblioteca fornece consiste na criação de páginas dinâmicas recorrendo a C#, HTML, CSS e JavaScript, com uma sintaxe para este tipo de páginas *web*, designada por Razor. Ela permite conciliar estas diversas linguagens de programação num só

ficheiro, onde o código escrito em C# é executado no servidor e o HTML resultante enviado para o cliente, onde, no seu lado, o código JavaScript é executado (Microsoft 2023d).

Para além da *framework* supracitada, também é utilizada a da própria Ciberbit, assim como na backend, para a apresentação de diversos elementos visuais nas *dashboards*, como tabelas. Contudo, ela não permite apresentar alguns tipos de elementos, como gráficos, por exemplo. Por causa disso, foi escolhida a *framework* DevExtreme para tornar possível esse tipo de apresentação. Ela possui um conjunto de diversos componentes para interfaces que podem ser utilizados com JQuery, Angular, AngularJS, Knockout, ASP.NET Core, e até mesmo ASP.NET MVC, incluindo diversos tipos de gráficos interativos, entre outros elementos (DevExtreme 2023).

Esta biblioteca foi escolhida devido ao facto da mesma já se encontrar presente na versão inicial do projeto, mesmo não sendo utilizada nessa fase, e por recomendação do supervisor da empresa, não tendo sido realizada nenhuma análise de outras possíveis soluções.

6.1.3 Servidor web utilizado e reverse proxy

Por forma a tornar possível a hospedagem das páginas *web* apresentadas para os utilizadores, e redirecionamento dos pedidos realizados pelos mesmos para a backend, a versão inicial do Status Monitor faz uso do IIS, que consiste num servidor *web* flexível, seguro, e gerenciável, desenvolvido pela Microsoft que permite responder a essa necessidade. Por esse motivo, este servidor é utilizado no projeto.

6.1.4 Base de dados utilizada

Esta secção apresenta a forma como as informações são armazenadas e geridas ao longo de todo o sistema desenvolvido.

Um *Database Management System* (DBMS), traduzido como sistema de gestão de bases de dados, consiste num sistema que permite aos seus utilizadores a criação e gestão de uma base de dados. Ele facilita os processos de definir, construir, manipular, e partilhar dados. Uma determinada aplicação aceder à base de dados ao enviar pedidos para o DBMS, recorrendo a *queries* ou transações. Outra função bastante importante que é fornecida por este tipo de sistemas consiste na proteção e manutenção da base de dados ao longo de um grande período de tempo (Elmasri e Navathe 2017).

Estes sistemas podem ser classificados de acordo com o modelo de dados utilizado, que consiste numa estrutura de organização de dados (Elmasri e Navathe 2017). Existem alguns, como (i) modelo relacional e (ii) modelo baseado em documentos, que são analisados no presente documento.

6.1.4.1 Modelo relacional

O modelo relacional representa a base de dados como sendo um "conjunto de relações", onde cada relação se assemelha a uma tabela de valores (Elmasri e Navathe 2017). Ele baseia-se num esquema para definir de forma exata e sem ambiguidade todas as relações de interesse para os utilizadores (Zaniolo e Meklanoff 1981).

Quase todas as bases de dados relacionais utilizam SQL para interagirem com os dados armazenados. Esta linguagem, ao contrário de outras, como Python ou C, que são genéricas por natureza, permitindo a implementação de diversas tarefas, das mais simples às mais

complexas, possui um propósito mais específico, no caso, interação com bases de dados relacionais (Batra 2018). Ela é considerada uma linguagem descritiva, no sentido de que a suas "instruções descrevem o resultado desejado, em vez das etapas de computação necessárias"(Meier e Kaufmann 2019).

Existem algumas tecnologias que seguem este modelo de dados, como: (i) MySQL, (ii) PostgresSQL, (iii) Oracle database, e (iv) Microsoft SQL Server.

6.1.4.2 Modelo baseado em documentos

De acordo com (Miloslavskaya e Tolstoy 2016), o modelo baseado em documentos surgiu devido à crescente necessidade de analisar e armazenar dados não estruturados. Isto é, este tipo de modelo não define a estrutura de dados como sendo tabelas, nem a linguagem SQL como sendo a utilizada, ao contrário do modelo relacional (Meier e Kaufmann 2019). Neste caso, a base de dados armazena dados em documentos flexíveis, sendo considerada uma boa alternativa para as bases relacionais. As bases que adotam este modelo também são conhecidas como não-relacionais ou NoSQL (MongoDB 2023b).

Uma tecnologia popular que segue este modelo é a MongoDB, que consiste num sistema gratuito e open source que armazena os seus dados em conjuntos de documentos do tipo JSON (no caso, JSON binário, ou BSON) flexíveis, o que significa que os campos podem variar de documento para documento, e a estrutura de dados pode ser alterada ao longo do tempo. Para além disso, também permite realizar consultas *ad hoc*, indexação e agregação em tempo real, fornecendo boas formas de acesso e análise dos dados armazenados (MongoDB 2023c).

6.1.4.3 Escolha da tecnologia

Relativamente aos modelos previamente analisados, pode-se constatar que o modelo relacional possui algumas vantagens, como estar em conformidade com o ACID (que significa atomicidade, consistência, isolamentp, e durabilidade), que consiste num padrão que garante a confiabilidade das transações realizadas. Isto é, se alguma mudança falhar, toda a transação falha, e a base de dados irá manter o mesmo estado que tinha antes da transação ser iniciada, o que é importante porque em alguns casos, as consequências de certas transações podem ser bastante negativas.Para além disto, também garante precisão dos dados, com o suso de chaves primárias e estrangeiras, que evitam a duplicidade dos dados. Contudo, apresentam um pior desempenho e flexibilidade em comparação com modelos não-relacionais, uma vez que não existe nenhum esquema rígido para os dados, permitindo aplicar facilmente mudanças à base de dados (MongoDB 2023a).

Para o presente projeto, foi escolhido o modelo relacional, uma vez que, de acordo com (MongoDB 2023a), os dados são previsíveis, em termos de estrutura e tamanho, e as relações entre as entidades têm uma grande importância. Para além disto, os sistemas que adotam este modelo possuem um grande suporte e ferramentas, uma vez que já existem no mercado há bastantes anos. Este tipo de modelo também coincide com o utilizado pelo Status Monitor inicialmente, o que reforça ainda mais a escolha do mesmo.

Uma vez que o modelo foi escolhido, é necessário escolher a tecnologia a ser utilizada. Para este propósito, não foi realizado nenhum tipo de análise de diversas possíveis, uma vez que, como o projeto adota diversas tecnologias Microsoft, optou-se por apenas considerar o Microsoft SQL Server, desenvolvido pela própria, o que permite uma boa integração

com produtos da mesma marca, sendo uma escolha popular para empresas que possuam um ecossistema Microsoft (Plesk 2023). Para além disto, esta tecnologia corresponde à utilizada pela versão inicial do Status Monitor.

6.2 Descrição técnica

Esta secção guia o leitor pelas novas funcionalidades que foram adicionadas à Status Monitor, com uma descrição técnica de diversos elementos que são apresentados para os seus utilizadores. Os tópicos descritos são os seguintes:

- Dashboards implementadas;
- Status Monitor API e WatchDog;
- Injeções de código.

É importante mencionar que a empresa não conseguiu instrumentar o I'MTHOM por completo, pelo que alguns dados apresentados nos *prints* presentes nas próximas secções são simulados, e não correspondem a dados reais.

