

Databases of Things

Gonçalo Passos - 88864
João Vasconcelos - 89022
Luís Valentim - 93989
Marta Ferreira - 88830
Rita Ferrolho - 88822

Relatório de Projeto em Informática da Licenciatura em Engenharia Informática da Universidade de Aveiro, realizado por Gonçalo Passos, João Vasconcelos, Luís Valentim, Marta Ferreira e Rita Ferrolho sob a orientação de Mário Antunes, Professor Auxiliar do

Departamento de Engenharia, Eletrónica e Telecomunicações da Universidade de Aveiro

Keywords

Database, Internet of things, Machine to machine, Data visualization, Big Data

Abstract

A utilização de IoT é cada vez mais recorrente nos dias de hoje, surgindo assim a necessidade de facilitar a aquisição e consumo de dados de uma maneira o mais abrangente e eficiente possível. As soluções atuais pecam quer seja por performance, escalabilidade, simplicidade de inserção, capacidades de querying, privacidade e/ou um conjunto dos mesmos e é com esse intuito que surge a DataBase of Things, uma solução inovadora para armazenamento de dados IoT. A base de dados colunar Cassandra é um tipo de base de dados que possui boa escalabilidade, permitindo pesquisas e escritas muito rápidas, o que se torna muito interessante no contexto de IoT. No entanto, esta base de dados tem diversas restrições quanto ao tipo de queries suportados, uma vez que possui uma estrutura muito rígida. O objetivo deste trabalho é criar uma extensão desta base de dados, que seja capaz de decompor um ficheiro estruturado (tipicamente estrutura em JSON) num conjunto de tabelas que facilite a sua pesquisa e consumo. As tabelas são criadas de forma dinâmica para acomodar os tipos de dados à medida que os mesmos são inseridos na base de dados. A linguagem que interação do Cassandra (CQL) será estendida para suportar as novas funções desenvolvidas. Além disso, foram desenvolvidos mecanismos de autenticação para suportar múltiplos utilizadores. Finalmente criamos uma API que permite o consumo direto dos dados através de ferramentas de visualização de dados, neste caso o Grafana.

Pretendemos alcançar uma solução IoT, rápida, escalável, distribuída, segura e sem restrições de querying que torna a inserção, procura e visualização de dados mais simples e mais acessível para todos.

Índice

1. Introdução	5
1.1. Contexto	5
1.2. Motivação	6
1.3 Objetivos	6
2. State of the Art	7
2.1 Projetos relacionados	7
2.1.1 Modular and generic IoT management on the cloud (2018)	7
2.1.2 VergeDB: A Database for IoT Analytics on Edge Services (2021)	7
2.1.3 Scalable semantic aware context storage (2016)	7
2.2 Ferramentas Relacionadas	8
3. Requisitos do sistema e Arquitetura	9
3.1 Requisitos do sistema	9
3.1.1 Requisitos Funcionais	9
3.1.2 Requisitos Não Funcionais	10
3.1.2.1 Essenciais	10
3.1.2.2 Prioridade alta	10
3.1.2.3 Prioridade baixa	10
3.2 Actors e Use Cases	10
3.3 System Architecture	13
3.3.1 Inserção de dados	14
3.3.2 Pesquisa de dados	14
3.3.3 Visualização	14
4. Implementation	15
4.1 Data Model	15
4.2 Biblioteca de Interação	17
4.2.1 DB	17
4.2.2 Parser	18
4.2.3 Cache	18
4.3 Web Server	19
4.3.1 Autenticação	20
4.3.1.1 Register	20
4.3.1.2 Login	21
4.3.1.3 Logout	22
4.3.2 Insert	23
4.3.3 Query	23
4.4 Grafana	25
4.4.1 JSON API Grafana Datasource	25

4.4.2 Configuração	25
4.4.3 Criar uma dashboard	27
5. Resultados e Análise da Solução	28
5.1 Testes	28
5.1.1 Planeamento	28
5.1.2 Resultados	28
5.1.2.1 Testes em pequena escala	28
5.1.2.2 Performance	29
5.1.2.3 Escalabilidade	32
5.2 Análise	33
6. Conclusão e Trabalho Futuro	35
7. Referências	36
Apêndices	37
Configurações	37
Cassandra	37
Django	38

1. Introdução

1.1 Contexto

A proliferação de sensores em ambientes inteligentes possibilita a aquisição de dados para uso posterior, o que implica uma estrutura otimizada para processá-los.

Estima-se que venham a existir 62 milhões de conexões IoT até 2024, como podemos observar na figura 1.1.

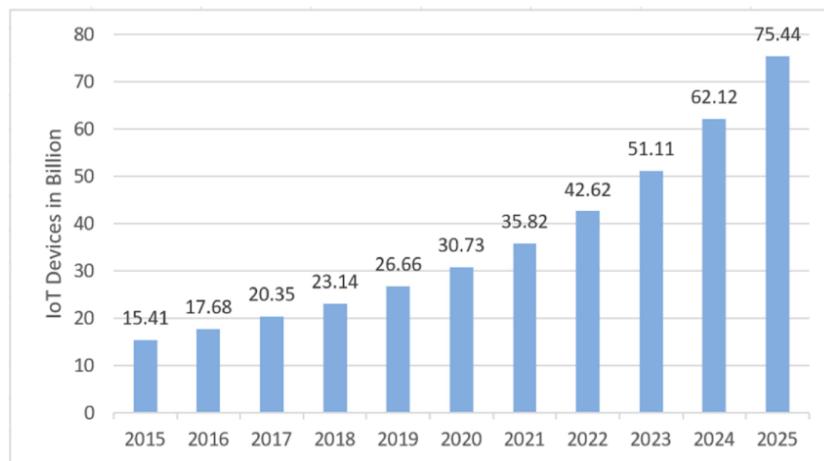


Figura 1.1 - Previsão do uso de sensores até 2025.

À medida que IoT é cada vez mais utilizado, a quantidade de dados gerados por sensores e outros dispositivos aumenta. Para além disso, a variação do tipo de data destes sensores, tal como a velocidade com que estes lidam com informação também aumentou, isto leva a um problema, pois consoante esta evolução, armazenamento e recuperação de dados torna-se difícil.

Apesar de existirem outras alternativas de base de dados, como o *TimescaleDB* e o *InfluxDB*, quase todas apresentam, no geral, restrições de inserção, performance, querying ou uma combinação dos 3 fatores.

O nosso projeto pretende apresentar uma solução agnóstica baseada em Cassandra para aproveitar a sua alta performance e escalabilidade mas ultrapassando as restrições de

CQL, no entanto uma solução absolutamente agnóstica não é viável, nesse sentido utilizamos um token para identificar o utilizador e um inteiro para definir o sensor no qual o utilizador quer inserir o que nos pareceu a melhor forma de conjugar inserção agnóstica com utilidade a nível prática e guardamos num conjunto de tabelas específico de acordo com o nosso modelo, as tabelas relativas a atributos são criadas dinamicamente conforme os atributos do ficheiro json inserido. Possui ainda uma UI na web para inserções e queries manuais que seria secundária, uma vez que a comunicação numa situação real de produção seria efetuada diretamente através dos endpoints. A visualização de dados na forma de gráficos timeseries pode ainda ser efetuada através do serviço grafana associado ao nosso sistema.

1.2 Motivação

- Cenários de IoT requerem capacidades alargadas a nível de processamento, análise e armazenamento de dados.
- Diferentes tipos de sensores podem publicar dados estruturados de forma diferente.
- Não existe uma representação uniforme para formatação de dados enviados pelos sensores.
- Explorar uma solução agnóstica para este problema.

1.3 Objetivos

- Parsing e inserção de dados de forma o mais agnóstica possível para ficheiros JSON.
- Desenvolver um modelo de dados que guarde a informação após esta ser processada de forma a facilitar a análise e visualização dos mesmos.
- Estender a linguagem CQL de forma a suportar querying mais complexo, como range querying, filtragem e agregações sem as restrições associadas a CQL.
- Desenvolver uma ferramenta de análise e visualização de dados.
- Assegurar a privacidade dos dados dos utilizadores.

2. State of the Art

2.1 Projetos relacionados

2.1.1 Modular and generic IoT management on the cloud (2018)

Este projeto consiste na criação de um *data collection service* para suportar dados genéricos de IoT. Permite suportar tráfego de entrada rápido e separar dispositivos IoT e cloud system com suporte de armazenamento de dados NoSQL. Oferece uma solução que permite armazenar e recolher de uma forma rápida e eficiente dados de IoT num cloud system.

2.1.2 VergeDB: A Database for IoT Analytics on Edge Services (2021)

Consiste numa base de dados que comprime dados IoT que suporta tarefas de análise complexas, recorrendo a machine learning para isso.

2.1.3 Scalable semantic aware context storage (2016)

Paper que estuda formas eficientes de lidar com informação com modelos de representação diferentes e propõe um novo modelo de representação d-dimension. Concluiu-se que este modelo melhorou a escalabilidade e a extração semântica.

2.2 Ferramentas Relacionadas

O *Elasticsearch* é um mecanismo de busca e análise de dados distribuído, gratuito e aberto para todos os tipos de dados, incluindo textuais, numéricos, geoespaciais, estruturados e não estruturados. O *Elasticsearch* é desenvolvido sobre o Apache Lucene e foi lançado pela primeira vez em 2010 pela *Elasticsearch N.V.* (agora conhecida como *Elastic*). Conhecido pelas suas REST APIs simples, a sua natureza distribuída, velocidade e escalabilidade, o *Elasticsearch* é o componente central do *Elastic Stack*, um conjunto de ferramentas gratuitas e abertas para ingestão, enriquecimento, armazenamento, análise e visualização de dados. Comumente chamado de *ELK Stack* (pelas iniciais de *Elasticsearch*, *Logstash* e *Kibana*), o Elastic Stack agora inclui uma rica coleção de agentes lightweight conhecidos como *Beats* para enviar dados ao Elasticsearch. É basicamente um search and analytics engine para todos os tipos de dados, a componente central do Elastic Stack.

No entanto tem as suas limitações: não oferece compatibilidade com versões anteriores, requer uma configuração inicial complexa, o que contribui para uma curva de aprendizagem elevada, até para *developers*. Além disso, é importante referir que, sendo um mecanismo de busca e não de *querying*, apresenta a sua limitação a nível de precisão de dados. O seu logótipo é apresentado na figura 2.1.



Figura 2.1 - Logótipo do *ElasticSearch*.

Elassandra é um *fork* do Elasticsearch modificado para rodar no Apache Cassandra numa arquitetura peer-to-peer escalável e resiliente. O código do Elasticsearch está embutido nos nós do Cassandra, fornecendo recursos de pesquisa avançada nas tabelas do Cassandra e o Cassandra serve como um armazenamento de dados e configuração do Elasticsearch. *Elassandra* é, basicamente, Elasticsearch implementado em cima de Cassandra, integrando Cassandra DB com o motor de busca Elasticsearch, e permite assim, escalabilidade horizontal. No entanto, assim como Elasticsearch, apresenta uma limitação relativamente à precisão dos dados. O logótipo do *Elessandra* é mostrado na figura 2.2.



Figura 2.2 - Logótipo do *Elessandra*.

