Projeto

CampusPark – Smart Campus Parking Platform

Relatório que fundamenta as funcionalidades implementadas no projeto

|  |  |
| --- | --- |
| **Nome:** | Miguel Alendouro |
| **Número:** | 2160853 |
| **Turno prático:** | PL3 |
| **Docente do turno prático:** | Marisa Maximiano |
|  |  |
| **Nome:** | Rúben Lopes |
| **Número:** | 2160852 |
| **Turno prático:** | PL3 |
| **Docente do turno prático:** | Marisa Maximiano |
|  |  |
| **Nome:** | Tiago Batista |
| **Número:** | 2161353 |
| **Turno prático:** | PL3 |
| **Docente do turno prático:** | Marisa Maximiano |
|  |  |
| **Nome:** | Tiago Ferreira |
| **Número:** | 2152150 |
| **Turno prático:** | PL3 |
| **Docente do turno prático:** | Marisa Maximiano |

Índices

1 Introdução 3

2 Providers 3

3 Data Acquisition and Content Enricher (ParkDACE)   3

4 Parking Spot Transmission Unit (ParkTU) 3

5 Parking Spot Storage App (ParkSS) 3

6 Smart Park RESTful Web Service 3

7 Admin Dashboard (ParkDashboard) 3

8 Requisitos não implementados 3

9 Contribuição de cada elemento do grupo 4

10 Conclusão 4

*Anexo A* – Listagem dos Ficheiro XML utilizados 5

*Anexo B* – Outras informações 5

# Introdução

Para o projeto da cadeira de Integração de Sistemas foi pedido que criássemos uma plataforma integrada que usa uma rede de sensores que permite a deteção em tempo real da ocupação de lugares nos parques de estacionamento da [ESTG](https://www.ipleiria.pt/estg/). Esta plataforma inclui as seguintes micro aplicações:

**BOT-SpotSensors** (em SOAP) simula sensores que iriam ser instalados na vida real;

**Data Acquisition and Content Enricher App** (ParkDACE) recebe informações vindas dos sensores e as completa com informação adicional e então redireciona a informação para a plataforma de comunicação;

**Parking Spot Transmission Unit** (ParkTU) transmite informação da aplicação ParkDace para o ParkSS usando o [mosquitto](https://mosquitto.org/) como broker;

**Parking Spot Storage App** (ParkSS) que de forma automática guarda a informação dos sensores e dos parques numa base de dados relacional assim como o histórico de alterações dos *spots* ao longo do dia;

**SmartPark** (RESTFul web service), Api Rest disponível para consulta de dados;

**Admin Dashboard** (ParkDashboard), aplicação usada pelos administradores para verificarem o funcionamento do parque e dos seus respetivos lugares de estacionamento;

# Providers

O BOT é um serviço em SOAP que simula o funcionamento de um sensor real. Para obtermos dados deste serviço podemos optar por duas maneiras distintas, receber os dados em formato de objeto ou em formato XML. Para invocar o serviço basta criar uma nova instância do serviço e escolher a maneira como desejamos receber a informação sendo que

|  |
| --- |
| BotSpotSensor.ServiceBot\_SpotSensorClient service = new BotSpotSensor.ServiceBot\_SpotSensorClient(); |

devolve os dados em formato de XML e

BotSpotSensor.ServiceBot\_SpotSensorClient service = new BotSpotSensor.ParkingSpot();

devolve em formato de objeto. O serviço simula dados sobre um sensor de cada vez, estes dados incluem o seu tipo, valor (se está ocupado ou livre, que é calculado através de um valor *random*), *timestamp* que é a data com hora, minuto e segundo de quando são criados os dados e o estado da bateria que é um valor *random*(1 ou 0) mas que tem mais probabilidade de estar operacional (90%), sendo 1 que precisa de ser substituída a bateria e 0 que está operacional.

# Data Acquisition and Content Enricher (ParkDACE)

Na nossa implementação da plataforma integrada, escolhemos uma *WindowsFormApplication*, como micro aplicação para o ParkDACE. É onde são lidos os ficheiros de configuração nomeadamente o ficheiro de configuração dos *spots* de cada parque ([Campus\_2\_A\_Park1.xlsx](ESTG/IS/Projeto-IS/ParkDace/bin/Debug/Campus_2_A_Park1.xlsx) e [Campus\_2\_B\_Park2.xlsx](ESTG/IS/Projeto-IS/ParkDace/bin/Debug/Campus_2_B_Park2.xlsx)), onde são apresentados os nomes e localizações dos *spots*. É ainda apresentado a quantidade de lugares no parque, é lido lido o IP do broker do ficheiro ([FicheiroConfBroker.xlsx](ESTG/IS/Projeto-IS/ParkDace/bin/Debug/FicheiroConfBroker.xlsx)) e por último é lido um ficheiro que apresenta todas as informações acerca dos parques de estacionamento ([ParkingNodesConfig.xml](ESTG/IS/Projeto-IS/ParkDace/bin/Debug/ParkingNodesConfig.xml)).