6.2.1 Dashboards implementadas

Como mencionado, no caso deste projeto, apenas são apresentadas as funcionalidades que foram adicionadas à versão inicial do Status Monitor. As restantes, já existentes, encontram-se descritas na secção 4.2. O menu parcial que é apresentado para o utilizador mantém-se o mesmo, tendo sido adicionado um novo grupo, designado por **Dashboards**, com novas opções, como se pode constatar na figura 6.1. Cada grupo é apresentado nas subsecções seguintes, cada um com uma própria.

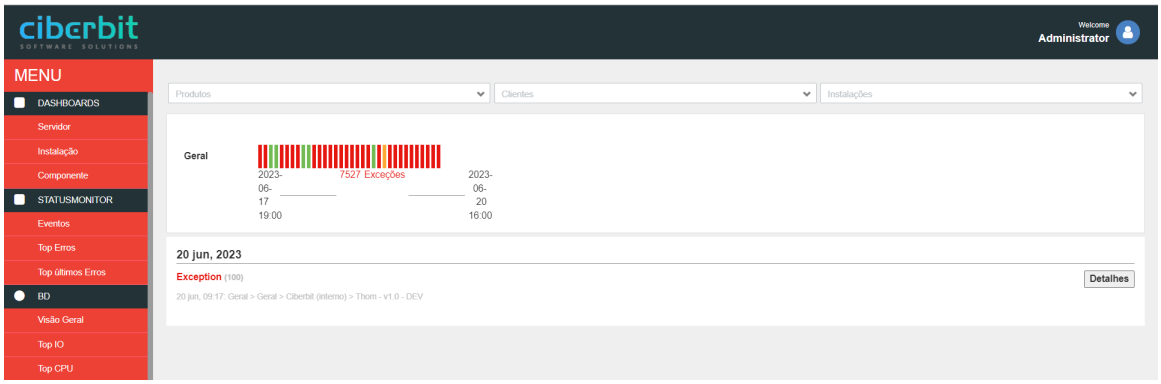

Figura 6.1: Página principal com novo grupo.

6.2.1.1 Dashboard dos servidores

A primeira opção desse grupo, apresentada na figura 6.2, permite que um determinado utilizador, com as devidas permissões, consiga visualizar uma lista com diversas informações referentes aos diversos servidores existentes referentes ao I'MTHOM.

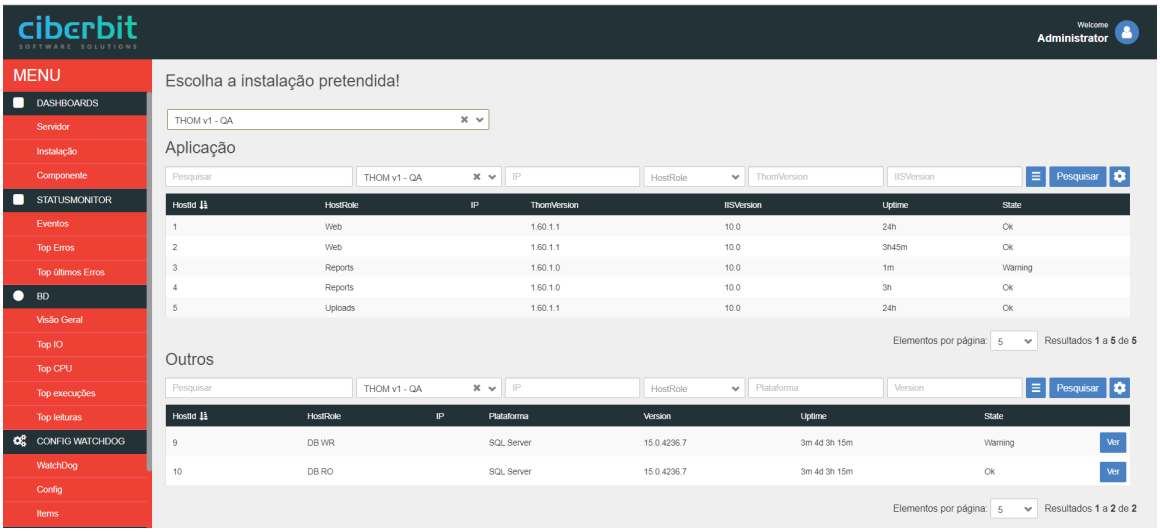

Figura 6.2: Dashboard dos servidores.

No início da página, é possível escolher uma instalação de uma lista com todas as presentes na base de dados, sendo que apenas são apresentados todos os servidores referentes à opção escolhida. Para além disso, os servidores encontram-se divididos em duas tabelas, sendo que os da primeira correspondem a servidores aplicacionais, enquanto que os da segunda se referem a servidores que correspondem a bases de dados e outros tipos existentes.

Para a criação das tabelas, foi utilizada a *framework* disponibilizada pela empresa, previamente mencionada na secção 6.1.1, permitindo, consequentemente, aplicar os diversos filtros aos dados que são apresentados na figura 6.2.

6.2.1.2 Dashboard das instalações

A segunda opção apresenta para o utilizador diversas métricas relativas a uma determinada instalação da aplicação monitorizada, como presente na figura 6.3.

A partir da página presente na figura 6.3, são apresentadas algumas métricas provenientes do I'MTHOM, como total de sessões ativas no momento, sendo que é recebida uma métrica com o valor do instante em que a mesma é enviada, e apresentada o valor da última recebida. Para além disso, também são apresentados o total de ocorrência de eventos do tipo exceção do último mês, e as que ocorreram no presente dia, e, também, a quantidade de servidores aplicacionais que se encontram num estado OK e NOK.

Algumas informações são apresentadas através de gráficos, construídos recorrendo à biblioteca DevExtreme, mencionada na secção 6.1.1. O primeiro gráfico apresenta a média mensal de alguns valores ao longo dos últimos 5 meses, como total de sessões ativas, número total de exceções, e número total de eventos, que neste caso não considera o tipo exceção. Já o segundo gráfico apresenta a quantidade dos diversos tipos de eventos que existem desde o último mês, com a adição do novo tipo implementado aquando da concretização deste projeto, referente a métricas.

É importante mencionar que a biblioteca utilizada para os gráficos permite que o utilizador consiga imprimir o conteúdo de cada um, assim como descarregar para o seu computador