Apache Cassandra, cujo logótipo é apresentado na figura 2.3, é uma base de dados distribuída NoSQL de código aberto confiável por milhares de empresas para escalabilidade e alta disponibilidade sem comprometer o desempenho. Escalabilidade linear e tolerância a falhas comprovada em hardware comum ou infraestrutura em nuvem tornam-no a plataforma perfeita para dados de missão crítica.



Figura 2.3 - Logótipo do Cassandra.

Grafana é uma aplicação web de análise de código aberto multiplataforma e visualização interativa da web. Ele fornece tabelas, gráficos e alertas para a Web quando conectado a fontes de dados suportadas. É expansível através de um sistema de plug-in. Os usuários finais podem criar painéis de monitoramento complexos usando criadores de consultas interativas. A figura 2.4 apresenta o logótipo do Grafana.



Figura 2.4 - Logótipo do Grafana.

3. Requisitos do sistema e Arquitetura

3.1 Requisitos do sistema

3.1.1 Requisitos Funcionais

O sistema necessita de capacidades de inscrição e login, através de uma api rest com token authentication para identificar os utilizadores e associado ao sistema de autenticação do django através da biblioteca de interação. Para além disso necessita de capacidades de inserção de dados de forma o mais agnóstica possível, na nossa solução sendo apenas necessário o ficheiro json, o token que identifica o utilizador e garante a privacidade do sistema e um inteiro representativo do sensor no qual queremos associar os dados, os dados presentes no json serão decompostos e inseridos nas tabelas apropriadas segundo o desenho do nosso modelo de dados. Era também essencial estender as capacidades de querying limitadas de CQL, nomeadamente através do desenvolvimento de filtragem por qualquer atributo, range

querying particularmente eficiente no valor de timestamp que estará sempre presente mesmo que não seja especificado no json, capacidade de efetuar agregações simples e queries específicas por sensor e utilizador. Por último mas não menos importante capacidade de visualizar dados de forma simples para o utilizador e com capacidade de filtragem sobre esses mesmos dados para adaptar a visualização às necessidades do utilizador, integramos endpoints de forma a permitir utilizar o serviço grafana com json simpod datasource para visualização e análise de dados.

3.1.2 Requisitos Não Funcionais

3.1.2.1 Essenciais

- Alta escalabilidade
Cenários IoT requerem capacidade para armazenar big data e escalar horizontalmente, possibilitado pela arquitetura de Cassandra.
- Privacidade de dados
Assegurar a privacidade dos dados do utilizador através da separação dos dados por keyspaces individuais e utilizando os sistemas de autenticação e permissões de Cassandra

3.1.2.2 Prioridade alta

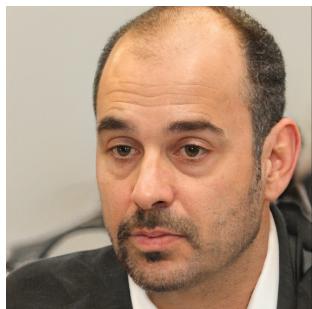
- Maintainability
O sistema deve ser modular e capaz de se adaptar facilmente às necessidades que a userbase impõe.
- Alta performance
É esperada uma boa performance quer na inserção quer no querying dos dados.

3.1.2.3 Prioridade baixa

- Facilidade de aprender/utilizar
O produto deve ser o mais intuitivo possível de forma a proporcionar uma boa experiência ao utilizador, no entanto é pressuposto que um utilizador do nosso sistema possui algum conhecimento na área e é capaz de compreender documentação relativa à área. Para além disso o sistema foi idealizado para ambientes de produção em que a comunicação é feita diretamente através dos endpoints. Foi criado um user manual para facilitar este processo.

3.2 Actors e Use Cases

André Andrade



André Andrade tem 45 anos e é dono de uma ETAR. Com o intuito de reabilitar a ETAR, pretende investir nas tecnologias de monitorização da qualidade de água. Está interessado em recolher e analisar dados relacionados com a pressão da água, provenientes de sensores IoT.

Cenário: No dia 16 de Abril, houve maior precipitação do que o previsto. O Andrade estava prestes a analisar o estado da água com o auxílio de DBoT quando se depara com uma notificação a alertar que a pressão da água estava acima de 80 psi. Andrade clica na notificação e vai parar à página da tabela da pressão da água. De seguida, selecciona o histórico de todos os dados de pressão recolhidos, organizados numa tabela, do mais recente ao mais antigo. Andrade confirma que os dados recentes da pressão da água eram superiores ao limite 80 psi, sugerindo que a água provavelmente deixou de ser potável.

Bárbara Brás



Bárbara Brás tem 25 anos e é engenheira civil numa empresa de infraestruturas na cidade do Porto. Ultimamente, com o aumento de trânsito rodoviário, tem surgido um aumento no número de acidentes rodoviários por semana. Com o intuito de reduzir o número de acidentes, Brás planeia adicionar ou reposicionar semáforos nos lugares que sejam o mais estratégicos possível. Para alcançar este objetivo, Brás necessita de recolher e analisar dados relacionados com o movimento rodoviário. Estes dados vão ser gerados por sensores IoT espalhados por diversas ruas da cidade.

Cenário: Sendo a Bárbara utilizadora do DBoT há quatro meses, decide consultar o número total de transportes detectados em cada sensor, por ordem decrescente. Ao analisar a tabela com os resultados, Brás surpreende-se com o resultado obtido no primeiro sensor, que era bastante superior aos resultados dos outros, deduzindo assim que era necessário colocar um semáforo perto daquela área.

Carlos Cardoso



Carlos Cardoso tem 43 anos e ficou recentemente desempregado devido à pandemia. Desde aí, tem passado por grandes dificuldades financeiras. Sendo assim, Cardoso tenciona usar um sistema de recolha e análise de dados, com o intuito de adquirir uma melhor noção do gasto mensal de água e energia.

Cenário: No dia 30 de junho, um mês após utilizar o DBoT, Cardoso decide comparar, num gráfico de barras, a quantidade de água e energia gasta em maio com a de junho. Para esse efeito, Cardoso selecciona os meses maio e junho, selecciona as componentes água e energia e escolhe o gráfico como tipo de visualização. Após interpretar os resultados, Cardoso conclui que gastou menos água e energia naquele mês do que no mês anterior.

A figura 3.1 é um fluxograma baseado nas pessoas:

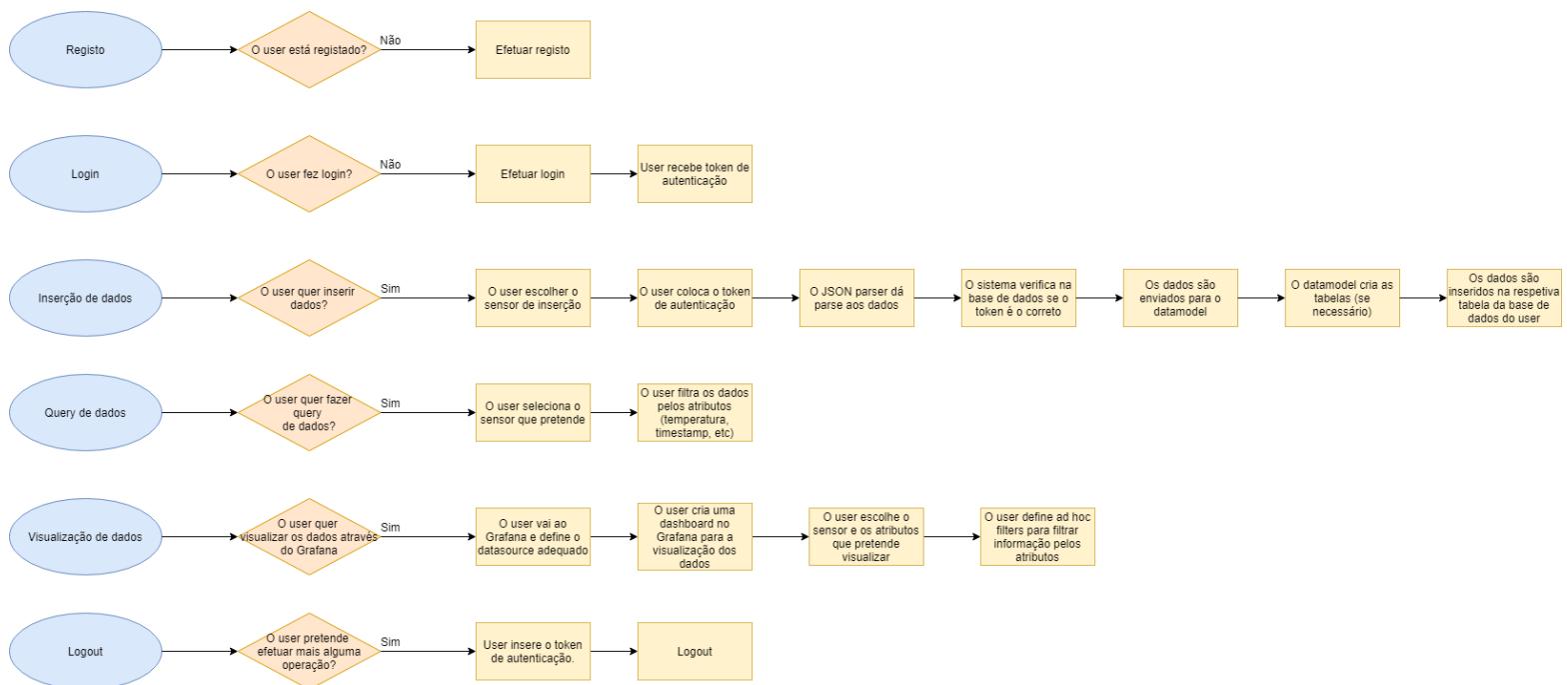


Figura 3.1 - Fluxograma baseado nas pessoas.

3.3 System Architecture

Nesta secção apresenta-se a arquitetura desenhada do nosso software, presente na figura 3.2. Esta é composta pelo frontend, backend, base de dados e um sistema de geração de dados. Com esta arquitetura, planeava-se a inserção de dados gerados automaticamente, a inserção de dados manualmente através da UI (interface gráfica) com credenciais e a pesquisa de dados pela UI.



Figura 3.2 - Arquitetura do sistema DBoT desenvolvido¹

¹ Cassandra está especificado na arquitetura visto que faz parte dos requisitos base do projeto.

3.3.1 Inserção de dados

1. User escreve numa caixa de texto o JSON que pretende inserir ou contacta diretamente o endpoint de inserção.
2. A API, implementada com Django e Django REST framework, gera um pedido POST para o endpoint de inserção com o json correspondente.
3. Os dados passam pelo parser de forma a ser decomposto num dicionário python que é enviado ao método de inserção da biblioteca de interação com a base de dados.
4. O método de inserção irá decompor a informação por campo e registar a informação nas tabelas apropriadas conforme o nosso modelo.

3.3.2 Pesquisa de dados

1. User interage com a UI, podendo procurar por um id de um sensor ou mais, por exemplo, e outros atributos.
2. A API envia pedido POST para o endpoint de procura que irá decompor o pedido de forma a entender a modalidade de query adequada e invoca o método da biblioteca de interação apropriado.
3. O método irá fazer os comandos CQL necessários de forma a recolher e organizar a informação de acordo com as especificações do pedido.
4. A resposta é retornada na forma de um array de dicionários, cada dicionário representa um registo, através do endpoint
5. Se o pedido foi feito através da UI esta irá apresentar cada um dos registos resposta ao utilizador

3.3.3 Visualização

1. O utilizador entra no serviço grafana que será lançado com a nossa aplicação
2. Utiliza o endpoint grafana, protegido com o seu token de autenticação para configurar um datasource json simpod
3. O serviço grafana apresenta uma UI especializada na qual um utilizador pode definir opções de visualização e querying.
4. É formado um gráfico time series com as especificações passadas através do grafana para a nossa rest framework.

