Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Unidade de Engenharia Elétrica

Relatório de Estágio Supervisionado

Marcelo Meneses da Silva

Campina grande, Paraíba

Novembro de 2019

MARCELO MENESES DA SILVA

Relatório de Estágio Supervisionado submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

.

Marcelo Meneses da Silva

Orientando

Professor Rafael Bezerra Correia Lima, D. Sc.

Orientador

Campina Grande, Paraíba

Novembo de 2019

RESUMO

Este relatório apresenta as atividades realizadas pelo aluno Marcelo Meneses da Silva durante o Estágio Supervisionado no Laboratório de Instrumentação Eletrônica e Controle (LIEC), pertencente ao Departamento de Engenharia Elétrica (DEE) da Universidade Federal de Campina Grande (UFCG), sob orientação do Professor Rafael Bezerra Correia Lima e supervisão do Professor George Acioli Júnior.

Palavras-chave: MQQT, Microsoft Azure, Cosmos DB, Xamarin

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

MQTT	Message Queuing Telemetry Transport
FTP	File Transfer Protocol (Protocolo de Transferência de Arquivos)
TCP	Transmission Control Protocol (Protocolo de Controle de Transmissão)
IP	Internet Protocol (Protocolo de Internet).
UFCG	Universidade Federal de Campina Grande
DEE	Departamento de Engenharia Elétrica
LIEC	Laboratório de Instrumentação Eletrônica e Controle
SQL	Structured Query Language (Linguagem de Consulta Estruturada)
NoSQL	Not Structured Query Language (Linguagem de Consulta Não-Estruturada)
DB	Data Base (Banco de Dados)
IDE	Integrated Development Environment (Ambiente de desenvolvimento integrado)
JSON	JavaScript Object Notation (Notação de Objetos JavaScript)

Sumário

1 Introdução

A disciplina Estágio Supervisionado é componente obrigatório da grade curricular do Curso de Engenharia Elétrica da Universidade Federal de Campina Grande e tem como principal objetivo possibilitar que o aluno em final de curso utilize os conhecimentos adquiridos nas disciplinas ao longo da sua formação para desempenhar atividades que o ajudem a se preparar para o que será encontrado no âmbito profissional. Este relatório contém a descrição das tarefas realizadas entre o período de 26 de agosto a 25 de outubro de 2019 com carga horária de 21 horas semanais, totalizando 209 horas sob a orientação do professor Rafael Bezerra Correia Lima e supervisão do professor George Acioli Júnior.

1.1 Local do estágio

Localizado na Universidade Federal de Campina Grande (UFCG) no campus de Campina Grande, o Laboratório de Instrumentação Eletrônica e Controle (LIEC) é um laboratório pertence ao Departamento de Engenharia Elétrica (DEE). Integrado por professores doutores, alunos de pós-graduação e de graduação, esse laboratório tem como principal objetivo desenvolver atividades e projetos ligados às áreas de automação, controle e instrumentação.

Figura 1 - Fachada do LIEC

Fonte:http://www.cct.ufcg.edu.br/wp-content/uploads/2017/11/liec.jpg acessado em 5/11/2019

1.2 Plano de estágio

A carga horária matriculada do Estágio Supervisionado foi de 180 horas cujo plano inicial estabelecido é descrito abaixo.

1. Revisão bibliográfica de protocolos de comunicação para aplicações Iot;
2. Familiarização com o protocolo MQTT;
3. Treinamento na plataforma Azure e Xamarin;
4. Implementação de um sistema de retaguarda para dispositivos embarcados em rede;
5. Execução de testes de validação;
6. Elaboração da documentação escrita dos desenvolvimentos.

2 Fundamentação Teórica

A seguir serão apresentados alguns conceitos relacionados com as atividades desenvolvidas no estágio.

2.1 MQTT

MQTT, sigla de Message Queuing Telemetry Transport, é um protocolo bastante utilizado por sensores e atuadores em Internet das Coisas. É leve, aberto, simples e projetado para ser fácil de implementar. Estas características o tornam ideal para ser usado em muitas situações, incluindo ambientes restritos como por exemplo nos contextos de comunicação Máquina para Máquina (Machine to Machine(M2M)) e IoT onde códigos que ocupam pouca memória são necessários e/ou a largura de banda da rede é limitada. [2]

O protocolo funciona acima da pilha de protocolos TCP/IP, ou acima de outros protocolos de rede que fornecem conexões sem perdas de pacotes e bidirecionais. Dentre as características do protocolo MQTT estão:

- Uso do padrão de mensagens publish/subscribe que fornece distribuição de mensagens de um-para-muitos e desacoplamento entre as aplicações.
- Um transporte de mensagens que é agnóstico em relação ao conteúdo do payload, ou seja dados de qualquer tipo podem ser transmitidos.
- Três níveis de qualidade de serviço.
- Um pequeno overhead de transporte e trocas de mensagens minimizadas para reduzir o tráfego de dados na rede.
- Um mecanismo para notificar os dispositivos interessados quando desconexões anormais ocorrerem.

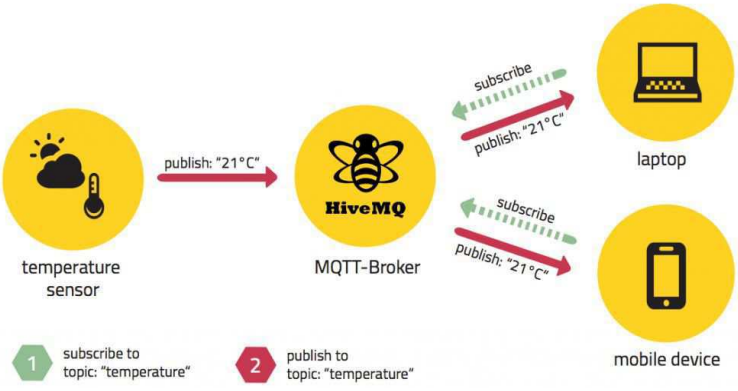
2.1.1 Padrão *Publish/Subscribe*

O protocolo MQTT é baseado no princípio de publicação de mensagens e assinatura de estruturas denominadas tópicos, o que é tipicamente referido como modelo publish/subscribe. Neste modelo, clientes podem assinar tópicos que sejam pertinentes a eles e portanto receber quaisquer mensagens que sejam publicadas nestes tópicos. Alternativamente, clientes também podem publicar mensagens em tópicos, deixando-as disponíveis para todos os assinantes destes tópicos. [1]

O padrão publish/subscribe (pub/sub) é uma alternativa ao tradicional modelo cliente-servidor, onde um cliente se comunica diretamente com um servidor. Neste padrão o cliente, que está enviando uma determinada mensagem (chamado publicador) é desacoplado de outro cliente (ou mais clientes), que estão recebendo a mensagem (chamado assinante). Isto significa que o publicador é assinante não sabem a respeito um do outro. Existe um terceiro componente, chamado broker, que é conhecido por ambos o publicador e o assinante, que é responsável por filtrar todas as mensagens que chegam até ele e as distribui adequadamente. [1]

Na figura abaixo, é apresentado um exemplo do uso do padrão pub/sub para a troca de mensagens entre um dispositivo medidor de temperatura e outros clientes interessados em receber estes valores de temperatura.