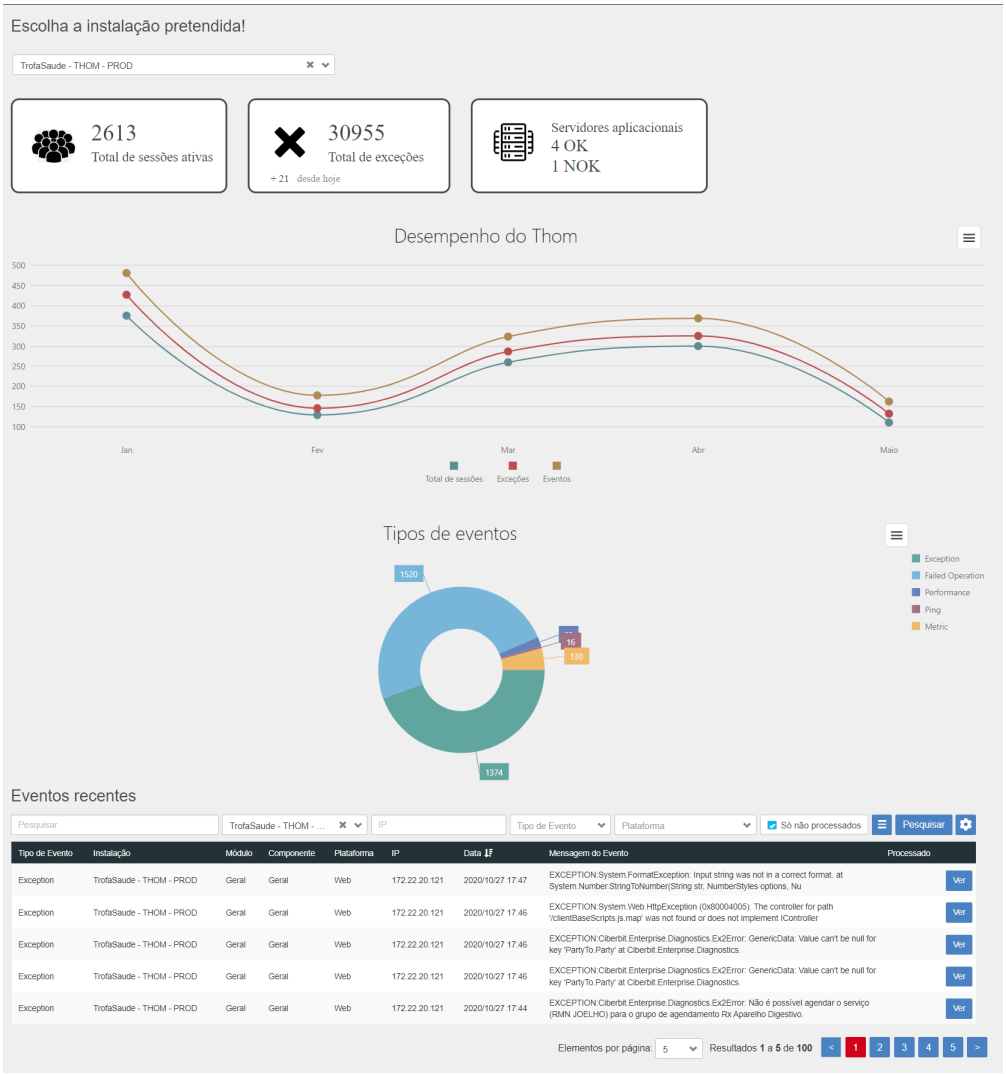

Figura 6.3: Dashboard das instalações.

imagens dos mesmos em diversos formatos, nomeadamente PNG, PDF, JPEG, SVG e até mesmo GIF.

Por último, é apresenta uma lista, construída recorrendo à *framework* desenvolvida pela empresa, contendo os eventos mais recentes para a instalação escolhida, isto é, os últimos 100 registados na base de dados, sendo utilizado um processo de paginação que apresenta 5 por página.

6.2.1.3 Dashboard dos componentes

Por último, a terceira opção, que consta na figura 6.4, apresenta diversas informações e métricas referentes a um determinado componente do sistema.

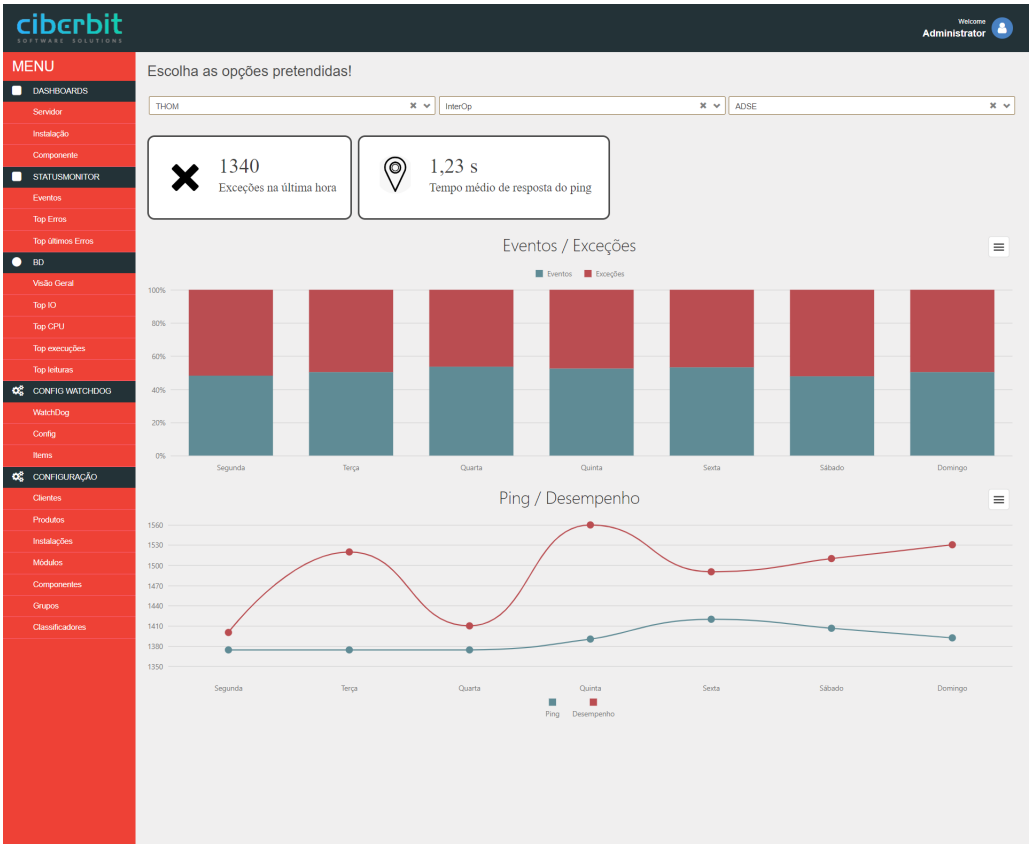

Figura 6.4: Dashboard dos componentes.

Uma vez que é possível existirem diversos componentes repetidos, já que módulos diferentes podem conter componentes iguais, e produtos diferentes podem conter módulos iguais, são apresentadas 3 listas para o utilizador aquando do acesso à página em questão. No caso, numa primeira fase, o utilizador apenas pode escolher um produto dentro dos existentes, não sendo apresentada qualquer opção para os restantes campos. Após essa escolha, é possível selecionar um módulo de uma lista de módulos, referentes ao produto previamente selecionado, e assim sucessivamente.

Após a escolha do componente, são apresentados dados referentes à junção das 3 opções escolhidas. São, portanto, apresentados o número total de exceções associados aos mesmos que ocorreram na última hora, assim como o tempo médio de resposta de um *ping* nesse mesmo dia. Para além disto, são apresentados dois gráficos, construídos da mesma forma que os mencionados na secção anterior, sendo que o primeiro permite visualizar dados referentes ao número de exceções e número de eventos (com exclusão do tipo exceção) ao longo da última semana, para cada dia da mesma. Já o segundo permite visualizar valores referentes ao tempo médio do *ping* e desempenho do I'MTHOM (isto é, tempo médio de resposta a um pedido), também ao longo da última semana, em milisegundos.

6.2.2 Status Monitor API e WatchDog

Relativamente à API utilizada na captação de dados telemétricos do I'MTHOM e do serviço WatchDog, os mesmos foram devidamente configurados por forma a suportar a nova

estrutura de dados, no caso, eventos do tipo métrica. Ou seja, foi implementado um novo modelo de dados, adequado para os mesmos, tanto para que pudesse ser utilizado no Status Monitor, para os dados recebidos do I'MTHOM, como no WatchDog, uma vez que os dados que o mesmo capta do I'MTHOM são disponibilizados no formato JSON, sendo, numa fase posterior, convertidos para um modelo de dados específico e enviados para o Status Monitor.