4. Implementation

As escolhas que trouxeram este projeto à vida foram deliberadas durante as fases de conceptualização, como já foi referido Cassandra não foi uma escolha visto que era um dos requisitos base do projeto no entanto é certamente a tecnologia exterior mais impactante à nossa solução visto que grande parte das nossas maiores valias de performance, escalabilidade, para além disso as capacidades de autenticação do mesmo também foram cruciais para a nossa solução. O modelo de dados foi desenvolvido por nós e foi em grande parte influenciado pela pesquisa de serviços semelhantes na área de IoT e por discussão com o orientador que foi uma grande ajuda para percebermos que caminhos tomar e que funcionalidades possuíam melhor ou menor interesse no nosso caso. A biblioteca pretendia conferir maior modalidade ao projeto tornando o nosso trabalho adaptável para outras API's no futuro, no entanto devido a dificuldades de tempo e organização de trabalho acabou por ficar mais dependente da API do que tinha sido idealizado inicialmente ainda assim é um módulo externo que pode ser replicado para outros projetos tendo em conta algumas restrições. A framework para web projects python com que nos sentimos mais confortáveis devido à simplicidade e documentação disponível foi o Django que consideramos ser simples, intuitivo e mais que capaz de implementar tudo o que tínhamos em mente. Grafana foi escolhido como ferramenta de visualização devido ao datasource json simpod que permitiu uma integração simples com a nossa base de dados, para além disso as capacidades de mostrar gráficos timeseries com opções de filtragem fáceis de utilizar e customização extensa foram uma grande mais valia que consideramos ideal para a visualização de dados IoT.

4.1 Data Model

O Data Model ou modelo de dados é o modelo que seguimos para guardar a informação dos registos enviados de forma a poder estender a linguagem CQL e retirar facilmente os dados necessários para as funcionalidades da API e está organizado da seguinte forma. O desenvolvimento deste foi fortemente iterativo e associado a testes funcionais com grande influência de discussão de resultados com o orientador do projeto.

Tabelas:

- Tabela 4.1 - metadata

Attribute	Timestamp	Pk
PRIMARY KEY (attribute, timestamp, pk)		

- Tabela 4.2 - metadata_reverse

Attribute	Pk	Timestamp
PRIMARY KEY (attribute, pk, timestamp)		

- Tabela 4.3 - sensors

User	Sensor_id	Pk	Attributes
PRIMARY KEY (user, sensor_id, pk)			

- Tabela 4.4 - Attribute_table

Pk	Attribute
PRIMARY KEY (pk, attribute)	

Através desta estrutura, apresentada nas tabelas 4.1 a 4.4, podemos alcançar range queries, e agregações com boa performance em qualquer registo e através de múltiplas formatações, desta forma é possível ter acesso aos dados por atributo e não por registo, permitindo por exemplo que um utilizador procure por registos com x valor de temperatura este poderá obter informação relativa a todos os registos de sensores que possuíssem o atributo temperatura no entanto se procurar por pressão atmosférica para registos com temperatura superior a x, só serão apresentados resultados de registos que possuem simultaneamente temperatura e pressão atmosférica.

Devido à área em causa o sistema atribui timestamps aos registos que não o possuíam anteriormente, para além disso os métodos de interação desenvolvidos têm em conta que o modelo permite range querying particularmente eficiente no atributo timestamp.

O projeto ao longo do seu desenvolvimento veio a afastar-se um pouco do conceito inicial, uma vez que como grupo considerámos segurança um requisito indispensável para uma base de dados e não estava inicialmente previsto no conceito do projeto, isto obrigou a uma mudança drástica do nosso sistema uma vez que para assegurar a privacidade dos dados foi necessário separar o que deveria ser uma base de dados comum a todos os utilizadores, em keyspaces particulares por utilizador e implementar o mecanismo de autenticação de Cassandra. Esta mudança implicou a replicação do modelo de dados para cada utilizador, o que como será discutido em mais detalhe nos resultados, trouxe complicações a nível de performance que não tinham sido previstas. O modelo de dados foi otimizado para a solução inicial, daí existir uma certa dualidade no projeto entre o trabalho exploratório desenvolvido sobre o conceito inicial que deu origem ao data model e o produto final que foi produzido com preocupações de segurança.

4.2 Biblioteca de Interação

Foi desenvolvida uma biblioteca com as ferramentas necessárias para interagir com a base de dados, a biblioteca possui todos os elementos necessários para possibilitar transformar registos jsons em dicionários flattened/sem recursividade, armazená-los seguindo a estrutura do nosso modelo de dados e posteriormente pesquisar ditos registos com capacidades de filtragem e agregação superiores às de CQL, para além dos métodos centrais a este processo inclui ainda todos os elementos necessários para suportar este processo como a implementação de uma Cache simples para gestão de sessões o próprio parser de JSON, e métodos de apoio à integração de grafana e funções da API.²

O processo de autenticação ter sido feito em parte através da nossa API e não completamente através desta biblioteca foi um erro imprevisto que gerou uma vulnerabilidade de segurança bastante séria uma vez que impossibilitou a eliminação do super utilizador Cassandra, permitindo qualquer pessoa com algum conhecimento de Cassandra aceder à base de dados com credenciais de super utilizador, todo o processo de autenticação do utilizador deveria ser traduzido para a biblioteca.

Documentação específica dos métodos presente no README.md do repositório.

A biblioteca possui os seguintes componentes:

4.2.1 DB

Conjunto de métodos que interagem com o nosso cluster de Cassandra. Está separado em cinco grandes secções.

Roles and Personal Keyspace Handling onde estão os métodos para criar o keyspace e role necessários bem como as tabelas básicas para o funcionamento do keyspace e autenticação do utilizador através de Cassandra.

Insertion Functions conjunto de métodos que ao receber um conjunto de jsons flattened organiza a informação contida nestes tabelas que se necessário serão dinamicamente criadas com atenção ao tipo de dados em causa.

Query Functions os diferentes métodos de querying, 4 métodos que foram desenvolvidos separadamente de forma a otimizar conforme o tipo de pedido pretendido, dividem-se entre pedidos por utilizador ou por sensor específico e com ou sem range de timestamps específica. As queries obtém o seu resultado cruzando os pks associados ao utilizador ou sensor com todos os pks de todos os atributos presentes na query, organiza num dicionário atributo:pk e utiliza as subqueries de forma a identificar quais destes pks passam as condições de filtragem, os pks que estiverem simultaneamente em todas as condições são os registos amostrados. Foi pensado com a base de dados comum inicial em mente.

Aggregation Handling, uma série de métodos que permitem utilizar 5 agregações, soma, média, número de elementos, valor máximo e mínimo.

² Re却tório da biblioteca de interação - https://github.com/LuisValentim1/PI_DBOT_Lib

Support Functions, métodos de suporte à API, retornam informação relevante para a utilização de um sistema de gestão da base de dados, por exemplo associação utilizador e os seus atributos.

4.2.2 Parser

O JSON Parser transforma os registos JSON enviados pelo utilizador em flattened jsons, dicionários com todos os atributos do json e sem recursividade, de forma a poderem ser utilizados pela DB, conforme mostrado na figura 4.1.

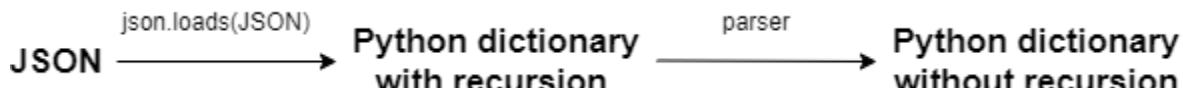


Figura 4.1 - Funcionamento do JSON parser.

4.2.3 Cache

Cache simples para gestão de sessões de Cassandra ativa, limita o número de utilizadores que podem estar a interagir ao mesmo tempo mas é necessário pois as sessões colocam stress no sistema e geram baixas de performance, cada sessão expira após 50 minutos ou caso seja a mais antiga e existam outros pedidos para iniciar novas sessões, no entanto a recuperação da sessão é automática pelo que os utilizadores não irão perder acesso a não ser que exista um número inesperadamente alto de utilizadores a aceder ao mesmo tempo. O tamanho do Cache pode ser alterado na sua inicialização, mas uma vez deployd não será alterável. O fluxograma que representa a cache é ilustrado na figura 4.2.

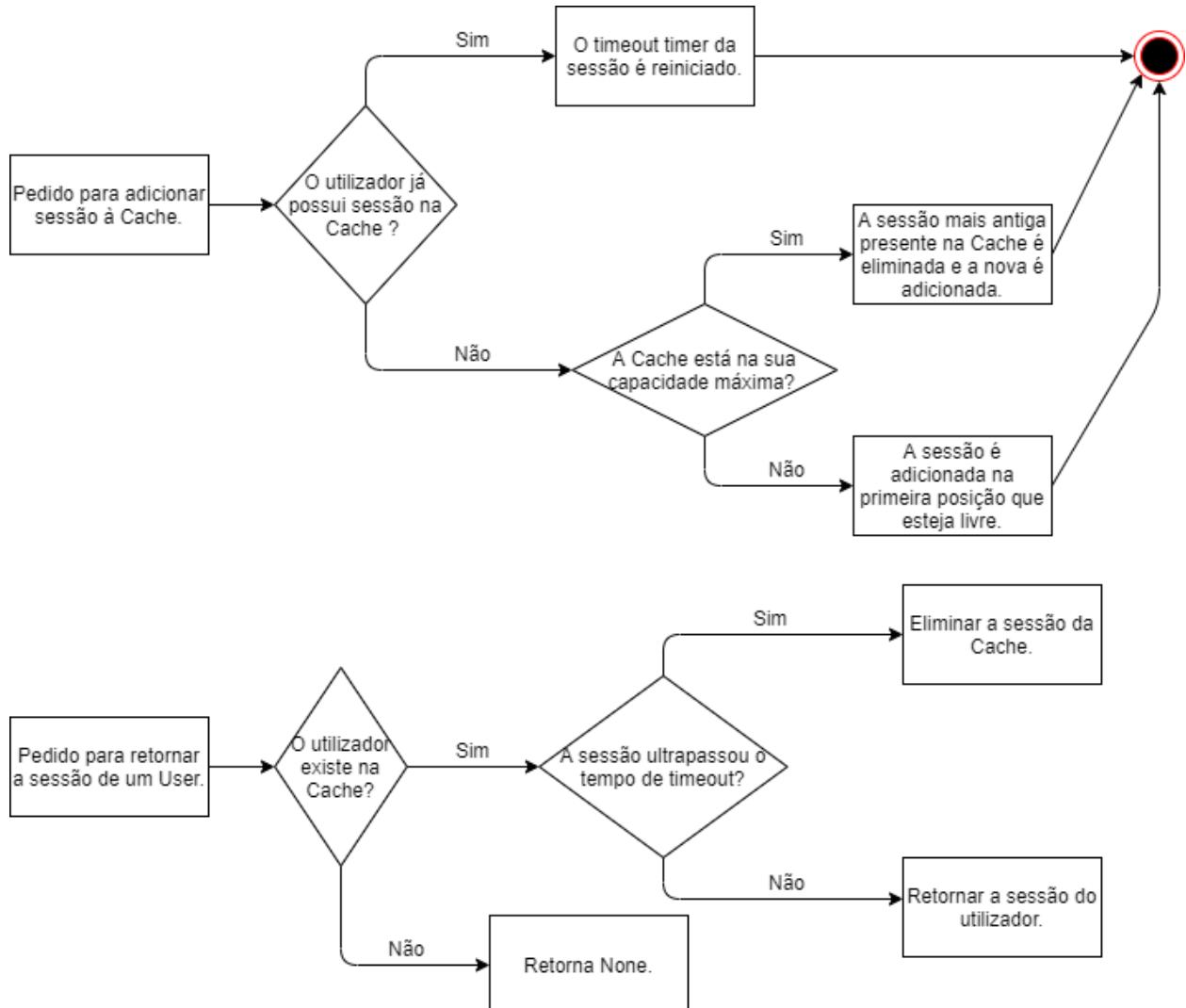


Figura 4.2 - Fluxograma de funcionamento da cache.