Figura 2 - Modelo *Publish / Subscribe*



Fonte:

https://www.mogozobo.com/wp-content/uploads/2017/08/pub-sub-mqtt-1024x588.png acessado 5/11/2019

2.1.2 Estrutura de Tópicos

Mensagens em MQTT são publicadas em tópicos, que podem ser pensadas como áreas de interesse. Clientes, por sua vez, se inscrevem para receber mensagens particulares por meio da assinatura de um tópico. Assinaturas podem ser explícitas, o que limita as mensagens que estão sendo recebidas a um tópico em específico ou podem usar designadores wildcards, como por exemplo o sinal (#) para receber mensagens de uma variedade de tópicos. [4]

Em outras palavras, um tópico é uma string que é utilizada pelo broker para filtrar as mensagens para cada cliente conectado. Cada nível de um tópico é separado por uma barra. A seguir alguns exemplos de tópicos:

- LIEC/Sala_102/temperatura
- LIEC/Sala_93/maquina_3/rotacao

2.1.3 Níveis de Qualidade de Serviço (QoS)

O nível de qualidade de serviço - Quality of Service (QoS) - é um acordo entre o remetente e o receptor da mensagem no que concerne a garantia de entrega da mensagem. No protocolo em questão, existem 3 níveis de QoS.

2.1.4.1 QoS 0 - at most once

Este é o menor nível de qualidade de serviço e o mais rápido. Ao usar este nível de qualidade de serviço o cliente/broker tentará entregar a mensagem uma vez, porém sem nenhuma confirmação de entrega. Assim, não há garantias que o pacote chegue ao broker. [3]

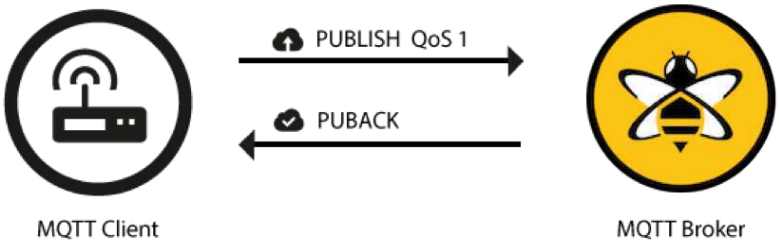
Figura 3 - Mensagens trocadas entre Cliente e Broker com QoS 0

Fonte: https://www.hivemq.com/img/blog/QoS-0.png acessado em 5/11/2019

2.1.4.2 QoS 1 - at least once

Ao usar o nível de qualidade de serviço 1, existe a garantia de que a mensagem chegará ao seu destino pelo menos uma vez. O cliente enviará o pacote até que o Broker envie um pacote de recebimento. Desta forma, é possível que a mensagem seja entregue mais de uma vez ao destinatário. [3]

Figura 4 - Mensagens trocadas entre Cliente e Broker com QoS 1



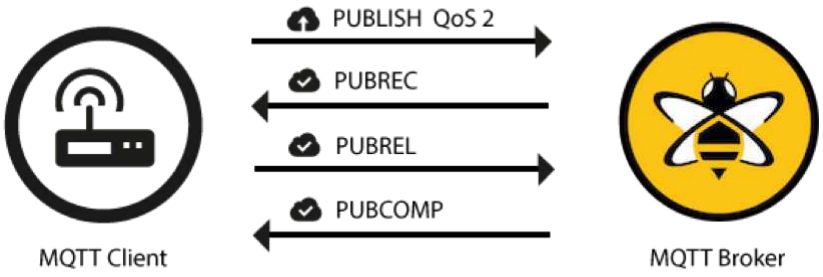
Fonte: https://www.hivemq.com/img/blog/QoS-1.png acessado em 5/11/2019

2.1.4.3 QoS 2 - exactly once

Este é o mais alto nível de QoS e também o mais lento. Ao usar o nível 2 de qualidade de serviço é garantido que a mensagem será entregue apenas uma vez no destinatário.

Em suma, o nível de QoS define o quanto o cliente/broker irá se esforçar para garantir que a mensagem seja entregue com sucesso ao destinatário. A escolha do nível de QoS a ser utilizado depende bastante da aplicação em questão e dos recursos de rede disponíveis. Por exemplo, se o dado transmitido não é tão importante ou se é enviando em pequenos intervalos de tempo e para aplicação em questão é aceitável que algumas mensagens não cheguem ao destinatário,o nível de QoS 0 é suficiente. Por outro lado, se é crítico para a aplicação receber todas as mensagens e apenas uma vez deve-se optar pelo nível QoS 2. [3]

Figura 5 - Mensagens trocadas entre Cliente e Broker com QoS 2



Fonte: https://www.hivemq.com/img/blog/QoS-2.png acessado em 5/11/2019

2.2 Azure

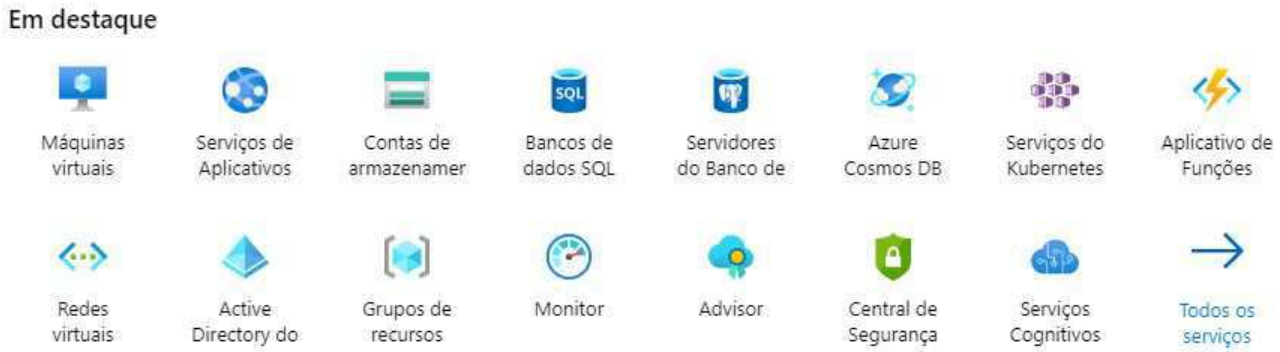
O Azure é um conjunto cada vez maior de serviços de computação em nuvem para ajudar sua organização a enfrentar seus desafios de negócios. O Azure oferece a liberdade de criar, gerenciar e implantar aplicativos em uma rede global massiva usando suas ferramentas e estruturas preferidas. [9]

Entre as vantagens e funcionalidades da plataforma Azure destacadas em seu site pode-se destacar:

- **Menor Custo** - A computação em nuvem acaba com a despesa de configurar e executar datacenters no site, que geralmente têm custos agregados como empregar a equipe e comprar e manter o terreno, os edifícios e o hardware do computador. A nuvem permite que as empresas acessem os recursos do computador de que elas precisam em tempo real para atender às necessidades delas sob demanda. [9]
- **Maior Segurança** - A segurança é um foco importante dos provedores de nuvem, que investem grandes montantes de dinheiro para proteger sua infraestrutura. Os provedores de nuvem também oferecem um amplo conjunto de políticas, conformidade, tecnologias e controles que fortalecem sua postura de segurança, protegendo os dados, os aplicativos e a infraestrutura contra ameaças. [9]
- **Maior Produtividade -** Desenvolva e gerencie seus aplicativos com mais eficiência com recursos de computação em nuvem quase ilimitados. Os provedores de nuvem atualizam continuamente as redes de datacenter com o hardware de última geração, o que proporciona recursos de computação rápidos e eficientes que nunca ficam obsoletos e que seria mais caro de serem implementados em um único datacenter. [9]
- **Escala Global -** A computação em nuvem é executada em data centers em todo o mundo, oferecendo resiliência e confiabilidade geral ao permitir que o backup dos seus dados seja feito em mais de uma localização geográfica. Isso permite que seus recursos de TI sejam distribuídos de localizações geográficas específicas quando necessário. [9]