6.2.3 Injeções de código

Como mencionado na secção 3.2.3, um dos atributos de suportabilidade consiste em validar os dados recebidos, por forma a evitar ocorrências de injeções de código. Para isso, a todos os métodos de classes Controller que recebem dados, foi aplicado um método que verifica os dados e os transforma, por forma a que os mesmos não possam ser executados no *browser* quando apresentados para os utilizadores. O Código 6.1 apresenta o exemplo de uma aplicação desse processo - linha 3.

```
1     public ActionResult AveragePing(string value)
2 {
3     Server.HtmlEncode(value);
4     var averagePing = ComponentDashboardModelMgr.GetAveragePing(ctx, value
      );
5     return PartialView("~/Views/ComponentDashboard/AveragePing.cshtml",
      averagePing);
6 }
```

Código 6.1: Exemplo de uma proteção a injeção de código

6.3 Testes

De acordo com (IBM 2023a), a testagem de *software* consiste num "processo para avaliar e verificar que um produto de *software* ou aplicação faz o que é suposto fazer". Existem diversos benefícios provenientes do desenvolvimento de testes, como validar as funcionalidades implementadas, prevenir *bugs*, melhorar a qualidade do código, assim como reduzir custos de desenvolvimento (Hughes 2017; IBM 2023a).

Como constatado em (Pittet 2023; ticapp 2021), existem diversos tipos de testes que podem ser implementados. Após uma reunião realizada com o supervisor do autor, foi decidido que apenas são realizados testes para as 4 categorias apresentadas a seguir, por serem mais relevantes para a empresa atualmente:

- **Testes unitários:** Permitem testar o comportamento de funções individuais pertencentes a classes, componentes, ou módulos da solução;
- **Testes End-to-End:** Permitem replicar o comportamento de um determinado utilizador do produto, verificando se esses fluxos funcionam como esperado;
- **Testes de desempenho:** Permitem verificar se o produto consegue funcionar sob determinadas situações, permitindo avaliar a estabilidade, disponibilidade e confiabilidade do sistema;
- **Testes de usabilidade:** Permitem avaliar a qualidade do sistema, atendendo ao grau de eficácia, eficiência e satisfação dos utilizadores na execução das funcionalidades implementadas.

Das categorias supracitadas, os testes de desempenho e usabilidade são discutidos com maior detalhe no capítulo 7. Já os restantes são analisados nas seguintes secções, com alguns exemplos de testes realizados.

6.3.1 Testes unitários

Esta secção é focada nos testes unitários que foram desenvolvidos para a solução. A título de exemplo, apenas um teste será apresentado, no Código 6.2. O mesmo verifica que um utilizador consegue obter um modelo com diversas informações referentes à *dashboard* de instalações. O teste depende do acesso à base de dados, por forma a obter os dados necessários. Uma vez que se trata de um teste unitário, e que a sua responsabilidade não se prende em verificar a integridade da solução, as classes que acedem aos recursos mencionados precisam de ser simuladas. Para esse efeito, é utilizada a *framework* MOQ, juntamente com a *framework* NUnit, para realizar este tipo de testes.

```
1     [SetUp]
2     public void SetUp()
3     {
4         _mockRepo = new Mock<InstallationBO>();
5         _mockRepo.Setup(repo => repo.GetAllInstallations(null))
6             .Returns(new List<Installation>());
7     }
8
9     [Test]
10    public void ensureModelIsCorrect()
11    {
12        _service = new InstallationDashboardModelMgr(_mockRepo.Object);
13
14        var result = _service.GetModel(null);
15
16        _mockRepo.Verify(mock => mock.GetAllInstallations(null), Times.
    Once());
17
18        Assert.IsInstanceOf<InstallationDashboardModel>(result);
19        Assert.AreEqual(0, result.Installations.Count);
20    }
```

Código 6.2: Exemplo de um teste unitário

Como se pode constatar, apenas é verificado o comportamento isolado do serviço InstallationDashboardModelMgr. Esse serviço necessita da classe InstallationBO, que corresponde à classe que atua como repositório, para poder comunicar com a base de dados. Por isso, a mesma é simulada - linhas 4, 5 e 6 - por forma a não existir contacto com a base. Após isso, é verificado se o objeto simulado foi chamado uma vez - linha 16 - se o tipo retornado pelo método é o mesmo do modelo - linha 18 - e se o número de elementos da lista é igual a 0, uma vez que o objeto simulado foi instruído a simular o retorno de uma lista vazia - linha 19.

6.3.2 Testes End-to-End

Esta secção apresenta, a título de exemplo, um teste end-to-end realizado para o Status Monitor. Este tipo de teste avalia a forma como o sistema responde a diversas interações provenientes de utilizadores do mesmo. Todos os testes foram realizados recorrendo ao Cypress, uma *framework* específica para a realização destas testagens.

O Código 6.3 apresenta o exemplo de um comando criado para poder ser utilizado em testes. No caso, o comando é responsável por procurar o texto "Instalação"no menu principal, e selecioná-lo, permitindo, consequentemente, o acesso à página onde é apresentado o *dashboard* das instalações.

```
1     Cypress.Commands.add('accessInstallationDashboard', () => {
2         cy.contains('Instalação').click();
3  });
```

Código 6.3: Exemplo de um comando do Cypress

O teste apresentado no Código 6.4 faz uso do comando supracitado, permitindo o acesso de um utilizador devidamente autenticado, no caso um administrador do sistema, à *dashboard* das instalações, sendo verificado que o mesmo possui acesso a todos os conteúdos da mesma.

```
1     describe('UI', () => {
2         beforeEach(() => cy.visit('/'));
3         it('should present informations about installation dashboard', ()
    => {
4             cy.login();
5             cy.accessInstallationDashboard();
6             cy.contains('Total de sessões ativas');
7             cy.contains('Total de exceções');
8             cy.contains('Servidores aplicacionais');
9             cy.contains('Desempenho do Thom');
10            cy.contains('Tipos de eventos');
11            cy.contains('Eventos recentes');
12        })
13    })
```

Código 6.4: Exemplo de um teste end-to-end

Capítulo 7

Avaliação

Este capítulo tem como objetivo descrever as avaliações que foram efetuadas sobre a solução desenvolvida.

O comportamento esperado do sistema, de acordo com os requisitos funcionais propostos, pode ser verificado com a execução de testes determinísticos, isto é, testes que fornecem sempre os mesmos resultados, considerando que a solução não é modificada, presentes na secção 6.3. Contudo, alguns requisitos, como desempenho e usabilidade, não podem ser testados de forma determinística. Por isso, todas as experiências realizadas que focam em testar esses requisitos, definidos na secção 3.2, são apresentadas neste capítulo.

Por forma a concretizar isso, o presente capítulo encontra-se dividido em duas secções, uma para cada um dos requisitos a serem testados. Cada secção apresenta a abordagem a ser adotada, seguida da apresentação de todos os resultados obtidos com as experiências realizadas, e conclusões consequentes, com base nesses mesmos resultados.

7.1 Usabilidade

O objetivo da avaliação desta grandeza consiste em provar se os requisitos não-funcionais definidos na secção 3.2.2 são ou não cumpridos. Isto é, se o sistema, após a adição das funcionalidades implementadas, possui um alto grau de usabilidade, por forma a fornecer aos seus utilizadores uma boa experiência de utilização.