4.3 Web Server

O **Django** é uma framework em python usada para desenvolver aplicações Web de forma rápida e limpa. O **Django Rest Framework** é um conjunto de ferramentas usadas para desenvolver Web APIs. No caso da nossa webapp, no que toca a RESTful API, todos os endpoints usados consistem em métodos **GET** e **POST**.

Os **modelos** da webapp são guardados numa base de dados chamada 'DB' que é criada/atualizada quando é executado o comando "python manage.py sync_cassandra".

Normalmente seria usado o comando “python manage.py migrate”, mas neste caso a base de dados trata-se de uma base de dados cassandra.

Estes modelos apenas são usados para a autenticação, consistem em tabelas de utilizadores e tokens em que maior parte dos dados são encriptados, de forma a ser garantida bastante segurança ao utilizador no uso da aplicação. Quanto ao resto dos dados, são manuseados através do **DataModel**, sendo que a nossa Framework utiliza os endpoints de forma a enviar os valores de forma correta e identificando o utilizador que está autenticado.

4.3.1 Autenticação

A autenticação pode ser dividida em três partes: **register**, **login** e **logout**. Cada um destes métodos tem um endpoint atribuído de modo a comunicar com o datamodel e realizar as operações necessárias.

4.3.1.1 Register

O **register**, cuja página é mostrada na figura 4.3, consiste em criar uma nova conta. O utilizador fornece um **nome**, um **email** e uma **password**. O nome de utilizador tal como o email têm que ser únicos, caso contrário (se existir alguma conta com um dos dados iguais) a operação não é permitida e uma mensagem é apresentada ao utilizador.

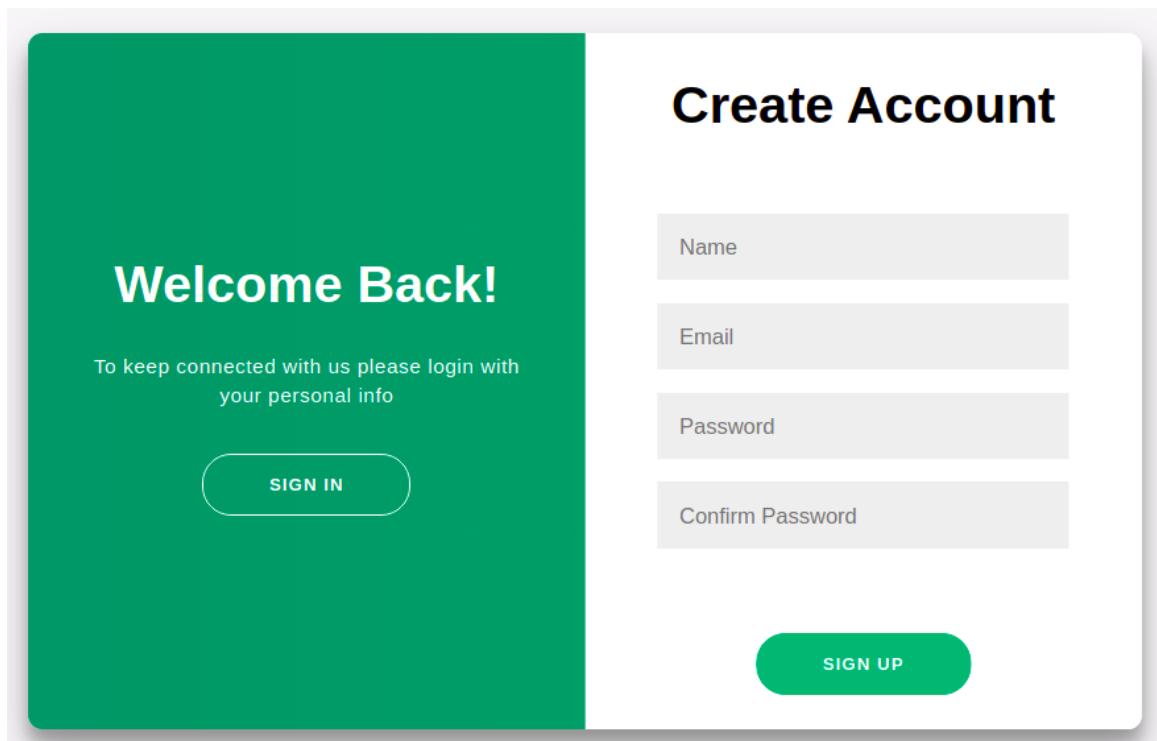


Figura 4.3 - Página *register*.

O nome é necessário ser único pois, ao criarmos uma conta, é criada uma base de dados, tal como uma role com o nome do utilizador. O mail também é necessário ser único pois quando formos realizar operações como o insert e o query, iremos precisar dele para confirmar se o token que o utilizador fornece é o correto ou não. Usamos o email para esta confirmação e não o nome, pois o email é encriptado na base de dados, não sendo possível uma pessoa com acesso à base de dados saber que token está atribuído a que utilizador. Para além disso, o token também é encriptado quando inserido na base de dados, sendo então a única forma de o obter logo após o login (quando o token é apresentado ao utilizador, o utilizador encontra-se num url com o próprio token, impedindo outra pessoa de saber esse endereço).

4.3.1.2 Login

O **login**, cuja página é mostrada na figura 4.4, consiste em autenticar a conta do utilizador e iniciar sessão fornecendo o email e password.

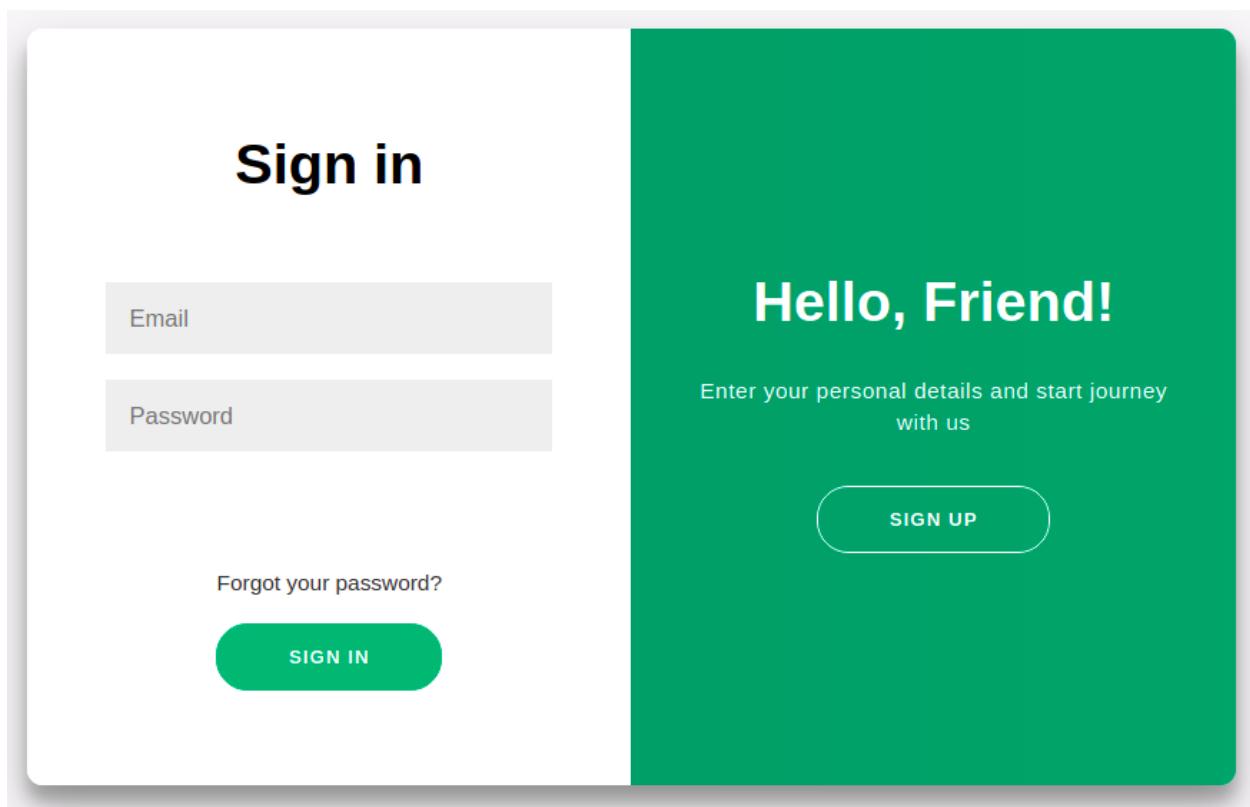


Figura 4.4 - Página *login*.

Se os dados estiverem corretos, o utilizador é autenticado na sua base de dados com a sua role (tanto a base de dados como a role foram criadas quando o utilizador se registou).

Após isto, um token é gerado (e encriptado) e inserido numa tabela chamada **TokensTable**. Nesta tabela é inserido o token tal como o email, também encriptado. Se o email

já estivesse na tabela (no caso de o logout não ter sido efetuado com sucesso), esse dado é apagado e é inserido um novo com estes valores.

Finalmente o utilizador é redirecionado para uma página em que lhe é apresentado o **token** que irá de ter de usar para efetuar as operações **insert**, **query** ou **logout**, como é demonstrado na figura 4.5.

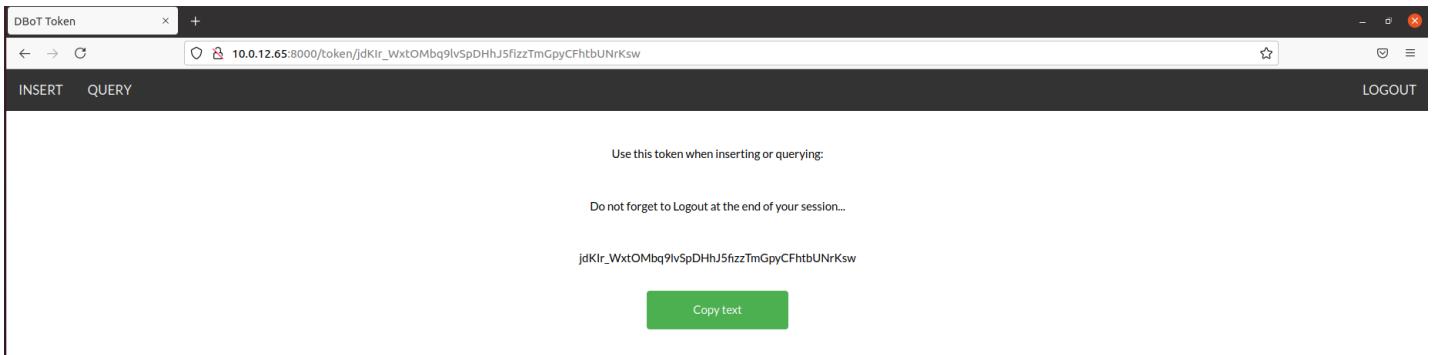


Figura 4.5 - Página em que o *token* é apresentado.