Na parte inicial da plataforma podemos ver as funcionalidades principais presentes na Azure. Como pode-se ver na figura 6, algumas dessas funções são o Aplicativo de Funções e Cosmos DB que serão melhor abordadas nos próximos capítulos. [9]

Figura 6 - Painel principal das funções em destaque do portal Azure



Fonte: Adaptado de Microsoft Azure Education (2019)

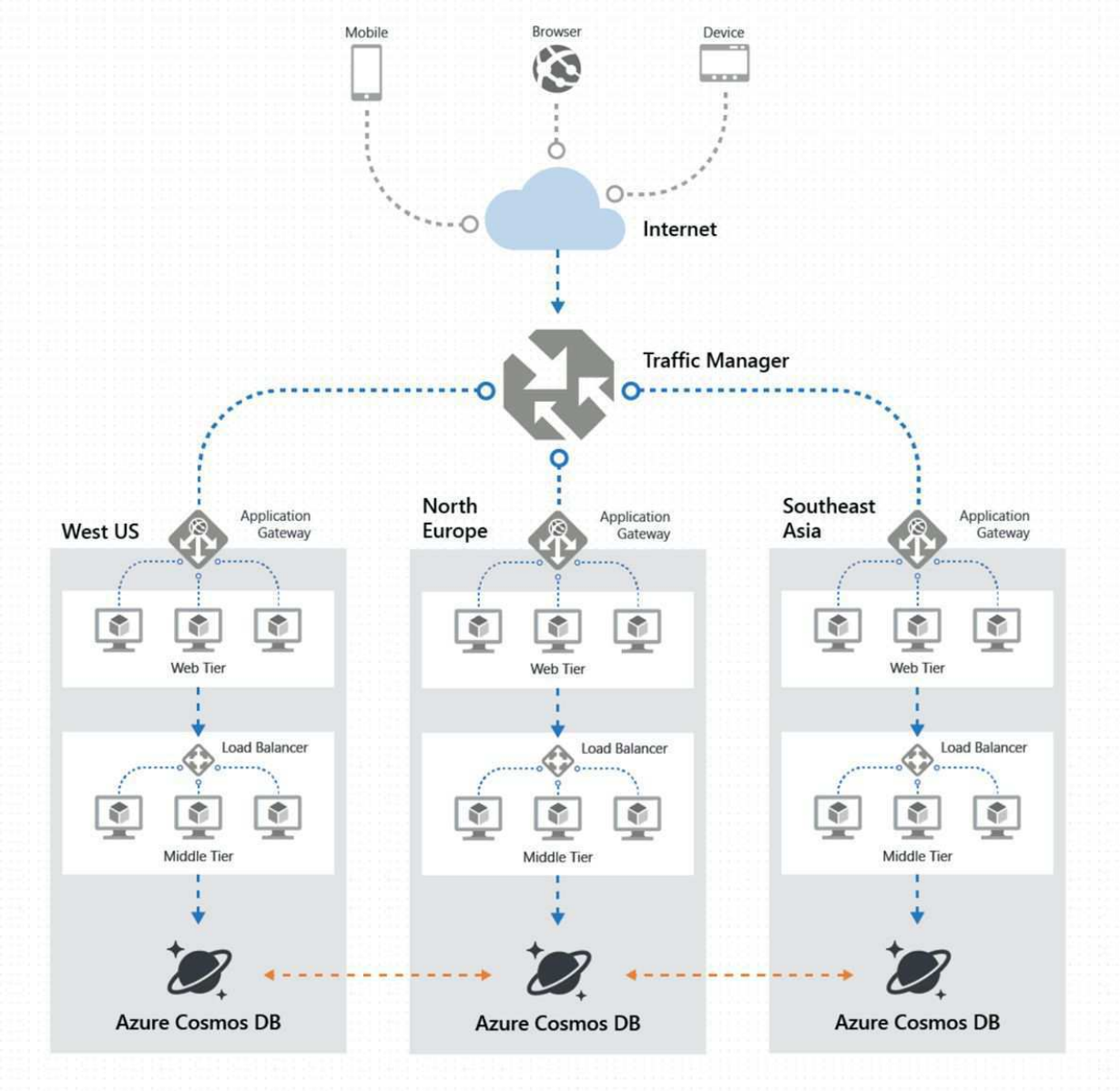
2.2.1 Cosmos DB

As aplicações de hoje precisam ser altamente responsivas e sempre on-line. Para obter baixa latência e alta disponibilidade, instâncias desses aplicativos precisam ser implantadas em *datacenters* próximos aos seus usuários. Esses aplicativos geralmente são implantados em vários datacenters. [8]

Segundo a *Microsoft*, o Azure Cosmos DB é um serviço de banco de dados distribuído globalmente projetado para fornecer baixa latência, escalabilidade elástica de taxa de transferência, semântica bem definida para consistência de dados e alta disponibilidade.

É possível configurar os bancos de dados para serem distribuídos globalmente e disponíveis em qualquer uma das regiões do Azure. Para diminuir a latência, coloque os dados próximos de onde os usuários estão. Cosmos DB replica os dados de forma transparente para todas as regiões associadas à conta do cosmos do usuário. Ele fornece uma única imagem do sistema do banco de dados do Azure Cosmos globalmente distribuído e dos contêineres que o aplicativo pode ler e gravar localmente. [8]

Figura 7 - Comportamento de um banco de dados Cosmos DB em multiplos continentes



Fonte: https://docs.microsoft.com/pt-br/azure/cosmos-db/distribute-data-globally acessado 5/11/2019

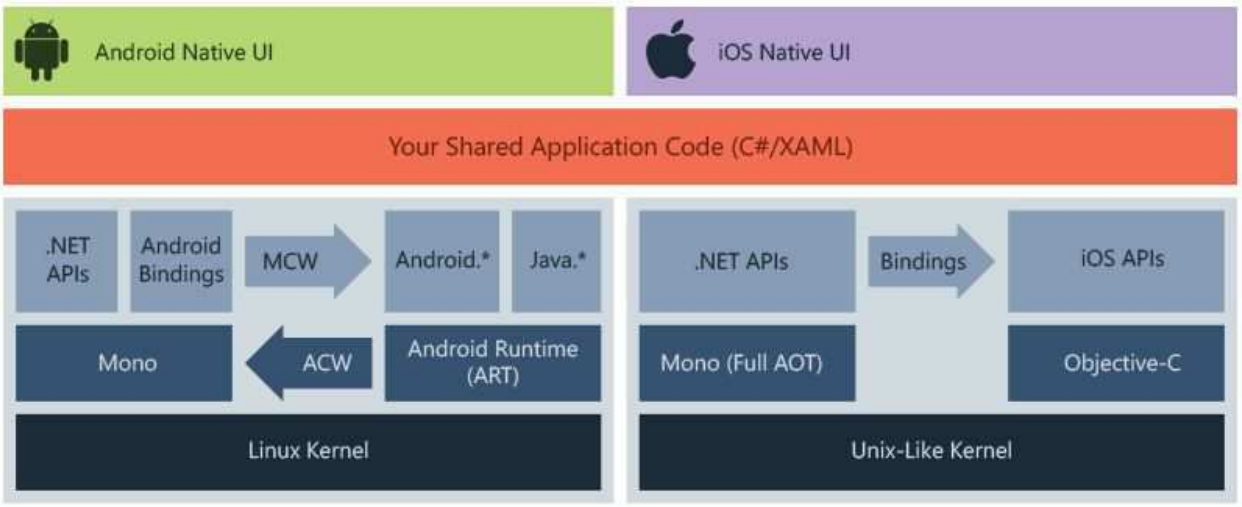
2.2.2 Xamarin

O Xamarin é uma plataforma de desenvolvimento mantida pela Microsoft que permite a criação de aplicativos móveis nativos para iOS, Android e Windows Phone utilizando como base a linguagem C# com o *framework* .NET. Além de ter acesso as APIs nativas de cada plataforma é possível criar interfaces do usuário nativas e compiladas para cada uma delas, o que melhora o tempo de execução da aplicação, principalmente se comparado à aplicativos híbridos. [7]