7.1.1 Abordagem

Para tal, a abordagem a ser adotada consiste na realização de inquéritos de satisfação, aplicados a um determinado grupo de indivíduos. Uma vez que o projeto desenvolvido ainda se encontra numa fase inicial de desenvolvimento, onde o público geral não possui acesso ao mesmo, o grupo de teste é apenas constituído pelo próprio supervisor da empresa, o que não representa uma amostra muito significativa.

Este inquérito é baseado no método *System Usability Scale* (SUS), que consiste num conjunto de dez questões que permitem avaliar a usabilidade de uma grande variedade de produtos e serviços, incluindo *hardware*, *software*, *websites*, aplicações, entre outros. Neste caso, serve para avaliar a *interface* das funcionalidades desenvolvidas. As questões apresentadas para os utilizadores são as seguintes (Brooke 1995):

- 1. Gostaria de utilizar este sistema frequentemente;
- 2. Acho o sistema desnecessariamente complexo;

- 3. Acho o sistema fácil de utilizar;
- 4. Sinto que preciso da ajuda para conseguir utilizar o sistema;
- 5. Considero que as funções do sistema estão bem integradas;
- 6. Acho que existia muitas inconsistências no sistema;
- 7. Imagino que a maioria das pessoas se adaptaria rapidamente ao sistema;
- 8. Acho que o sistema é muito complicado de se utilizar;
- 9. Sinto bastante confiança quando utilizo o sistema;
- 10. Preciso de aprender bastantes coisas antes de começar a utilizar o sistema.

Relativamente às respostas, a escala utilizada é a de Likert, onde os inquiridos especificam, para cada questão, o seu nível de concordância ou discordância com base no seguinte formato (Brooke 1995):

- Discordo totalmente;
- Discordo;
- Indiferente;
- Concordo;
- Concordo totalmente.

7.1.2 Resultados

Na presente secção são apresentados os resultados obtidos com o inquérito previamente mencionado. Uma vez que apenas um utilizador do sistema conseguiu responder, os resultados são apresentados numa única tabela, no caso, a tabela 7.1. Para cada pergunta, é apresentado um número, que corresponde ao número dela por ordem de apresentação na lista de questões contidas no inquérito, presente na secção anterior.

Tabela 7.1: Resultados para inquérito de usabilidade

Resultados					
Pergunta	Discordo totalmente	Discordo	Indiferente	Concordo	Concordo totalmente
1	-	-	-	✓	-
2	-	✓	-	-	-
3	-	-	-	✓	-
4	-	✓	-	-	-
5	-	-	✓	-	-
6	-	-	✓	-	-
7	-	-	-	✓	-
8	-	✓	-	-	-
9	-	-	-	✓	-
10	-	✓	-	-	-

Com recurso ao método SUS é possível obter um número que manifesta a usabilidade geral do sistema. Por forma a calcular esse valor, é necessário sumar os pontos de contribuição

de cada questão. Para as questões 1, 3, 5, 7, e 9 a pontuação de cada consiste no valor da sua posição menos 1. Já para os restantes, a pontuação consiste em subtrair a sua posição ao número 5. No final, o valor obtido precisa de ser multiplicado por 2.5, por forma a obter o resultado final, que varia entre 0 e 100 (Brooke 1995).

Considerando os resultados obtidos, a pontuação adquirida foi 70. Ou seja, dentro da escala, corresponde a um bom valor de usabilidade.

7.2 Desempenho

Assim como acontece na grandeza anteriormente apresentada, o objetivo consiste em provar se os requisitos não-funcionais definidos na secção 3.2.4 são ou não cumpridos. Isto é, se o sistema apresenta os dados para o utilizador, quando o mesmo acede a uma das *dashboards* desenvolvidas, num espaço de tempo de 2 segundos em 90% dos casos, podendo ser utilizado por diversos utilizadores ao mesmo tempo.

7.2.1 Abordagem

A abordagem adotada para avaliar esta grandeza consiste no envio de volumes cada vez mais elevados de pedidos HTTP para um *endpoint* referente a uma *dashboard* implementada, por forma a determinar os limites da aplicação aquando desse tipo de acesso.

A avaliação é realizada para cada *dashboard* implementada: (i) Dashboard dos servidores, (ii) Dashboard das instalações, (iii) Dashboard dos componentes.

Os testes de desempenho, no caso, de carga, foram efetuados recorrendo a uma ferramenta designada K6. Esta ferramenta permite desenvolver esse tipo de testes em JavaScript.

Para realizar os testes, a solução foi executada na máquina local do autor, uma vez que não existia possibilidade de utilizar uma *Virtual Machine* (VM). Essa mesma máquina também foi utilizada para executar os testes, não existindo, consequentemente, nenhum tipo de problema de comunicação que pudesse surgir com o uso de VMs, como estabilidade da conexão. A máquina possui as seguintes especificações:

- Memória: 16 GB;
- Número de núcleos vCPU: 4;
- *Solid State Drive* (SSD).

7.2.2 Resultados

Como mencionado anteriormente, 3 cenários que simulam um conjunto de utilizadores a tentarem aceder a cada uma das seguintes dashboards desenvolvidas foram testados:

- Dashboard dos servidores;
- Dashboard das instalações;
- Dashboard dos componentes.

Cada um dos cenários examina o tempo desde que um determinado utilizador acede a uma página com dados referentes a dashboards, até ao momento em que a mesma os apresenta.

Para o caso destas experiências, considerou-se que os utilizadores já se encontravam autenticados na plataforma, evitando, assim, todo o processo de *login*. A tabela 7.2 apresenta diversos dados referentes a cada experiência realizada, para cada dashboard. No caso, para cada experiência, cada utilizador acede à página de 5 em 5 segundos, até atingir o valor máximo de pedidos.

Tabela 7.2: Detalhes referentes às experiências realizadas

Experiência	Número de utilizadores	Intervalo de envio	Total de pedidos
A	50	5 segundos	500
B	100	5 segundos	1000
C	200	5 segundos	2000
D	500	5 segundos	5000
E	1000	5 segundos	10000

Os resultados de cada cenário são apresentados nas secções seguintes, uma para cada dashboard. Esses mesmos resultados são apresentados em tabelas contendo as seguintes métricas: (i) tempo médio dos pedidos, (ii) tempo máximo de um pedido, (iii) tempo mínimo de um pedido, (iv) mediana, (v) percentil 90, e (vi) percentil 95. Um percentil, por exemplo, de 90, significa que 90% dos resultados são iguais ou inferiores a esse valor.

7.2.2.1 Dashboard dos servidores

A tabela 7.3 apresenta os resultados referentes ao tempo que demora a ser apresentada a página da *dashboard* em questão para um utilizador, desde o momento em que o mesmo a requisita.

Tabela 7.3: Resultados para dashboard dos servidores (em segundos)

Experiência	Média	Mínimo	Máximo	Mediana	Percentil 90	Percentil 95
A	0.112	0.026	0.669	0.086	0.159	0.382
B	0.161	0.028	1.330	1.000	0.262	0.708
C	0.317	0.026	2.810	0.061	1.310	1.530
D	1.930	0.138	6.580	1.580	3.150	3.840
E	12.380	0.131	21.300	12.270	19.680	20.530

Os resultados apresentados na tabela 7.3 demonstram que o sistema consegue responder, de forma bem sucedida, a 36 pedidos por segundo, enquanto consegue responder aos requisitos definidos. O valor mencionado foi calculado a partir do tempo total de execução da experiência C, a última bem sucedida, uma vez que as experiências D e E apresentam resultados que não cumprem com esses mesmos requisitos.