4.3.1.3 Logout

O logout é uma operação que pode ser realizada a partir do momento em que o utilizador tem acesso ao token e é uma das três opções presentes na navbar. O query e o insert estão localizados no lado esquerdo da navbar enquanto que o logout está do lado direito. Um *screenshot* desta operação é mostrado na figura 4.6.

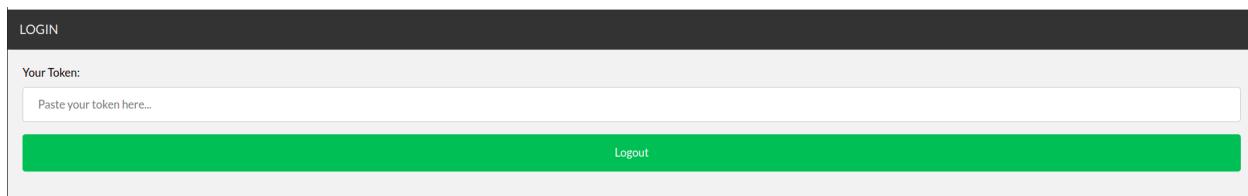


Figura 4.6 - *Screenshot* de logout.

Nesta página o utilizador insere o seu token e clica no botão. Ao clicar no botão, o token deixa de ser válido (os valores são apagados da tabela **TokensTable**) e o utilizador é redirecionado para a página inicial (onde o login e o register são efetuados).

4.3.2 Insert

Na página do *insert*, que corresponde à figura 4.7, são pedidos três campos: o **sensorid**, o **token** e os **dados** a inserir.

Os dados tanto podem ser inseridos dentro de uma lista como não, sendo que cada valor tem que estar separado por uma vírgula e em formato de dicionário. O datamodel irá criar uma tabela para cada um dos atributos inseridos (se ainda não tiverem sido criadas anteriormente), e adicionar os valores. Caso o token esteja incorreto o utilizador é informado e o formulário não é submetido.

O formato de como os dados são inseridos é possível ser consultado no user manual³.

The screenshot shows a web interface titled 'insert'. At the top, there are two tabs: 'INSERT' (which is active) and 'QUERY'. On the right side of the header is a 'LOGOUT' link. Below the tabs, there are three input fields: 'Sensor ID' (placeholder: 'Write Sensor ID here...'), 'Your Token' (placeholder: 'Paste your token here...'), and 'JSON input:' (placeholder: 'JSON input..'). At the bottom of the page is a green horizontal button labeled 'Insert'.

Figura 4.7 - Página de *insert*.

4.3.3 Query

Na figura 4.8 é apresentado um dos *layouts* da página de *query*.

The screenshot shows a web interface titled 'query'. At the top, there are two tabs: 'INSERT' (disabled) and 'QUERY' (active). On the right side of the header is a 'LOGOUT' link. Below the tabs, there are several input fields: 'Your Token' (placeholder: 'Paste your token here...'), 'Sensor target' (dropdown menu showing 'All'), 'Attributes to be shown:', 'Conditions:', 'From:' (date input: 'mm/dd/yyyy') with a calendar icon, 'To:' (date input: 'mm/dd/yyyy') with a calendar icon, and a 'Load Attributes' button. At the bottom of the page is a green horizontal button labeled 'Query'.

Figura 4.8 - Página de *query*.

O utilizador primeiro insere o token e clica no botão “Load Attributes”, o que este botão faz é listar todos os sensores presentes na base de dados do utilizador. Além disso, ao clicar neste botão a página também é atualizada com os atributos de todos os sensores. Isto acontece pois consoante a opção que o utilizador escolhe no dropdown “Sensor target”, é

³ User Manual - <https://github.com/ritamf/PI/blob/main/README.md>

mostrado os atributos desse sensor. Como a opção “All” está sempre presente (incluindo antes do token ser validado) a página é atualizada com todos os atributos.

Caso o utilizador selecione outro sensor target, apenas lhe irão ser apresentados atributos desse sensor.

A partir daqui o utilizador pode escolher entre fazer queries simples ou mais complexas. Os “Attributes to be shown”, que pode ser lido na figura 4.9, são checkboxes com os atributos que lhe irão ser apresentados no resultado. Se o utilizador selecionar o sensor “1” e escolher o atributo “temperature”, vão-lhe ser apresentados todos os valores “temperature” desse sensor.

Para efetuar queries mais complexas, o utilizador pode especificar condições tal como range queries (de tempo).

Nas condições há a opção de fazer uma ou duas condições por atributo (ou nenhuma se não especificar valor). Por exemplo, é possível obter todos os valores de “temperature” maiores que 5 (>5) e menor ou iguais a 10 (≤ 10).

Em relação às datas, também é possível obter resultados dentro de um certo período de tempo. Por exemplo, podemos obter os valores entre o dia 6 de Junho de 2020 e 7 de Junho de 2020. Não há opção para especificar horas/minutos/segundos mas é possível fazê-lo através do grafana.

O formato de como os dados são enviados para efetuar a operação de query é possível ser consultado no user manual⁴.

The screenshot shows a user interface for querying data. At the top, there are tabs for 'INSERT' and 'QUERY', with 'QUERY' being active. On the right side of the header are 'LOGOUT' and 'Load Attributes' buttons. The main area contains the following fields:

- Your Token:** A text input field containing the value: jdKlr.WxtOMBq9lvSpDHhJ5fizzTrnGpyCFhtbUNrKsw.
- Sensor target:** A dropdown menu set to '1'.
- Attributes to be shown:** Two checkboxes: 'sensorid' and 'temperature'.
- Conditions:** Two sets of comparison operators and value input fields:
 - For 'sensorid': Operator >, Value input field 'write the value here'.
 - For 'temperature': Operator >, Value input field 'write the value here'.
 - For 'sensorid': Operator >, Value input field 'write the value here'.
 - For 'temperature': Operator >, Value input field 'write the value here'.
- From:** A date input field with placeholder 'mm/dd/yyyy'.
- To:** A date input field with placeholder 'mm/dd/yyyy'.
- Query:** A large green button at the bottom.

Figura 4.9 - Outro layout da página de query.

⁴ User Manual - <https://github.com/ritamf/PI/blob/main/README.md>

4.4. Grafana

O Grafana é uma aplicação web usada para a visualização interativa de dados. Esta visualização é normalmente feita em formato de gráficos, mas também pode ser feita através de gauges e tabelas.

4.4.1 JSON API Grafana Datasource

O “*JSON API Grafana Datasource*” consiste em desenvolver vários endpoints de forma a depois o grafana comunicar com a webapp e obter os valores necessários para disponibilizar gráficos ao utilizador. O data source executa pedidos em back-ends arbitrários e analisa a resposta JSON em data frames Grafana. De modo a usarmos esta data source, foi preciso implementarmos quatro endpoints no nosso backend:

- “/” (GET) - retorna uma resposta HTTP 200, é usado para testar a conexão na página de configuração do data source.
- “/search” (POST) - retorna as métricas quando invocado (usado para escolher os valores que irão aparecer no gráfico).
- “/query” (POST) - retorna as métricas dependendo do input (tem em atenção os adhoc filters e o período de tempo).
- “/annotations” (POST) - retorna as anotações.

Para além disso foram implementados mais dois endpoints opcionais:

- “/tag-keys” (POST) - retorna as tag keys para os ad hoc filters.
- “/tag-values” (POST) - retorna as tag values para os ad hoc filters.

4.4.2 Configuração

De forma a utilizar o grafana, o utilizador tem primeiro que se dirigir ao porto 3000 e adicionar o JSON datasource. Para adicionar o datasource é preciso o utilizador ir a “Configuration” e selecionar a opção “Datasource” onde ele irá escolher a opção “JSON”. Após isto é necessário configurar o url por onde o grafana irá comunicar com a webapp. Este url será do formato: “<http://10.0.12.65:8000/><str:user_token>/grafana” onde <str:user_token> é o token fornecido ao utilizador após realizar o login. O layout do grafana durante este procedimento é demonstrado na figura 4.10.

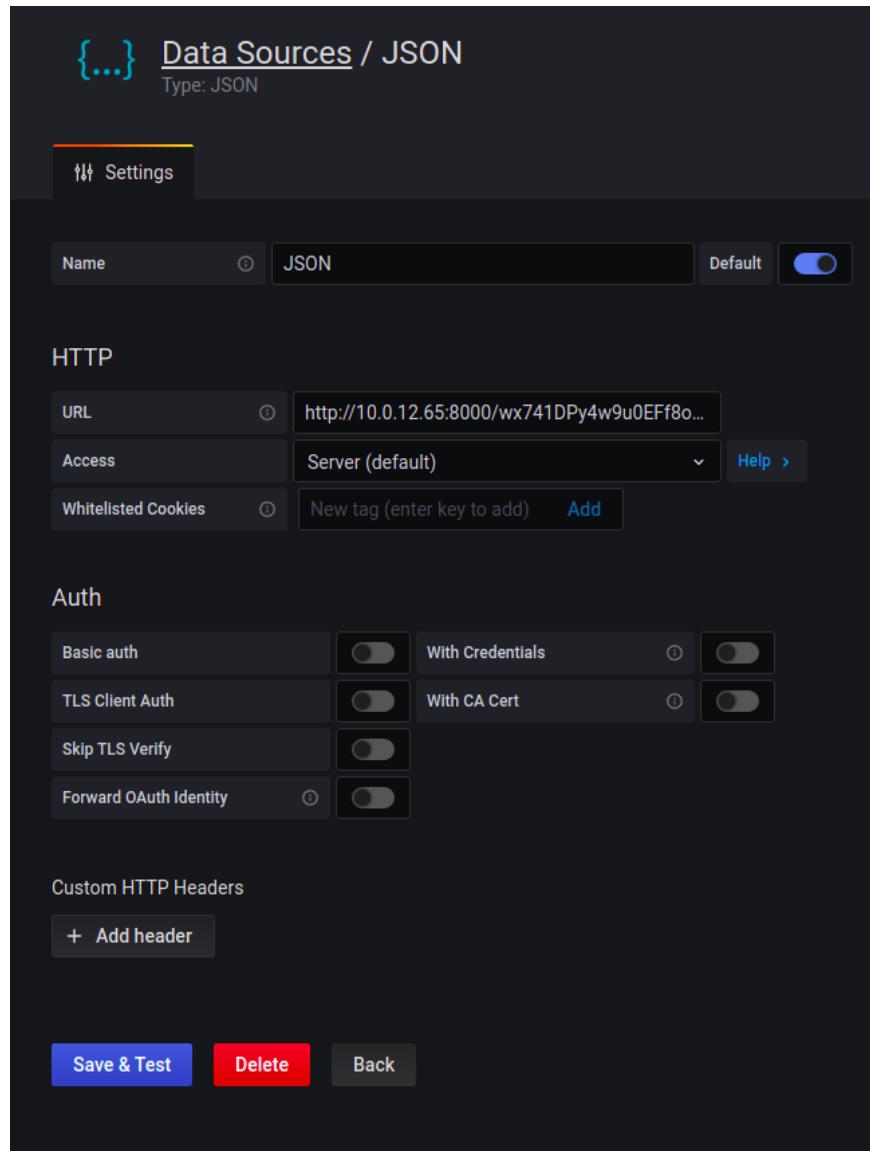


Figura 4.10 -Layout da página de configuração de Grafana.