A criação de aplicativos multiplataformas sem a necessidade de conhecer as linguagens específicas de cada uma, como Java para Android e Objective-C ou Swift para iOS, por exemplo, ganha produtividade com a utilização do Visual Studio que permite ainda a depuração remota de cada dispositivo. [7]

Na figura 8 pode-se ver o funcionamento da arquitetura multiplataforma que gera aplicativos Android e iOS nativos.

Figura 8 - Arquitetura multiplataforma de desenvolvimento único Xamarin



Fonte: https://docs.microsoft.com/pt-br/xamarin/get-started/what-is-xamarin acessado em 5/11/2019

3 Atividades desenvolvidas

Neste capítulo serão apresentados as principais atividades realizadas durante o período de duração do estágio supervisionado.

3.1 Arquitetura do Sistema

O sistema de retaguarda abordado neste relatório se propõe a conectar a parte eletrônica e de instrumentação presente em salas do LIEC com um sistema de interação com o usuário. A eletrônica preparada comunica-se por meio do protocolo de comunicação MQTT, por isso, é necessário a familiarização com este protocolo.

Os dados enviados pelos sensores serão salvos em um banco de dados e seu acesso deve ser restrito a um usuário autorizado. Para isso é necessário a implementação de um banco de dados cosmos db e um sistema web de retaguarda.

3.2 Familiarização com o protocolo MQTT

Para o pleno desenvolvimentos das atividades do estágio foi necessário se familiarizar com o protocolo de comunicação MQTT, introduzido na fundamentação teórica. Essa proximidade se deu por meio de cursos online e texto base de apoio. Estudou-se as caracteristica principais do protocolo e feito testes utilizando servidores públicos e *softwares* gratuitos como o MQTTBox.

O conhecimento adquirido nesta etapa foi fundamental para o desenvolvimento de códigos que enviam mensagens MQTT que serão abordados nos tópicos posteriores.

3.3 Treinamento na plataforma Azure e Xamarin

Após a familiarização com o protocolo MQTT se iniciou um treinamento de capacitação e familiarização com as plataformas de desenvolvimento integrado da microsoft. Instalou-se a IDE Visual Studio e feito cursos online de linguagem de programação C#. Esse aprendizado de

uma nova linguagem, manuseio de softwares e plataformas nunca utilizadas pelo estagiário requerem bastante tempo.

3.3.1 Xamarin

A primeira etapa do treinamento na plataforma de desenvolvimento integrado Xamarin forms se deu por meio de um curso online da *Udemy*, visto na figura 9, e séries de vídeos de desenvolvedores microsoft.

Figura 9 - Descrição do curso online de treinamento em Xamarin Forms

Fonte: Adaptado de https://www.udemy.com/course/xamarin-forms-course/ acessado em 5/11/2019

Após a conclusão o curso, forneceu-se o código fonte de um aplicativo Xamarin em desenvolvimento cujo objetivo era atuar e receber dados por meio do protocolo MQTT. Alterou-se o código fonte desse aplicativo, simplificado e padronizado.

No código principal foi criado uma única função para lidar com o protocolo MQTT que pode ser vista abaixo:

```
1.      private async void SendMqttMessageLED(string mensageMqtt) {
2.          var message1 = new MqttApplicationMessage(LEDPath,
    Encoding.UTF8.GetBytes(mensageMqtt));
3.          await client.PublishAsync(message1, MqttQualityOfService.AtMostOnce);
4.      }
```

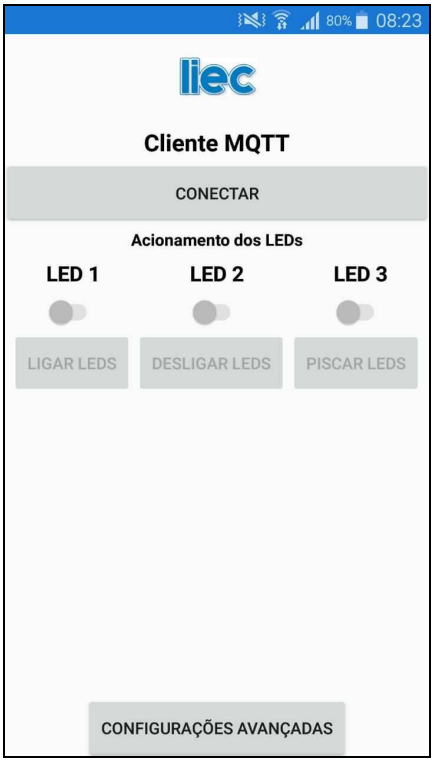
Então, qualquer outra função se delega apenas a passar qual será a mensagem a ser enviada, podendo, assim, separar melhor a parte gráfica da parte de processamento. O código abaixo mostra uma função que é chamada quando o usuário clica no botão nomeado “ Button_Clicked_7Async”, comparando se o cliente está conectado ao broker e passando como parâmetro a string “on” para a função codificada acima. Além da parte de processamento, a

função possui as linhas 7 e 8 destinadas a manipulação da interface gráfica alterando a disponibilidade ao clique de dois botões.

```
1.      private void Button_Clicked_7Async(object sender, EventArgs e){
2.          if (client.IsConnected == true){
3.              SendMqttMessageLED("on");
4.              Botao8.IsEnabled = false;
5.              Botao9.IsEnabled = true;
6.          }
7.      }
```

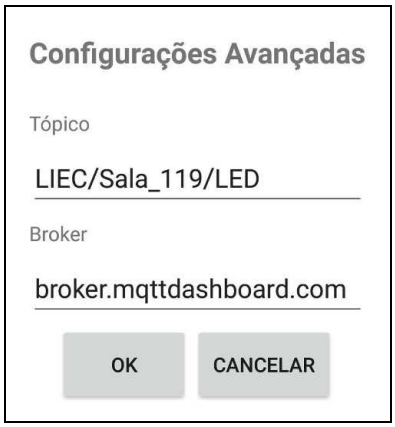
Modificou-se a interface com o usuário também, incluindo a logo do LIEC, diminuição das chaves, como vistos na figura 10, e padronização dos nomes das variáveis e objetos gráficos. Implementou-se um botão de configurações avançadas para adicionar a funcionalidade de alterar o broker (servidor) e o tópico o qual as mensagens MQTT deveriam ser enviadas, como vistos na figura 11.

Figura 10 - Tela inicial do aplicativo modificado



Fonte: O próprio autor (2019)

Figura 11 - Tela de configurações avançadas



Fonte: O próprio autor (2019)

3.3.2 Azure

Como o estagiário nunca havia interagido com a plataforma da Azure foi necessário um treinamento. A microsoft disponibiliza em seu site alguns cursos gratuitos para a familiarização com seus produtos. Como introdução realizou-se dois cursos, um sobre os conceitos básicos da plataforma -vide figura 12 - e outro sobre a implementação de um site nos servidores da Azure como visto na figura 13.