7.2.2.2 Dashboard das instalações

Os resultados referentes ao tempo medido anteriormente, mas aplicado à *dashboard* das instalações desenvolvida encontram-se descritos na tabela 7.4.

Como é possível constatar, a tabela 7.4 demonstra que o sistema consegue responder, de forma bem sucedida, a 37 pedidos por segundo (correspondendo à experiência C), enquanto

Tabela 7.4: Resultados para dashboard das instalações (em segundos)

Experiência	Média	Mínimo	Máximo	Mediana	Percentil 90	Percentil 95
A	0.186	0.032	1.470	0.104	0.230	1.030
B	0.193	0.028	1.410	0.880	0.552	0.767
C	0.268	0.029	2.970	0.105	0.534	1.460
D	3.380	0.180	10.540	2.860	5.900	6.790
E	9.080	0.425	13.480	8.920	12.130	12.720

consegue responder aos requisitos definidos. Assim como no cenário anterior, as experiências D e E apresentam resultados que não cumprem com esses mesmos requisitos.

7.2.2.3 Dashboard dos componentes

Por último, os resultados referentes ao tempo medido anteriormente, mas aplicado à *dashboard* dos componentes desenvolvida encontram-se presentes na tabela 7.5.

Tabela 7.5: Resultados para dashboard dos componentes (em segundos)

Experiência	Média	Mínimo	Máximo	Mediana	Percentil 90	Percentil 95
A	0.085	0.009	0.726	0.042	0.216	0.335
B	0.087	0.008	0.481	0.048	0.188	0.262
C	0.130	0.008	1.320	0.049	0.252	0.651
D	0.414	0.008	4.470	0.030	1.340	2.850
E	0.944	0.183	5.910	0.704	1.340	2.500

Diferente daquilo que se verifica nos cenários anteriores, a experiência E apresenta resultados que respondem aos requisitos definidos, demonstrando que o sistema consegue responder, de forma bem sucedida, a 160 por segundo. Esta diferença deve-se ao facto de que, para esta *dashboard* em particular, é apresentada apenas uma lista assim que um utilizador acede à mesma, e não todos os componentes da mesma, como nos 2 casos anteriores. A opção escolhida influencia os dados de 2 outras listas, que consequentemente origina o surgimento de diversos outros dados, esses sim de extrema importância para o utilizador.

Capítulo 8

Conclusão

Neste capítulo são apresentados os objetivos alcançados, assim como aqueles que não foram concretizados, o trabalho futuro deste projeto e, por último, uma apreciação final do autor, explicitando a influência que este trabalho teve, tanto no desenvolvimento da solução final, como no próprio, em termos de conhecimento adquirido.

É importante mencionar que, durante o tempo de produção do projeto, os objetivos inicialmente definidos demonstraram-se bem mais complexos e desafiantes do que era previsto, especialmente considerando o tempo disponível. Para além disto, as mudanças realizadas ao nível dos requisitos, no que toca aos objetivos fizeram com que o autor tivesse perdido bastante tempo e recursos, o que, consequentemente, influenciou o produto final.

Contudo, o autor focou em 3 objetivos, dos 5 propostos. Os mesmos foram desenvolvidos de acordo com os requisitos discutidos durante as reuniões com o supervisor, tendo apresentado níveis positivos de avaliação, tanto de usabilidade como de desempenho.

8.1 Objetivos alcançados

O objetivo principal desta dissertação consistiu em aumentar o grau de observabilidade dos dados de uma aplicação da empresa designada por I'MTHOM, por forma a tornar possível visualizar diversas métricas dessa aplicação, como erros, estado de diversos componentes, entre outros. Para isso, foram adicionadas funcionalidades a uma ferramenta já existente, com esse propósito, mas que se encontrava bastante incompleta.

As funcionalidades implementadas tornaram possível a captação de métricas, algo que não era possível anteriormente, e a apresentação das mesmas, para além de outras informações, referentes a diversos conceitos do I'MTHOM, como servidores, instalações, e componentes. Mesmo que não tenham sido desenvolvidas todas as funções inicialmente propostas, o trabalho realizado permitiu que a empresa conseguisse ter acesso a uma plataforma mais rica, que permite uma melhor incorporação de outras funcionalidades, acrescentando, assim, ainda mais valor ao sistema.

Em suma, pode-se concluir que grande parte dos objetivos foram alcançados. Mais precisamente, os objetivos referentes à criação de dashboards de servidores, instalações, e componentes. Estes foram escolhidos para serem implementados em detrimento dos outros, devido ao facto de ter sido definido na secção 3.1 que estes eram os de maior prioridade.

8.2 Objetivos não alcançados

O autor acredita que os objetivos de maior prioridade foram implementados de forma bem sucedida. Contudo, nem todos foram, nomeadamente a criação de uma *dashboard* que apresentasse informações e métricas referentes às múltiplas instâncias de base de dados do I'MTHOM, assim como a implementação do conceito de alertas, com base em regras criadas. O motivo principal para estas funcionalidades não terem sido desenvolvidas prende-se com a sua complexidade, tendo em consideração o tempo disponível, e na prioridade definida para os mesmos, para além de algumas falhas, por parte do autor, no planeamento do projeto, numa fase inicial do mesmo. Assim como aconteceu com os requisitos funcionais, nem todos os atributos de qualidade propostos foram respondidos, sendo que o Apêndice A apresenta mais alguns detalhes a respeito disso.

Para além dos fatores supracitados, é importante mencionar que na proposta inicial, criada em outubro de 2022, os objetivos eram os mesmos, contudo, os mesmos seriam divididos por dois alunos, que iriam trabalhar em conjunto por forma a atingi-los. No caso, o autor desta dissertação ficaria responsável pelo desenvolvimento de toda a parte referente à frontend do sistema, enquanto que o outro colega seria responsável pela criação da backend. Contudo, após algumas reuniões com os respetivos orientadores e supervisor da empresa, foi definido que seria melhor criar 2 abordagens diferentes, em vez que existir um projeto partilhado, uma vez que os objetivos inicias não pareciam ser suficientes para cada aluno.

Desta forma, a proposta final, realizada por volta de fevereiro de 2023, que corresponde à adotada no presente documento, é considerada mais apropriada. Contudo, a reformulação prejudicou um pouco o trabalho anteriormente desenvolvido, o que também contribuiu, de certo modo, para o estado atual da solução.

De uma forma geral, tendo em conta tudo o que foi discutido nesta secção, o autor considera que os objetivos implementados, mesmo não correspondendo a todos os propostos, acrescentam valor à solução inicial.

8.3 Trabalho futuro

Este projeto, e consequentemente a solução desenvolvida, ainda possui diversas ideias e funcionalidades que podem ser implementadas futuramente, nomeadamente as seguintes:

- Desenvolvimento dos objetivos propostos que não foram alcançados, assim como melhoria dos que foram implementados, no sentido de aumentar ainda mais o grau de observabilidade dos dados provenientes do I'MTHOM;
- Alteração de toda a interface da aplicação para uma nova, com um visual mais apelativo. Este requisito foi discutido com o supervisor numa fase inicial, contudo, o mesmo decidiu que, por enquanto, não seria necessário incorporá-lo na solução atual;
- Recolha de informações referentes a todas as interações do I'MTHOM, isto é, dados de *distributed traces*. Como mencionado na secção 2.1.2, este tipo de dado corresponde a um dos três pilares de observabilidade, sendo de extrema importância a sua recolha e posterior apresentação para utilizadores interessados. Este ponto também foi discutido com o supervisor, que considerou que a inclusão deste tipo de informação no Status Monitor exigiria um esforço demasiado elevado e, portanto, nem sequer foi considerado para este âmbito;

- Implementação de um processo de registo de utilizadores, por forma a tornar mais prática a adição dos mesmos, permitindo que qualquer utilizador do sistema, com as devidas permissões, possa fazer isso através da plataforma, em vez que ter que escrever consultas que comunicam diretamente com a base de dados.