Após o url ser escrito, ao clicar no botão “Save & Test”, o grafana irá comunicar com a aplicação pela primeira vez utilizando o endpoint “<str:user_token>/grafana/”, este endpoint é para confirmar que a aplicação está a funcionar como é devida e deverá retornar uma resposta HTTP 200. Caso seja esse o caso, será apresentada uma mensagem a dizer “Data source is working”.

4.4.3 Criar uma dashboard

Após o utilizador acabar de configurar o data source, é possível criar uma dashboard. Para isto é preciso o utilizador dirigir-se a “Create”, selecionar a opção “Dashboard” e clicar em “Add an empty panel”.

Por forma a ver os valores a serem mostrados no gráfico, é necessário selecionar uma das opções do dropdown menu “Metric”, isto é uma lista retornada do endpoint “`<str:user_token>/grafana/search`” com os atributos de cada sensor (excluindo o atributo “timestamp”).

Após isso o grafana vai realizar uma request através do endpoint “`<str:user_token>/grafana/query`” que irá retornar vários pontos num certo período de tempo (definido no canto superior direito da dashboard), ilustrados na figura 4.11.

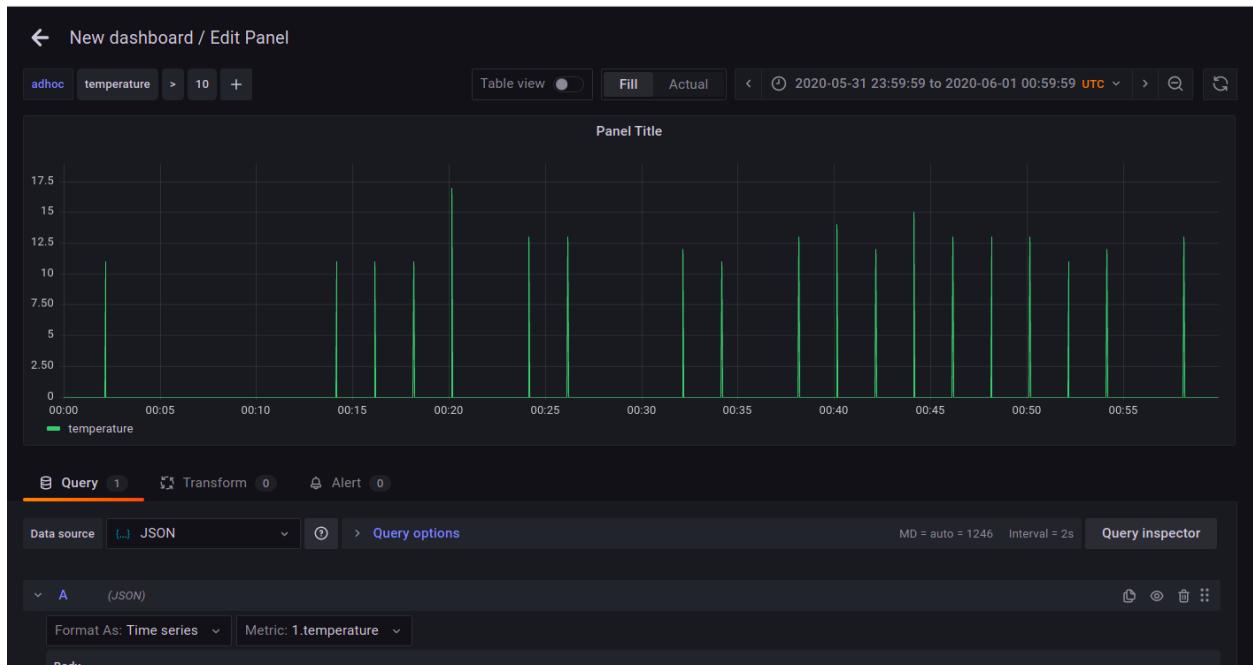


Figura 4.11 - Layout de *dashboard* do Grafana.

Também é possível adicionarmos ad hoc filters. Ao abrirmos as dashboard settings (no canto superior direito), clicarmos em “Variables” e adicionarmos uma nova variável com o “Type” sendo “Ad hoc filters”. Após isto, quando voltarmos à página de editar o gráfico, podemos ver mais opções de query no canto superior esquerdo, agora é possível, por exemplo, apresentar valores no gráfico com valores de “temperature” acima de 2.

É possível ver todos os passos de configuração e criação de uma dashboard no user manual fornecido no git.

5. Resultados e Análise da Solução

5.1 Testes

5.1.1 Planeamento

Primeiramente a quando do desenvolvimento do modelo de dados, antes deste ser integrado na nossa API foram efetuados testes em pequena escala de forma a entender se o modelo em si tinha uma performance adequada aos nossos objetivos quer a nível de querying quer de inserções. Os valores foram obtidos através do script de testes funcionais do modelo. E os cálculos dos crescimentos e construção de gráficos através de **Excel**. Durante todo o desenvolvimento do modelo de dados foram constantemente realizados testes funcionais de forma a verificar que as operações de inserção e query se comportavam como esperado.⁵

De forma a verificar a viabilidade do sistema foram previstos testes de performance e escalabilidade à base de dados, com intuito de verificar se num cenário de IoT com grande volume de registo e utilizadores de que forma variaria a performance da base de dados tanto a nível de inserts como querying. Ambos os testes foram efetuados através de scripts exteriores, facilmente customizáveis, de forma a explorar os diferentes fatores influentes, estes comunicam diretamente com os endpoints de forma a simular um cenário real com sensores e imprimem gráficos customizáveis através da biblioteca **matplotlib** de python.⁶

5.1.2 Resultados

5.1.2.1 Testes em pequena escala

Durante o desenvolvimento do modelo de dados foram efetuados testes em pequena escala, cujos resultados são mostrados na figura 5.1. De forma a tentar perceber a complexidade das diferentes operações que estavam a ser desenvolvidas, nomeadamente inserções e as 3 modalidades de querying existentes na altura, querying geral que serviu apenas como modelo para criar as futuras 4 modalidades, querying por utilizador e querying por sensor. Estes métodos apresentaram resultados muito satisfatórios uma vez que todas as operações pareciam ter complexidade inferior a O(n), no entanto estes métodos ainda sofreram alguma alteração no processo de adaptação à API que pode ter alterado a performance prevista.

⁵ Testes de Funcionais / Requisitos efetuados em pequena escala -
https://github.com/LuisValentim1/PI_DBOT_Lib/tree/main/DBoT

⁶ Testes Finais de Performance e Escalabilidade - <https://github.com/ritamf/PI/tree/main/TestesFinais>

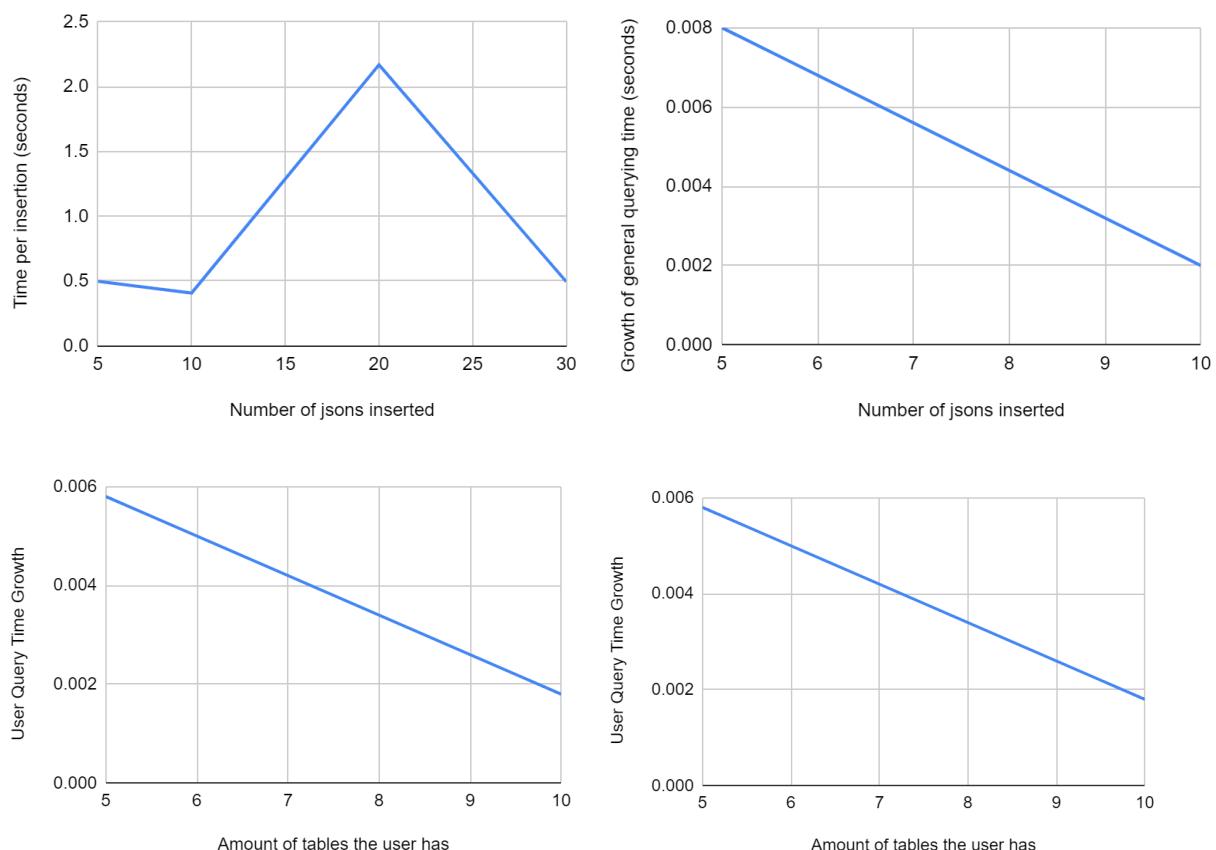


Figura 5.1 - Testes em pequena escala

O aumento inicial nas inserções estava visivelmente associado à criação das tabelas nas primeiras inserções.

5.1.2.2 Performance

Os testes de performance das queries apresentaram os resultados esperados, tendo em conta os valores que haviam sido apresentados previamente apresentados pelos testes de pequena escala, com crescimentos nulos a variância consideravelmente baixa. Esses resultados apresentam-se na figura 5.2.