Criou-se uma conta gratuita de estudante que disponibiliza 100 dólares de créditos para testes de serviços azure. Os valores gastos estão detalhados no ANEXO II. Tentou-se aproveitar o máximo de funcionalidades dentro da oferta gratuita disponibilizada. O preço de cada componente está disponível tanto em tabelas descritivas quanto em um site para simulação de precificação. Assim, o desenvolvedor pode testar quanto será o seu gasto mensal com uma aplicação antes de implementá-la, podendo se preparar melhor ou escolher um plano mais adequado às suas disponibilidades financeiras e requisitos do sistema desenvolvido.

Figura 12 - Curso gratuito de conceitos básicos do Azure

Fonte: Adaptado de https://docs.microsoft.com/pt-br/learn/paths/azure-fundamentals/ acessado em 5/11/2019

Figura 13 - Curso gratuito de Implementação de um site no Azure

Fonte: Adaptado de https://docs.microsoft.com/pt-br/learn/paths/deploy-a-website-with-azure-app-service/ acessado em 5/11/2019

3.3.3 Cosmos DB

Como pode-se ver no ANEXO II, a implementação de um banco de dados Azure com núcleo SQL é muito cara. Portanto, para reduzir os custos de manutenção do sistema optou-se pelo uso gratuito do Cosmos DB. Este permite até 100 requisições por segundos gratuitas e a criação de uma única *conteiner* (similar a uma "tabela" SQL mas com dados estruturados em JSON).

A microsoft disponibiliza um curso gratuito que fornece uma introdução ao desenvolvimento e implementação do Cosmos DB, como pode ser visto na figura 14. Porém, além desse curso foi necessário uma profunda análise da documentação disponibilizada pela plataforma, seguindo e testando os passos para a correta implementação do banco de dados.

Passada a fase de treinamento inicial ainda persistiram algumas dúvidas e problemas quanta correta implementação do acesso ao banco de dados Cosmos DB por um aplicativo web hospedados dentro dos servidores da Azure, essas dificuldades atrasaram bastante o cronograma já apertado do estágio.

Figura 14 - Curso gratuito sobre Cosmos DB com dados NoSQL

Fonte: Adaptado de https://docs.microsoft.com/pt-br/learn/paths/work-with-nosql-data-in-azure-cosmos-db/ acessado em 5/11/2019

3.4 Implementação de um sistema de retaguarda para dispositivos embarcados em rede

Após o treinamento nas plataformas de desenvolvimento integrado da microsoft e da melhoria do aplicativo desenvolvido em Xamarin Forms, deu-se início ao desenvolvimento do sistema de retaguarda para dispositivos embarcados em rede utilizando o protocolo MQTT.

3.4.1 Requisitos

Para o correto entendimento do sistema foram levantados alguns requisitos para o funcionamento mínimo do sistema. Tais requisitos se encontram evidenciados na Tabela 1.

Tabela 1 - Lista de Requisitos

R001	O usuário deve ser capaz de logar no sistema.
R002	O sistema deve ser capaz de enviar mensagens no protocolo MQTT, de maneira padronizada, para um tópico pré-estabelecido.
R003	O sistema web deve ser capaz de se comunicar com o banco de dados Cosmos DB
R004	O usuário deve ser capaz de atuar nos atuadores disponíveis da sala que ele tem acesso permitido.
R005	Deve-se utilizar um *template* bootstrap para agilizar o desenvolvimento do sistema.

O padrão informado da estrutura de tópicos das mensagens MQTT é a seguinte:

- LIEC/Sala_119/Temperature
- LIEC/Sala_119/Light
- LIEC/Sala_119/Presence
- LIEC/Sala_119/Curtain
- LIEC/Sala_119/Air_Conditioner

Onde “Sala_119” indica a sala que o usuário tem acesso seguido pelo nome do atuador ou sensor presentes na sala.

Para cada tópico há um padrão definido de troca de mensagens, descrito abaixo:

1. Cortina
    - UP
    - DOWN
    - STOP

Onde “UP” e “DOWN” controlam o sentido de movimentação da cortita, respectivamente subindo e descendo-a. “STOP” é usado para parar o movimento da cortina.

2. Ar condicionado
    - ON
    - OFF
    - SetPoint

Onde “ON”e “OFF” liga e desliga o aparelho, respectivamente. “Set point” é um inteiro entre 18 e 28 cujo valor é equivalente a temperatura em graus Celsius do ar-condicionado da sala.

3. Sensor de presença
    - MOVIMENTO
    - OCIOSO

Onde “MOVIMENTO” indica a ativação do sensor e “OCIOSO” sua inativação.

4. Temperatura e Luminosidade

A temperatura e a luminosidade são informadas no tópico por meio do seu respectivo valor, sendo um número inteiro enviado como *string*.

3.4.2 Página web

Seguindo o requisito R005, escolheu-se um *template* bootstrap gratuito e uso permitido que tivesse disponibiliza de uma interface gráfica simples e boa usabilidade e propicia uma boa manutenção de código.

Escolheu-se o NICE ADMIN com quase 94 mil downloads e disponibilizado por meio do site https://bootstrapmade.com/nice-admin-bootstrap-admin-html-template/ por conter ferramentas interessantes de visualização de dados para implementação futura e possibilidade de inicialmente manter uma interfáce gráfica simples com o usuário.

Dentro das subdivisões da página web, criou-se um arquivo HTML chamado atuadores.html o qual seria responsável pela atuação do usuário nos atuadores disponíveis. A partir desse estágio de desenvolvimento houve aparecimento de problemas nos arquivos dentro do servidor da Azure que não ocorrerem num servidor local. Dado o nível de conhecimento do estagiário e o tempo disponível, a identificação e testes de erros no sistema web Azure se mostrou demorada e ineficiente.

Abaixo pode-se ver parte da interface gráfica de atuação de cada componente. Implementou-se um botão com legendas “On” e “Off” distinguíveis, também, por um padrão de cores.

Figura 15 - Atuadores ativados

Fonte: O próprio autor (2019)

Figura 16 - Atuadores desativados

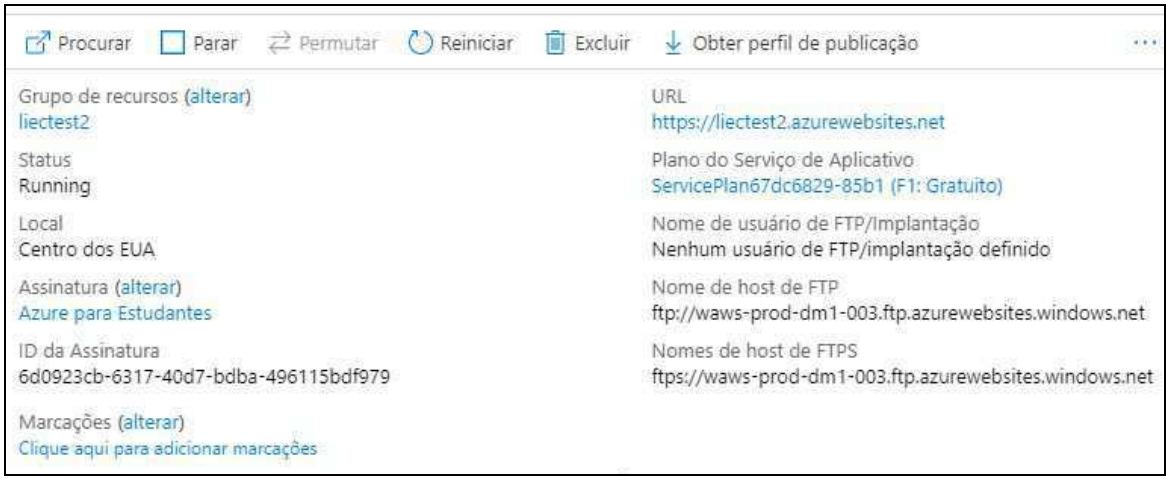
Fonte: O próprio autor (2019)

3.4.2.1 Publicação em serviço de aplicativo Web Azure por meio de protocolo FTP

Para enviar os arquivos desenvolvidos para o servidor da Azure, a documentação disponibilizada indica o passo a passo a ser seguido. É necessário baixar o arquivo disponibilizado no "Obter perfil de publicação". Esse arquivo contém a URL de publicação que deve ser conectada pela porta 21, reservada para o protocolo FTP.