8.4 Apreciação final

Em suma, a solução apresentada constitui um passo importante para o desenvolvimento de um projeto que permite apresentar diversos tipos de dados que denunciam o estado do I'MTHOM, uma vez que a solução ainda se encontra num estado incompleto, e necessita de um maior número de funcionalidades, por forma a se tornar um projeto mais robusto, com uma maior utilidade para a empresa.

Contudo, é importante reconhecer que a abordagem adotada não é a melhor, uma vez que já existem no mercado diversas soluções que conseguem responder às necessidades da organização, e que possuem um grande número de funcionalidades, grandes comunidades de desenvolvedores ativas, suporte disponível, e que são mantidas por empresas de confiança. O tipo de abordagem seguido, por sua vez, exigia a implementação dessas mesmas funcionalidades por parte de um único elemento, o que, consequentemente, resultou numa solução inferior quando comparada com as já existentes. Por culpa disso, considera-se que a outra abordagem que poderia ser adotada, que consistia no uso desses produtos já existentes e, a partir deles, criar uma nova solução, seria a melhor.

Apesar disso, o autor considera que o desenvolvimento do projeto o beneficiou. O mesmo considera que as suas competências técnicas aumentaram, devido à utilização de diversas tecnologias e *frameworks* para a implementação, assim como ao estudo e análise de diversos conceitos e soluções referentes ao tema da dissertação, contribuindo para o ganho de experiência e conhecimento na área de telemetria. O projeto também permitiu entender melhor as dificuldades provenientes da abordagem adotada, em detrimento de outras existentes.

Bibliografia

Batra, Rahul (2018). «A History of SQL and Relational Databases». Em: *SQL Primer: An Accelerated Introduction to SQL Basics*. Berkeley, CA: Apress, pp. 183–187. isbn: 978-1-4842-3576-8. doi: 10.1007/978-1-4842-3576-8_19. url: https://doi.org/10.1007/978-1-4842-3576-8_19.

Beyer, Betsy et al. (2016). *Site Reliability Engineering: How Google Runs Production Systems*. url: http://landing.google.com/sre/book.html.

Britannica, Encyclopedia (2023). *telemetry summary*. Acedido em: 31 jan. 2023. url: https://www.britannica.com/summary/telemetry.

Brooke, John (nov. de 1995). «SUS: A quick and dirty usability scale». Em: *Usability Eval. Ind.* 189.

Brown, Simon (2023). *The C4 model for visualising software architecture*. Acedido em: 03 maio 2023. url: https://c4model.com.

Chakraborty, Mainak e Ajit Pratap Kundan (2021). «Observability». Em: *Monitoring Cloud-Native Applications: Lead Agile Operations Confidently Using Open Source Software*. Berkeley, CA: Apress, pp. 25–54. isbn: 978-1-4842-6888-9. doi: 10.1007/978-1-4842-6888-9_2. url: https://doi.org/10.1007/978-1-4842-6888-9_2.

Ciberbit (2023a). *i'mthom*. Acedido em: 31 jan. 2023. url: https://www.imthom.com/.

– (2023b). *Site Oficial*. Acedido em: 31 jan. 2023. url: https://www.ciberbit.pt/pt/.

Dávideková, Monika e Michal Gregu MI (2016). «Software Application Logging: Aspects to Consider by Implementing Knowledge Management». Em: *2016 2nd International Conference on Open and Big Data (OBD)*, pp. 102–107. doi: 10.1109/OBD.2016.22.

DEI-ISEP (2021). «Model View Controller». Em: Acedido em: 13 maio 2023.

DevExtreme (2023). *DevExtreme*. Acedido em: 13 maio 2023. url: https://js.devexpress.com/.

Eeles, Peter (2005). «Capturing architectural requirements». Em: *IBM Rational developer works*.

Elmasri, R. e S. Navathe (2017). *Fundamentals of Database Systems*. Pearson/Addison Wesley. isbn: 9789332582705. url: https://books.google.pt/books?id=s5d8zwEACAAJ.

Freixo, Sara (out. de 2021). *Ciberbit: soluções tecnológicas de futuro*. url: https://www.valormagazine.pt/ciberbit-solucoes-tecnologicas-de-futuro/.

Gatev, R. (2021). *Introducing Distributed Application Runtime (Dapr): Simplifying Microservices Applications Development Through Proven and Reusable Patterns and Practices*. Apress. isbn: 9781484269992. url: https://books.google.pt/books?id=Hv19zwEACAAJ.

Grafana (2023). *OpenTelemetry Collector*. Acedido em: 16 fev. 2023. url: https://grafana.com/docs/opentelemetry/collector/.

Hughes, Karl (out. de 2017). *Why Testing Is Important for Distributed Software*. Acedido em: 15 junho 2023. url: https://www.linuxfoundation.org/blog/blog/testing-important-distributed-software.

IBM (2023a). *How does software testing work?* Acedido em: 15 junho 2023. url: https://www.ibm.com/topics/software-testing.

IBM (2023b). *What is three-tier architecture?* Acedido em: 04 maio 2023. url: https://www.ibm.com/topics/three-tier-architecture.

Infopédia (2023). *outlier*. Acedido em: 8 fev. 2023. url: https://www.infopedia.pt/dicionarios/lingua-portuguesa/outlier.

Janocová, Renata (2012). «Evaluation of software quality». Em: *IMEA 2012*, p. 24.

Kechagia, Maria, Dimitris Mitropoulos e Diomidis Spinellis (dez. de 2015). «Charting the API minefield using software telemetry data». Em: *Empirical Software Engineering* 20.6, pp. 1785–1830. issn: 1573-7616. doi: 10.1007/s10664-014-9343-7. url: https://doi.org/10.1007/s10664-014-9343-7.

Kruchten, Philippe (nov. de 1995). «The 4+1 View Model of Architecture». Em: *IEEE Software*, pp. 42–50. doi: 10.1109/52.469759.

Lamsweerde, Axel van (2009). *Requirements Engineering: From System Goals to UML Models to Software Specifications*. 1st. Wiley Publishing. isbn: 0470012706.

Logz.io (2023). *Beginners Guide to OpenTelemetry*. Acedido em: 16 fev. 2023. url: https://logz.io/learn/opentelemetry-guide/#overview.

Malan, Ruth, Dana Bredemeyer et al. (2001). «Functional requirements and use cases». Em: *Bredemeyer Consulting*.

Meier, Andreas e Michael Kaufmann (jan. de 2019). *SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management*. isbn: 978-3-658-24548-1. doi: 10.1007/978-3-658-24549-8.

Microsoft (set. de 2022a). *ASP.NET overview*. Acedido em: 13 maio 2023. url: https://learn.microsoft.com/en-us/aspnet/overview.

– (jun. de 2022b). *Overview of ASP.NET Core MVC*. Acedido em: 13 maio 2023. url: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-7.0&WT.mc_id=dotnet-35129-website.

– (2023a). *Analisar métricas de um recurso do Azure*. Acedido em: 8 fev. 2023. url: https://learn.microsoft.com/pt-pt/azure/azure-monitor/essentials/tutorial-metrics.