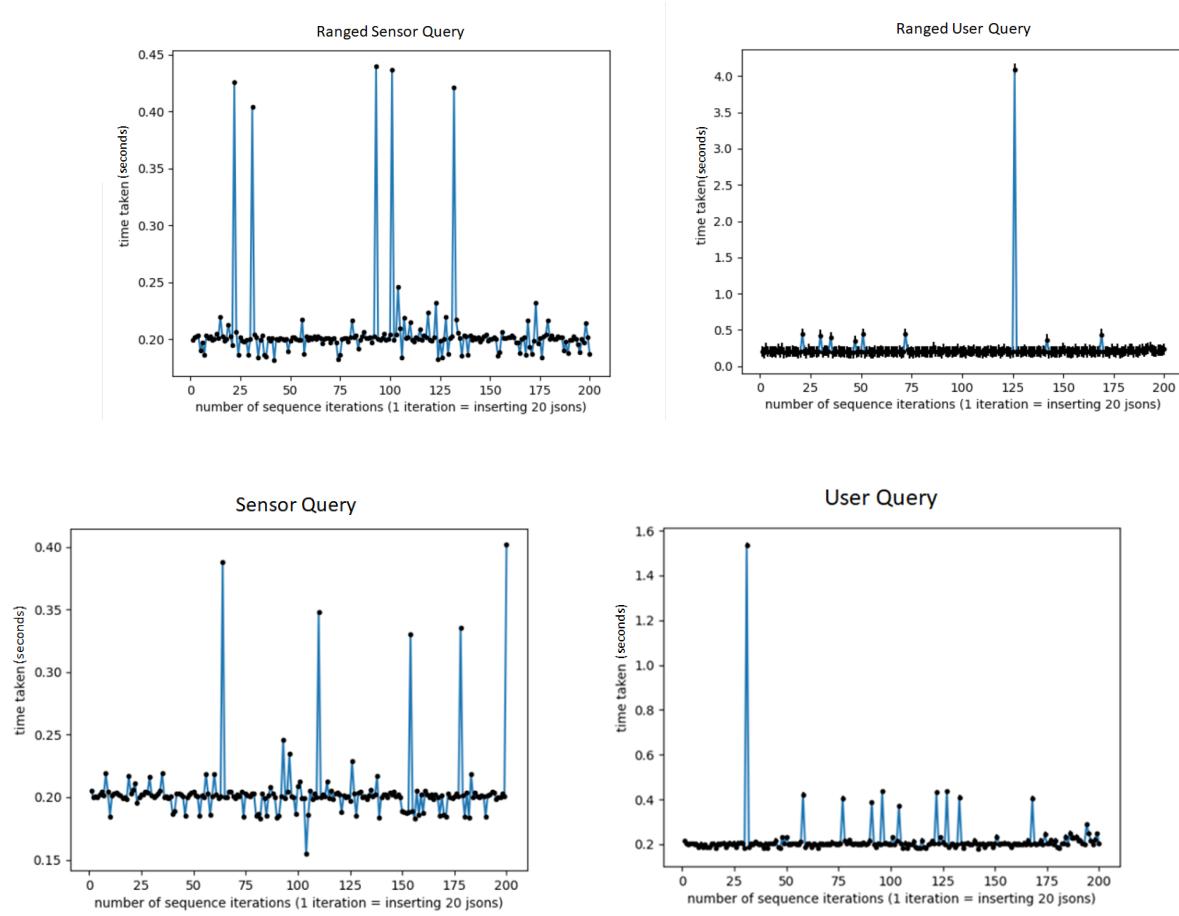


Figura 5.2 - Testes de performance

No entanto os resultados quanto à performance de inserção mostraram um problema crítico não previsto, existe um aumento no tempo de inserção que escala diretamente com o número de registos para um dado keyspace, este acréscimo está provavelmente relacionado com a criação de múltiplos keyspaces invés de ter apenas o keyspace público para todos os utilizadores que tinha sido considerado na solução inicial, as tabelas base são bastante extensas visto que registam informação em todos os registos e estão a ser replicadas em todos os keyspaces o que pode estar a causar demasiada pressão nos dois nós que possuímos, outros fatores que têm impacto neste acréscimo são o número de sessões de Cassandra ativas no Cache e a forma como os ficheiros são inseridos, introduzir dados em batches pequenos parece resultar num acréscimo bastante mais suave que batches de informação maiores, a biblioteca **cProfile** de python revelou que o processo que mais atrasa a inserção é a aquisição de thread locks, pelo que que estes registos forem inseridos com mínimo espaçamento temporal a performance irá melhor. Como é claro num cenário de IoT este problema compromete a viabilidade do produto, infelizmente não tivemos capacidade de o analisar e resolver no tempo restante quando o encontrámos mas visto que este está fortemente relacionado aos mecanismos de segurança é possível que não seja completamente mitigável para o produto que tínhamos em mente, mas acredito que seja viável na versão inicial com a base de dados comum a todos os utilizadores. O resultado pode ser confirmado na figura 5.3.

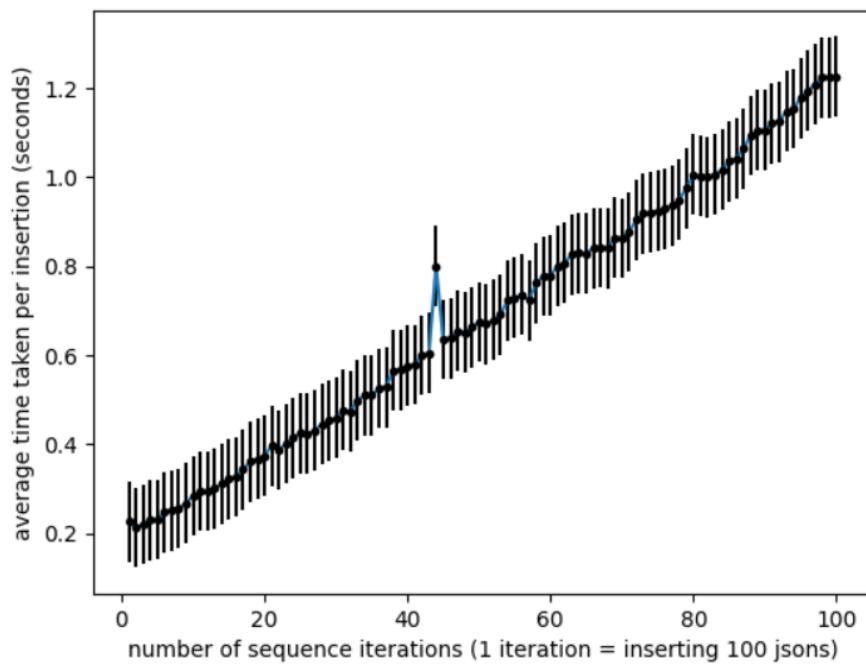


Figura 5.3 - Teste de performance

Ocasionalmente a performance estabilizou após já existir grande número de registo e se o número de inserções fosse baixo, no entanto estes resultados não foram consistentes ao longo de múltiplos testes pelo que não podemos confiar plenamente neles. O resultado pode ser verificado na figura 5.4.

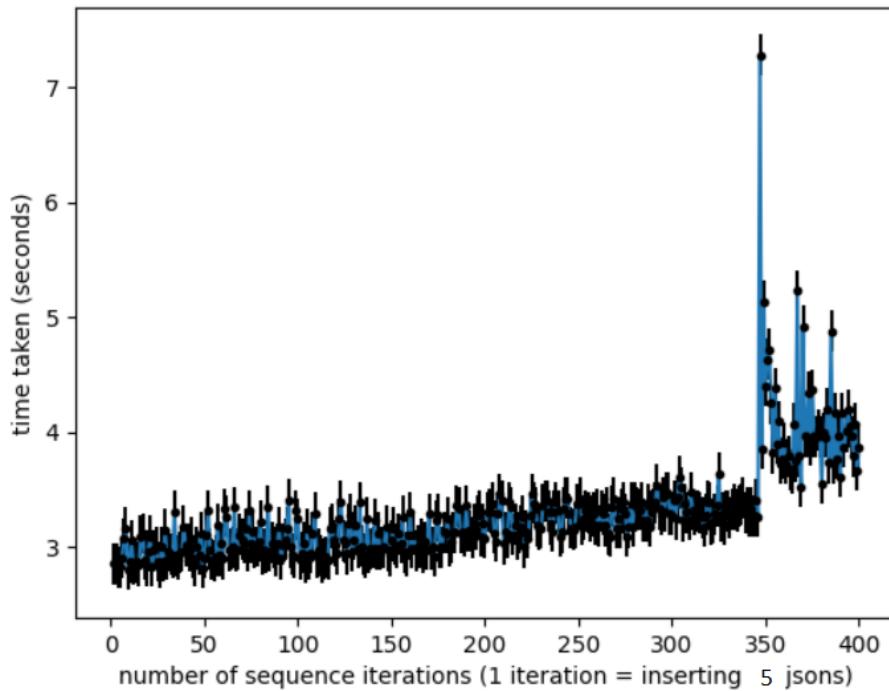


Figura 5.4 - Teste de performance

5.1.2.3 Escalabilidade

Os testes de escalabilidade apresentaram resultados aceitáveis, as operações têm uma perda de performance devido a existirem maior número de sessões ativas mas este valor escala com complexidade quase nula, sendo n o número de sensores concorrentes enquanto este se mantiver inferior à dimensão do Cache.

Para além disso, tendo em conta os problemas de performance previamente referidos, os resultados apresentados nos testes de escalabilidade parecem consideravelmente melhores, possivelmente por apresentarem menor número de inserções total, e as inserções são singulares. A imagem seguinte apresenta a performance com 3 threads a publicar em simultâneo.

Estes resultados relativos aos testes de escalabilidade podem ser vistos na figura 5.5.

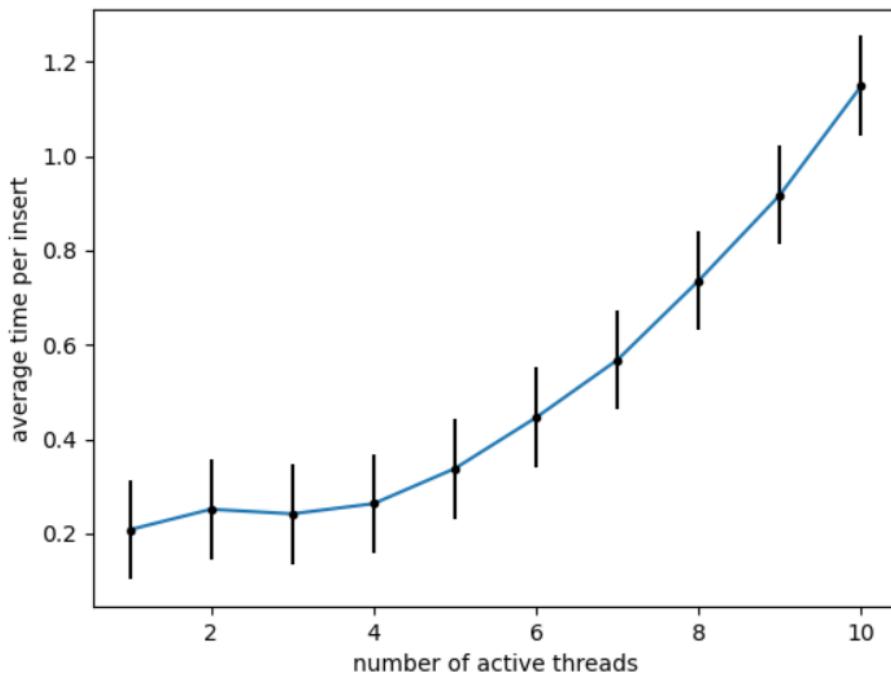
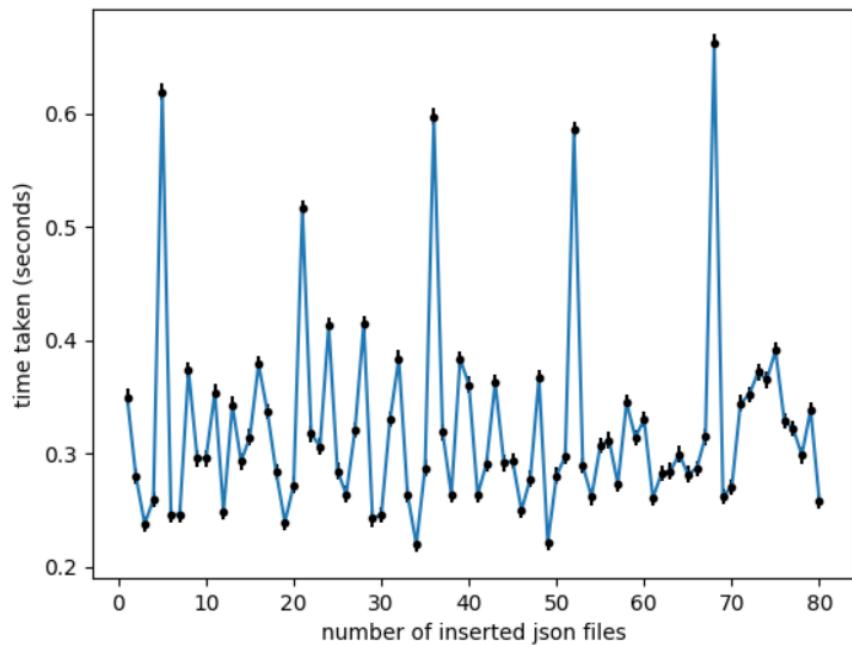