Na figura 17 pode-se ver o lugar para obtenção do arquivo e dados gerais sobre a hospedagem do sistema. Na tabela 2 encontra-se em destaque os três parâmetro necessários para a conexão com o servidor Azure. Utilizou-se o software FileZilla para gerar o cliente FTP e enviar os arquivos.

Figura 17 - Dados da Aplicação Web hospedadas nos servidores Azure



Fonte: Adaptado da tela inicial da plataforma Azure (2019)

Tabela 2 - Parâmetros para comunicação FTP com o servidor de aplicativo web Azure

Chave	Valor
publishUrl	ftp://waws-prod-dm1-003.ftp.azurewebsites.windows.net/site/wwwroot
userName	liectest2\$liectest2
userPWD	tjZYw8JC8qinaxTu7tHon2F3gRFz823DoSJtadacrBsfvxojDLBlpjAtLhHo

Fonte: Adaptado Plataforma Azure (2019)

3.4.3 Banco de dados Cosmos DB

Devido o baixo valor do armazenamento de dados NoSQL no Cosmos DB foi este o formato escolhido de armazenamento.

Na documentação disponibilizada evidência que é gratuita a criação de um banco de dados com apenas um contêiner (similar a tabela) e que o banco de dados tem 100 requisições por segundo gratuitas.

3.4.3.1 Padrão de Armazenamento

Como sugestão ao padrão de armazenamento é dado os seguintes parâmetros: Um campo para um id identificador do dado armazenado; alguns campos para o login e identificação do usuário, como nome, CPF, senha criptografada. E por fim um campo para os tópicos das mensagens MQTT que esse usuário tem acesso.Na figura 18 há um dado armazenado como testes. Campos com "_" antes do nome são reservados e gerados automaticamente pelo sistema.

Figura 18 - Exemplo de um dado armazenado em JSON no Cosmos DB

```
{
    "id": "2",
    "cpf": "19179712061",
    "nome": "Carlos",
    "PSW": "87654321",
    "topicos": [
        {
            "value": "LAMP"
        },
        {
            "value": "LED3"
        },
        {
            "value": "TEMP"
        }
    ],
    "_rid": "5kshAKm2w4gFAAAAAAAAAA==",
    "_self": "dbs/5kshAA==/colls/5kshAKm2w4g=/docs/5kshAKm2w4gFAAAAAAAAAA==/",
    "_etag": "\"01008ed1-0000-0b00-0000-5d9255fb0000\"",
    "_attachments": "attachments/",
    "_ts": 1569871355
}
```

Fonte: O próprio autor (2019)

3.4.3.2 App.js

Para uma aplicação ser comunicar com o banco de dados é necessário um arquivo JavaScript sendo executado pelo Node.JS. São fornecidos a URL de comunicação com o banco de dados, neste caso de teste liectest2bd.documents.azure.com, e uma senha que deve ser enviada.

Após o desenvolvedor instalar o software do Node.JS ele pode executar o arquivo criado e a comunicação com o banco de dados será feita. Na figura 19 vê-se o comando "NODE app.js" o qual executa o arquivo app.js e em seguida o resultado da execução. Nesse caso houve o retorno do nome e id de todos os itens salvos no banco de dados.

Figura 19 - Execução do arquivo APP.JS

Fonte: O prórpio autor (2019)

Para aumentar a segurança e facilitar a manutenção de código, os parâmetros de acesso ao servidor podem ser colocados em um arquivo distinto, preferencialmente chamado de config.js segundo a documentação. Abaixo vê-se o código do arquivo config.js, a linha 3 é a URL de conexão do banco de dados, a linha 4 a senha, a linha 6 e 10 descrevem qual o nome do banco de dados e do container que deseja-se fazer conexão.

```
1. var config = {};
2.
3. config.endpoint = 'liectest2bd.documents.azure.com'
4. config.key =
   'JDmZS1PRVXYUa1o3RbWCoYiPh6UFMyB8HQHL1svdwQ9Obj9WPr0T9Zx
   xz10P8NVf3nlVIYgczDDREJ1U6FzMYA=='
5.
6. config.database = {
7.   id: 'testdb'
8. }
9.
10. config.container = {
11.   id: 'liec_user'
12. }
```

No ANEXO I se encontra o código completo do APP.JS o qual foi executado na figura 19. Pode-se ver no trecho destacado abaixo que é possível fazer uma consulta SQL no banco de dados NoSQL do Cosmos DB. Uma string é passada no campo “query” e enviado como JSON, o Cosmos DB recebe essa requisição e faz uma consulta JSON equivalente ao comando SQL enviado pela string, facilitando o processo de pesquisa.

```
1.    // query SQL
2.    const querySpec = {
3.      query: "SELECT c.nome, c.id FROM c"
4.    }
```

3.4.4 Mensagens MQTT

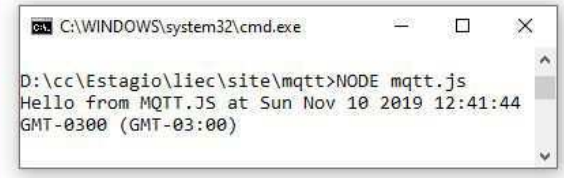
Para seguir o mesmo padrão de desenvolvimento da comunicação com o banco de dados, escrito em JavaScript, codificou-se um arquivo para enviar mensagens MQTT. na mesma linguagem. O arquivo executou sem problemas, como visto nas figuras 20 e 21 porém não estava funcionando quando enviado para o servidor da Azure, apenas localmente.

Caso seja instalado o Node.JS no computador que será o servidor da aplicação dentro do LIEC tanto a comunicação como o banco de dados quanto o envio de mensagens MQTT poderão ser feitas sem problemas pelos, respectivamente, arquivos App.JS e mqtt.js.

Visando o funcionamento imediato da funcionalidade de envio de mensagens MQTT desenvolveu-se um arquivo em uma linguagem diferente, devido ao prazo já perto de acabar e o conhecimento prévio de PHP do estagiário, optou-se por codificar o arquivo mqtt.php o qual funcionou perfeitamente no servidor da Azure. Esses problemas reduziram ainda mais o tempo disponível para o desenvolvimento do sistema de retaguarda.