– (2023b). *What is .NET Framework?* Acedido em: 9 fev. 2023. url: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework.

– (2023c). *What is .NET?* Acedido em: 8 fev. 2023. url: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet.

– (2023d). *What is ASP.NET?* Acedido em: 9 fev. 2023. url: https://dotnet.microsoft.com/pt-br/learn/aspnet/what-is-aspnet.

Miloslavskaya, Natalia e Alexander Tolstoy (2016). «Big Data, Fast Data and Data Lake Concepts». Em: *Procedia Computer Science* 88. 7th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2016, held July 16 to July 19, 2016 in New York City, NY, USA, pp. 300–305. issn: 1877-0509. doi: https://doi.org/10.1016/j.procs.2016.07.439. url: https://www.sciencedirect.com/science/article/pii/S1877050916316957.

MongoDB (2023a). *Relational vs. Non-Relational Databases*. Acedido em: 13 maio 2023. url: https://www.mongodb.com/compare/relational-vs-non-relational-databases.

– (2023b). *What is a Document Database?* Acedido em: 13 maio 2023. url: https://www.mongodb.com/document-databases.

– (2023c). *What Is MongoDB?* Acedido em: 13 maio 2023. url: https://www.mongodb.com/en-us/what-is-mongodb.

NewRelic (2023a). *Introduction to alerts and applied intelligence*. Acedido em: 20 junho 2023. url: https://docs.newrelic.com/docs/alerts-applied-intelligence/overview/.

– (2023b). *New Relic*. Acedido em: 20 junho 2023. url: https://newrelic.com/lp/developersignup?utm_medium=cpc&utm_source=google&utm_campaign=EVER-GREEN_BRAND_SEARCH_BRAND_EMEA_WESTERN_EN&utm_network=g&utm_keyword=new%20relic&utm_device=c&_bt=659627992530&_bm=e&_bn=g&gclid=Cj0KCQjw1_SkBhDwARIsANbGpFvLgeGJ0O8xCcI_pY-lzqLhBA68_7tIQbqGDnaR_alaywLb1Mqo34AaAgVIEALw_wcB.
– (2023c). *New Relic*. Acedido em: 20 junho 2023. url: https://newrelic.com/.
– (2023d). *Start using New Relic*. Acedido em: 20 junho 2023. url: https://docs.newrelic.com/docs/new-relic-solutions/get-started/intro-new-relic/.

OpenTelemetry (2023a). *Collector*. Acedido em: 16 fev. 2023. url: https://opentelemetry.io/docs/collector/.
– (2023b). *Collector - Configuration*. Acedido em: 16 fev. 2023. url: https://opentelemetry.io/docs/collector/configuration/.
– (2023c). *Collector - Deployment*. Acedido em: 16 fev. 2023. url: https://opentelemetry.io/docs/collector/deployment/.
– (2023d). *Components*. Acedido em: 16 fev. 2023. url: https://opentelemetry.io/docs/concepts/components/.
– (2023e). *Instrumenting*. Acedido em: 15 fev. 2023. url: https://opentelemetry.io/docs/concepts/instrumenting/.
– (2023f). *What is OpenTelemetry?* Acedido em: 15 fev. 2023. url: https://opentelemetry.io/docs/concepts/what-is-opentelemetry/.

Picoreti, Rodolfo et al. (2018). «Multilevel Observability in Cloud Orchestration». Em: *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pp. 776–784. doi: 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00134.

Pittet, Sten (2023). *The different types of software testing*. Acedido em: 15 junho 2023. url: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing.

Plesk (abr. de 2023). *MySQL vs MSSQL: Comparing Similarities and Differences*. Acedido em: 13 maio 2023. url: https://www.plesk.com/blog/various/mysql-vs-mssql/.

QualidadeBR (2008). *FURPS+*. Acedido em: 25 fev. 2023. url: https://qualidadebr.wordpress.com/2008/07/10/furps/.

Riedesel, Jamie (ago. de 2021). *Software Telemetry: Reliable Logging and Monitoring*. Manning.

S.A., Ciberbit (2023). *Sobre nós*. Acedido em: 31 jan. 2023. url: https://pt.linkedin.com/company/ciberbit.

Silva, Nuno (2020). *Views*. Acedido em: 03 maio 2023. url: https://bitbucket.org/nunopsilva/bulletproof-nodejs-ddd/wiki/Views.md.

ticapp (2021). *Testes de Usabilidade*.

Timescale (2021). *What Are Traces, and How SQL (Yes, SQL) and OpenTelemetry Can Help Us Get More Value Out of Traces to Build Better Software*. Acedido em: 16 fev. 2023. url: https://www.timescale.com/blog/what-are-traces-and-how-sql-yes-sql-and-opentelemetry-can-help-us-get-more-value-out-of-traces-to-build-better-software/.

Zaniolo, Carlo e Miachel A. Meklanoff (mar. de 1981). «On the Design of Relational Database Schemata». Em: *ACM Trans. Database Syst.* 6.1, pp. 1–47. issn: 0362-5915. doi: 10.1145/319540.319542. url: https://doi.org/10.1145/319540.319542.

Apêndice A

Respostas aos Requisitos

Este apêndice apresenta as respostas aos requisitos propostos para este projeto, mencionado quais é que foram cumpridos e os que não foram.

A.1 Requisitos funcionais

Como mencionado na secção 8.2, nem todos os requisitos foram implementados.

A.2 Atributos de qualidade

As presentes tabelas referenciam os atributos de qualidade do projeto, adotando a mesma numeração das listas em cada categoria.

A.2.1 Atributos de funcionalidade

Tabela A.1: Respostas para os atributos de funcionalidade

Número	Progresso	Detalhes
1	Feito	Já existente na versão inicial do Status Monitor.
2	Não feito	A funcionalidade que seria responsável por isto não foi implementada.

A.2.2 Atributos de usabilidade

Tabela A.2: Respostas para os atributos de usabilidade

Número	Progresso	Detalhes
1	Feito	Testado no capítulo 7.

A.2.3 Atributos de confiabilidade

Tabela A.3: Respostas para os atributos de confiabilidade

Número	Progresso	Detalhes
1	Feito	Mencionado no capítulo 6.
2	Feito	Através dos protocolos de comunicação utilizados, como HTTPS.
3	Feito	Através dos protocolos de comunicação utilizados, como HTTPS.

A.2.4 Atributos de desempenho

Tabela A.4: Respostas para os atributos de desempenho

Número	Progresso	Detalhes
1	Feito	Testado no capítulo 7.
2	Feito	Testado no capítulo 7.
3	Não feito	A funcionalidade que seria responsável por isto não foi implementada.

A.2.5 Atributos de suportabilidade

Tabela A.5: Respostas para os atributos de suportabilidade

Número	Progresso	Detalhes
1	Feito	A solução foi testada em diversos navegadores, e os dados são apresentados de forma devida.

A.2.6 Restrições de desenho

Nenhuma restrição foi proposta.

A.2.7 Restrições de implementação

Nenhuma restrição foi proposta.

A.2.8 Restrições de interface

Tabela A.6: Respostas para as restrições de interface

Número	Progresso	Detalhes
1	Feito	Já existente na versão inicial do Status Monitor.
2	Feito	Já existente na versão inicial do Status Monitor.
3	Não Feito	A funcionalidade que seria responsável por isto não foi implementada.

A.2.9 Restrições físicas

Nenhuma restrição foi proposta.