Figura 5.5 - Teste de escalabilidade.

Neste caso estava a ser utilizado um Cache de dimensão 4 como podemos observar até ao Cache ser preenchido a performance tem resultados muito consistentes, no entanto quando este valor é ultrapassado o tempo por insert começa a ser ineficiente.

5.2 Análise

Primeiramente é importante sublinhar que o problema de performance nas inserções é extremamente comprometedor para o conceito que tínhamos em mente e tendo em conta a área a que deveria ser aplicado é claramente impeditivo do funcionamento do produto em larga escala. Os restantes métodos estão todos a comportar-se como esperado e com valores bastante satisfatórios de performance e está pronto para lidar com múltiplos utilizadores a publicar dados em simultâneo.

Um dos principais fatores que influenciam a performance quer a lidar com múltiplas ligações, quer a lidar com big data é número de sessões ativas e o tamanho da Cache de sessões, com isto em mente este parâmetro deveria ser estudado com maior detalhe de forma a maximizar as capacidades do projeto.

Seria necessário mais tempo para comprovar três hipóteses que surgiram com a variação de condições dos testes, até que ponto esta solução é viável se os utilizadores se comprometerem a inserir valores diretamente invés de batches, se a solução seria viável com maior número de nodes cassandra e um tamanho ideal de Cache possivelmente definido de forma dinâmica, possivelmente tentar implementar mecanismos de segurança num sistema altamente experimental e com requisitos de performance e escalabilidade muito altos tenha sido demasiado otimista.

6. Conclusão e Trabalho Futuro

O nosso objetivo era construir uma base de dados sobre Cassandra, otimizada para cenários IoT capaz de lidar com grandes quantidades de dados e vários sensores a publicar em simultâneo e acessível online através de uma web app. Para além disso, foi proposto pelo grupo adicionar também mecanismos de segurança de forma a proteger a privacidade de dados dos utilizadores. O resultado obtido engloba tudo aquilo que tinha sido previsto com um grande calcnar de Aquiles devido aos problemas de performance das inserções, no entanto todos os requisitos funcionais e a esmagadora maioria dos não funcionais foram cumpridos.

Seria interessante explorar no futuro as duas soluções que foram desenvolvidas, ou seja tentar resolver os problemas de performance da base de dados segura com maior atenção à forma como pedidos de inserção são abordados e baixando ao máximo o tempo do processo de inserção quer através de melhorias em performance de código python quer na forma como a informação de suporte à inserção é acedida. Para além disso, explorar em maior detalhe a solução inicial sem preocupações de segurança e verificar se a performance se mantém consistente com os resultados dos testes de pequena escala e como melhorá-la para máxima eficiência e escalabilidade. Os testes efetuados revelaram uma dependência muito grande da performance do sistema com o tamanho da Cache e sessões ativas pelo que se a Cache conseguisse reconhecer dinamicamente quando está sobre maior pressão e alterar o seu tamanho de forma dinâmica poderíamos gerar resultados consideravelmente melhores, é claro que este é apenas um conceito experimental que não tivemos tempo de estudar em detalhe. A forma como lidamos com grandes volumes de dados deveria ser aprimorada e a capacidade de visualização de informação na forma tables pelo grafana independentemente do tipo de solução. Separar totalmente a interação com a base de dados do django.

7. Referências

- [1] "Documentação de Cassandra". [Online]. Available: <https://cassandra.apache.org/doc/latest/> . [Last Accessed: 30-Mai-2021]
- [2] "Documentação Cassandra". [Online]. Available: <https://docs.datastax.com/> . [Last Accessed: 02-Jun-2021]
- [3]"Documentação Cassandra" . [Online]. Available:
<https://www.tutorialspoint.com/cassandra/index.htm> . [Last Accessed: 03-Jun-2021].
- [4] “Elastic Search Pesquisa / Data Model”. [Online]. Available:
<https://www.elastic.co/guide/index.html> . [Last Accessed: 09-Abr-2021]
- [5]“VergeDB”. [Online]. Available: http://cidrdb.org/cidr2021/papers/cidr2021_paper11.pdf . [Last Accessed: 09-Abr-2021]
- [6]“Scalable semantic aware context storage”. [Online].
Available:<https://www.sciencedirect.com/science/article/abs/pii/S0167739X15002885> . [Last Accessed: 11-Abr-2021]
- [7] “Elassandra Pesquisa / Data Model”. [Online]. Available:
http://doc.elassandra.io/en/latest/?_ga=2.92552797.1711040407.1624558753-1639721739.1624558753 . [Last Accessed: 11-Abr-2021]
- [8] John Paparrizos , Chunwei Liu , Bruno Barbaroli , Johnny Hwang , Ikraduya Edian , Aaron J. Elmore , Michael J. Franklin , Sanjay Krishnan “VergeDB: A Database for IoT Analytics on Edge Devices” - VergeDB pesquisa .
- [9] “Documentação Django”. [Online]. Available: <https://www.djangoproject.org/> . [Last Accessed: 22-Mai-2021]
- [10] “Estrutura API Django” . [Online].
Available:<https://github.com/catrinaacsilva/data-retention-light/tree/main/dataRetention> . [Last Accessed: 02-Mai-2021]
- [11] “Estrutura de API Django”.[Online]. Available:
<https://github.com/catrinaacsilva/incidents-webapp/tree/master/semantic> . [Last Accessed: 02-Mai-2021]
- [12] “Estrutura de biblioteca externa”. [Online]. Available: <https://github.com/mariolpantunes/uts>. [Last Accessed: 04-Mai-2021]
- [13] “Cassandra Authentication” . [Online]. Available:
<https://docs.apigee.com/private-cloud/v4.18.05/enable-cassandra-authentication> . [Last Accessed: 09-Jun-2021]
- [14] “Cassandra Authentication” . [Online].
Available:<http://michal.sznurawa.pl/docker-cassandrarolemanager-doesnt-support-password/>. [Last Accessed: 09-Jun-2021]
- [15] “Cassandra Authentication”. [Online] . Available:
<https://www.guru99.com/cassandra-security.html> . [Last Accessed: 26-Mai-2021]
- [16] “Documentação Grafana JSON simpod datasource”. [Online] Available:
<https://grafana.com/grafana/plugins/simpod-json-datasource/>. [Last Accessed: 17-Jun-2021]

- [17] “JSON Datasource usage tutorial”. [Online]. Available :
<https://oznetnerd.com/2018/04/17/writing-a-grafana-backend-using-the-simple-json-datasource-task/> . [Last Accessed: 01-Jun-2021]
- [18] “Cassandra Deployment”. [Online]. Available:
<https://github.com/thelastpickle/docker-cassandra-bootstrap/>. [Last Accessed: 07-Jun-2021]
- [19] “Cassandra Deployment”. [Online]. Available:
https://docs.axway.com/bundle/APIGateway_762_CassandraGuide_allOS_en_PDF/raw/resource/enus/APIGateway_CassandraGuide_allOS_en.pdf . [Last Accessed: 07-Jun-2021]
- [20] “Cassandra Deployment”. [Online]. Available:
<https://digitalis.io/blog/apache-cassandra/containerized-cassandra-cluster-for-local-testing/> .
[Last Accessed: 05-Jun-2021]
- [21] “API Deployment” . [Online]. Available:
https://www.youtube.com/watch?v=nh1ynJGJuT8&ab_channel=LondonAppDeveloper . [Last Accessed: 05-Jun-2021]
- [22] “Python Django Web Framework tutorial” . [Online]. Available:
<https://www.youtube.com/watch?v=F5mRW0jo-U4> . [Last Accessed: 03-Jun-2021]
- [23] “API Deployment”. [Online]. Available:
<https://medium.com/@saijalshakya/5-steps-to-deploy-django-application-using-nginx-reverse-proxy-unicorn-and-ssl-certificate-52990194b4c5> . [Last Accessed: 08-Jun-2021]
- [24] “API de visualização de gráficos, Testes”. [Online]. Available:
<https://matplotlib.org/stable/api/> . [Last Accessed: 12-Jun-2021]

Apêndices

Configurações

Cassandra

No intuito de assegurar a privacidade de dados é utilizado o sistema de autenticação que requer algumas mudanças às configurações de cassandra nomeadamente:

- authenticator: PasswordAuthenticator
- roles_validity_in_ms: 0
- roles_update_interval_in_ms: 0
- credentials_validity_in_ms: 0
- credentials_update_interval_in_ms: 0
- authorizer: CassandraAuthorizer

Django

As configurações no django foram feitas dentro do ficheiro “settings.py”.

ALLOWED_HOSTS = ['10.0.12.65','10.0.12.66'] indica os endereços que o django pode servir.

A base de dados utilizada para a criação de tabelas relacionadas com a autenticação tem as seguintes configurações:

- ‘NAME’: ‘db’
- ‘USER’: ‘cassandra’
- ‘PASSWORD’: ‘cassandra’
- ‘HOST’: ‘10.0.12.65’
- ‘strategy_class’: ‘SimpleStrategy’
- ‘replication_factor’: 1
- ‘consistency’: ConsistencyLevel.ONE
- ‘port’: 9042

Também foi necessário configurar a REST framework, para isso foi adicionado ‘rest_framework’ a ‘INSTALLED_APPS’ e depois em ‘DEFAULT_PARSER_CLASSES’ foram indicados os parsers a poderem ser utilizados quando o request.data é acedido. Estes parsers são:

- rest_framework.parsers.JSONParser
- rest_framework.parsers.FormParser

- `rest_framework.parsers.MultiPartParser`