3.3.4.1 MQTT.JS

Figura 20 - Execução do arquivo MQTT.JS



Fonte: O prórprio autor (2019)

Figura 21 - Recebimento da mensagem enviada

Fonte: O prórprio autor (2019)

```
1. var broker = 'mqtt://mqtt.internetecoisas.com.br'
2. var topico = 'LIEC/Sala_119/LED';
3. var mensagem  = 'Hello from MQTT.JS at ' + Date()
4.
5. //Conecta com o broker
```

```
var mqtt = require('mqtt');
var client  = mqtt.connect(broker);

//Subscribe
client.on('connect', function () {
  client.subscribe(topico, function (err) {
    if (!err) {
      client.publish(topico, mensagem)
    }
  })
})

client.on('message', function (topic, message) {
  // message is Buffer
  console.log(message.toString())
  client.end()
})
```

3.4.4.2 MQTT.PHP

```
<?php
require("phpMQTT.php");

$server = "mqtt.internetecoisas.com.br";     // change if necessary
$port = 1883;                     // change if necessary
$username = "";                   // set your username
$password = "";                   // set your password
$client_id = "phpMQTT-M4rceloMen3ses"; // make sure this is unique for connecting to sever - you could use uniqid()

$mqtt = new Bluerhinos\phpMQTT($server, $port, $client_id);

if ($mqtt->connect(true, NULL, $username, $password)) {
    $mqtt->publish("LIEC/Sala_119/LED", "Hello World! at " . date("r"), 0);
    $mqtt->close();
} else {
    echo "Time out!\n";
}
?>
```

5 Conclusão

Por meio das atividades realizadas durante o estágio supervisionado foi possível aprofundar os conhecimentos a respeito das comunicações entre dispositivos usando o protocolo MQTT e as plataformas de desenvolvimento integrado da *Microsoft*. As atividades realizadas durante o estágio supervisionado permitiram, também, o aluno aplicar os conhecimentos adquiridos nas disciplinas de Informática Industrial e Redes de Computadores.

Devido ao curto tempo disponível para o estágio, não foi viável realizar a implementação do sistema completo de retaguarda, mas a contribuição documentada deste estágio para o laboratório permitirá que futuros alunos aprendam com menos dificuldade a finalizar o sistema implementando as funcionalidades disponibilizadas pela plataforma azure.

5.1 Trabalhos futuros

Como trabalhos futuros é sugerido a finalização das partes do sistemas que ficaram incompletas devido ao tempo curto tempo de 180h do estágio. Entre essas atividades pode-se destacar:

- Fazer o servidor local do LIEC rodar Node.JS
- Integrar o sistema com o banco de dados desenvolvido
- Fazer o sistema capaz de exportar dados

Referências

[1] LAMPKIN, Valerve; LEONG, Weng Tat, et al. **Building Smart Planet Solutions with MQTT and IBM WebSphere MQ Telemetry.** United States: IBM Redbooks, 2012.

[2] OASIS Standart. **MQTT Version 3.1.1**. Disponível em: < http://docs.oasisopen.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf > Acesso em 5 de novembro de 2019.

[3] HIVEMQ (Germany). **MQTT Essentials Part 6: Quality of Service 0, 1, & 2**. Disponível em: . Acesso em: 5 de novembro de 2019.

[4] HIVEMQ (Germany). **MQTT Essentials Part 5: MQTT Topics & Best Practices.** Disponível em: < http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtttopics-best-practices >. Acesso em: 5 de novembro de 2019.

[5] AZURE; **A Node.js SDK for connecting devices to Microsoft Azure IoT services** <https://github.com/Azure/azure-iot-sdk-node> Acesso em 5 de novembro de 2019.

[6] BLUERHINOS; **A simple php class to connect/publish/subscribe to a MQTT broker** <https://github.com/bluerhinos/phpMQTT> Acesso em 5 de novembro de 2019.

[7] MICROSOFT; **Documentação Xamarin Forms** < https://docs.microsoft.com/pt-br/xamarin/xamarin-forms/ > Acesso em 5 de novembro de 2019.

[8] MICROSOFT; **Trabalhar com os dados NoSQL no Azure Cosmos DB** <https://docs.microsoft.com/pt-br/learn/paths/work-with-nosql-data-in-azure-cosmos-db/> Acesso em 5 de novembro de 2019.

[9] AZURE; **Documentação do Portal Azure.** <https://docs.microsoft.com/pt-br/azure/azure-portal/> Acesso em 5 de novembro de 2019.

ANEXO I - App.js

```
// Conectar com o Banco de dados
const cosmos = require("@azure/cosmos");
const CosmosClient = cosmos.CosmosClient;

const endpoint = "https://liectest2bd.documents.azure.com:443/";
const key =
"JDmZS1PRVXYUa1o3RbWCoYiPh6UFMyB8HQHL1svdwQ9Obj9WPr0T9Zx
xz10P8NVf3nlVIYgczDDREJ1U6FzMYA==";

const databaseId = "testdb";
const containerId = "liec_user";
const partitionKey = { kind: 'Hash', paths: ['/Country'] }

const client = new CosmosClient({ endpoint, key })
// Conectar com o Banco de dados END

/**
 * Query the container using SQL
 */
async function queryContainer() {
  console.log(`Querying container:\n${containerId}`)

  // query SQL
  const querySpec = {
    query: "SELECT c.nome, c.id FROM c"
  }

  const { resources: results } = await client
    .database(databaseId)
    .container(containerId)
    .items.query(querySpec)
    .fetchAll()
  for (var queryResult of results) {
    let resultString = JSON.stringify(queryResult)
    console.log(`\tQuery returned ${resultString}\n`)
  }
}

/**
```

```
 * Exit the app with a prompt
 * @param {string} message - The message to display
 */
function exit(message) {
  console.log(message)
  console.log('Press any key to exit')
  process.stdin.setRawMode(true)
  process.stdin.resume()
  process.stdin.on('data', process.exit.bind(process, 0))
}

queryContainer()
  .then(() => {
    exit(`Completed successfully`)
  })
  .catch(error => {
    exit(`Completed with error ${JSON.stringify(error)}`)
  })
```

ANEXO II - Uso dos créditos da conta de estudante Azure

Subscription Cost: $21.52

SERVICE NAME	SERVICE RESOURCE	SPEND
SQL Database	vCore	$18.18
Azure Cosmos DB	100 RU/s	$3.07
SQL Database	Data Stored	$0.26
Log Analytics	Data Ingestion	$0
Storage	Batch Write Operations	$0
Storage	LRS Data Stored	$0
Storage	Read Operations	$0
Storage	Write Operations	$0
Azure App Service	F1	$0
Azure Cosmos DB	Free 100 RU/s	$0
Azure Cosmos DB	Free Data Stored	$0
Bandwidth	Data Transfer In	$0
Bandwidth	Data Transfer Out - Free	$0
IoT Hub	Free Unit	$0
Storage	LRS Data Stored - Free	$0
Storage	LRS Write Operations - Free	$0
Storage	Protocol Operations - Free	$0
Storage	Read Operations - Free	$0

ANEXO III - Código aplicativo Xamarin

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Net.Mqtt;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace MQTTXamarin
{

    // Learn more about making custom code visible in the Xamarin.Forms previewer
    // by visiting https://aka.ms/xamarinforms-previewer
    [DesignTimeVisible(false)]
    public partial class MainPage : ContentPage
    {

        IMqttClient client;
        SessionState sessionState;

        // Variáveis de configuração
        private string Sala = "119";
        private string CurtainPath = "LIEC/Sala_" + Sala + "/Curtain";
        private string AirPath = "LIEC/Sala_" + Sala + "/Air_Conditioner";

        private string BrokerURL = "broker.mqttdashboard.com";
        private int BrokerPort = 1883;
        private string MQTTClientID = "LiecAplicativoXamarin";

        public MainPage()
        {
            InitializeComponent();

        }
```

```
        private async void Button_ClickedAsync(object sender, EventArgs e)
        {
            var configuration = new MqttConfiguration
            {
                BufferSize = 128 * 1024,
                Port = 1883,
                KeepAliveSecs = 10,
                WaitTimeoutSecs = 2,
                MaximumQualityOfService = MqttQualityOfService.AtMostOnce,
                AllowWildcardsInTopicFilters = true
            };

            Botao1.IsEnabled = false;

            client = await MqttClient.CreateAsync(BrokerURL, BrokerPort);
            sessionState = await client.ConnectAsync(new MqttClientCredentials(clientId:
MQTTClientID));
            if (client.IsConnected)
            {
                await DisplayAlert("Conexão", "Estado: Conectado", "Ok");
            }

            Botao1.IsEnabled = false;
            Botao1.Text = "Conectado";
            Xamarin.Forms.Color xfColor = Xamarin.Forms.Color.FromRgb(0, 255, 0);
            Botao1.BackgroundColor = xfColor;

        }

        //------------------------------------------------------

        private async void SendMqttMessage(string mensageMqtt, strint topicoMQTT) {
            var message1 = new MqttApplicationMessage(topicoMQTT,
Encoding.UTF8.GetBytes(mensageMqtt));
            await client.PublishAsync(message1, MqttQualityOfService.AtMostOnce); //QoS0
        }

        // ------------------------------------------------------
```

```
// Cortina ----------------------------------------------------
private void Button_Clicked_CUPAsync(object sender, EventArgs e)
{
   if (client.IsConnected == true)
   {
      SendMqttMessage("UP", CurtainPath);
   }

}

private void Button_Clicked_CDOWNAsync(object sender, EventArgs e)
{
   if (client.IsConnected == true)
   {
      SendMqttMessage("DOWN", CurtainPath);
   }
}

private void Button_Clicked_CSTOPAsync(object sender, EventArgs e)
{
   if (client.IsConnected == true)
   {
      SendMqttMessage("STOP", CurtainPath);
   }
}

// Ar ---------------------------------------------------
private void Button_Clicked_ArONAsync(object sender, EventArgs e)
{
   if (client.IsConnected == true)
   {
      SendMqttMessage("ON", AirPath);
   }

}
```

```
private void Button_Clicked_ArOFFAsync(object sender, EventArgs e)
{
   if (client.IsConnected == true)
   {
      SendMqttMessage("OFF", AirPath);
   }
}

private void Button_Clicked_SetPointAsync(object sender, EventArgs e)
{
   if (client.IsConnected == true)
   {
      string temp = "28";
      temp = setPointText.Text;
      int temperatura = Convert.ToInt32(temp);

      if (temperatura < 18) {
         temp = "18";
         await DisplayAlert("Valor inválido", "A temperatura mínima é de 18 C", "Ok");
      }

      if (temperatura > 28) {
         temp = "28";
         await DisplayAlert("Valor inválido", "A temperatura máxima é de 28 C", "Ok");
      }

      SendMqttMessage(temp, AirPath);
   }
}

// Configurações Avançadas ----------------------------------------------------
private void Button_Clicked_configAvanc(object sender, EventArgs e)
{
   entryBroker.Text = BrokerURL;
   entryTopico.Text = Sala;
   boxViewDarkTrans.IsVisible = true;
   layoutConfigAvanc.IsVisible = true;
}
```

```
        private void Button_Clicked_okConfigAvanc(object sender, EventArgs e)
        {
            boxViewDarkTrans.IsVisible = false;
            layoutConfigAvanc.IsVisible = false;
            Sala = (string) entryTopico.Text;
            BrokerURL = (string) entryBroker.Text;
            DisplayAlert("Dados alterados!","Verifique o tópico e o endereço do broker","Ok");
        }

        private void Button_Clicked_cancelarConfigAvanc(object sender, EventArgs e)
        {
            boxViewDarkTrans.IsVisible = false;
            layoutConfigAvanc.IsVisible = false;
        }
    }
}
```

ANEXO IV - Código aplicativo Xamarin

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             mc:Ignorable="d"
             x:Class="MQTTXamarin.MainPage"
             Title="Cliente MQTT"
             Padding="0, 20, 0, 0">
    <AbsoluteLayout>
        <StackLayout>
            <Image Source="http://brtuning.com.br/images/logoliec.png" />
            <Label
                Text="Atuador MQTT"
                HorizontalOptions="Center"
                FontAttributes="Bold"
                TextColor="Black"
                Margin="0,20,0,0"
                FontSize="20"/>
            <Button Clicked="Button_ClickedAsync" Text="Conectar" x:Name="Botao1"/>
            <Label
                Text="Ar-condicionado"
                HorizontalOptions="Center"
                FontAttributes="Bold"
                TextColor="Black"
                FontSize="14"/>
            <StackLayout Orientation="Horizontal" HorizontalOptions="Center">
                <Button Clicked="Button_Clicked_ArONAsync" Text="ON" x:Name="arON"/>
                <Button Clicked="Button_Clicked_ArOFFAsync" Text="OFF" x:Name="arOFF"/>
                <StackLayout Orientation="Vertical" VerticalOptions="Center">
                    <Entry Text="Temperatura: 18 a 28" Keyboard="Numeric"
x:Name="setPointText"/>
                    <Button Clicked="Button_Clicked_SetPointAsync" Text="Set Point"
x:Name="arSetPoint"/>
                </StackLayout>
            </StackLayout>
```

```
        <Label Text="Cortina"
               HorizontalOptions="Center"
               FontAttributes="Bold"
               TextColor="Black"
               FontSize="14"/>
        <StackLayout Orientation="Horizontal" HorizontalOptions="Center">
          <Button Clicked="Button_Clicked_CUPAsync" Text="UP" x:Name="botaoCUP"/>
          <Button Clicked="Button_Clicked_CDOWNAsync" Text="DOWN"
x:Name="botaoCDOWN"/>
          <Button Clicked="Button_Clicked_CSTOPAsync" Text="STOP"
x:Name="botaoCSTOP"/>
        </StackLayout>
        <Button Clicked="Button_Clicked_configAvanc"
          Text="Configurações Avançadas"
          x:Name="BotaoConfigAvanc"/>
      </StackLayout>
      <BoxView  x:Name="boxViewDarkTrans" IsVisible="False"
           AbsoluteLayout.LayoutBounds="1,1,1,1"
           AbsoluteLayout.LayoutFlags="All"
           Color="#99000000"/>
      <StackLayout x:Name="layoutConfigAvanc" BackgroundColor="White" IsVisible="False"
             AbsoluteLayout.LayoutBounds="0.5,0.5"
             AbsoluteLayout.LayoutFlags="PositionProportional"
             Padding="20">
        <Label Text="Configurações Avançadas" HorizontalOptions="Center"
           FontAttributes="Bold" FontSize="20" Margin="0,0,0,20"/>
        <Label Text="Tópico"/>
        <Entry x:Name="entryTopico"/>
        <Label Text="Broker"/>
        <Entry x:Name="entryBroker"/>
        <StackLayout HorizontalOptions="Center" Orientation="Horizontal">
          <Button Text="Ok" x:Name="okConfigAvanc"
Clicked="Button_Clicked_okConfigAvanc"/>
          <Button Text="Cancelar" x:Name="cancelarConfigAvanc"
Clicked="Button_Clicked_cancelarConfigAvanc"/>
        </StackLayout>
      </StackLayout>
    </AbsoluteLayout>
</ContentPage>
```