Quando se carrega no botão “*Start data acquisition*”, este começa dois timers de 5 segundos, lê o ficheiro XML de configuração do broker e conecta-se ao broker, lê também o ficheiro de configuração do parque para saber onde estão localizados os ficheiros com as localizações dos *spots* associados aos parques e através do broker [mosquitto](https://mosquitto.org/) envia o documento ([ParkingNodesConfig.xml](ESTG/IS/Projeto-IS/ParkDace/bin/Debug/ParkingNodesConfig.xml)).

Quando os timers de 5 segundos acabam são chamadas duas funções de enriquecimento de dados

|  |
| --- |
| public void DataFromDLL (string str) |

onde recebemos os dados originados da DLL fornecida, os transformamos num documento XML e acrescentamos informações tais como a localização do spot, identificador do parque, nome do spot e o tipo.

Para acrescentar dados ao BOT usamos a função

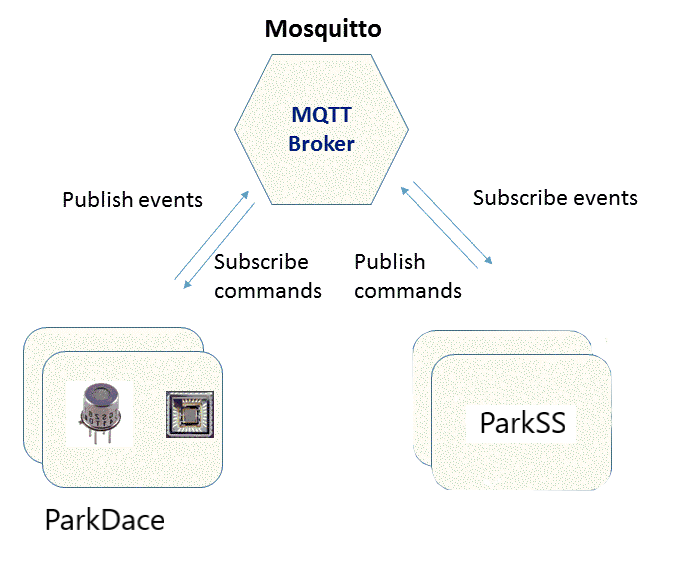
|  |
| --- |
| private void timer1\_Tick (object sender, EventArgs e) |

onde criamos uma instância do BOT, lemos o ficheiro com as localizações e nomes dos *spots* e os acrescentamos aos dados do BOT, no final destas duas funções ambas encaminham o documento XML produzido através do broker [mosquitto](https://mosquitto.org/) no tópico “*DATA*”.

Decidimos usar o XML como formato de dados para uniformizar o processo de leitura e escrita, depois deste processo de enriquecimento dos dados, todos os dados que circulam nesta plataforma têm um formato comum que é o XML, que nos permite utilizá-los mais facilmente.

# Parking Spot Transmission Unit (ParkTU)

O ParkTU usa o [mosquitto](https://mosquitto.org/) como *broker*, isto é, um *broker* de mensagens que implementa o protocolo [MQTT](https://pt.wikipedia.org/wiki/MQTT), que é um método de transmitir mensagens num modelo de “*publish/subscribe*”.

O ParkTU não é uma aplicação por si só, está embutida no ParkDace uma vez que toda a informação está lá disponível. Este conecta-se com o *broker*, envia a informação para ele selecionando qual o tópico e o *broker* redireciona para quem está subscrito naquele tópico, no nosso caso o ParkSS. Temos dois tópicos, um para transmitir informações sobres os parques de estacionamento disponível no tópico “*Parks*” e um tópico referente aos dados dos sensores de estacionamento “*Data*”.

# Parking Spot Storage App (ParkSS)

**Nota**: Não esquecer que têm que entregar a script SQL com o esquema da base de dados (ficheiro .sql)]

O ParkSS é uma *Console Application*, visto que num cenário realista é uma micro aplicação que corre em background e que não necessita de *inputs*.

Para esta micro aplicação decidimos ter três tabelas, uma tabela com informações sobre os parques de estacionamento, uma tabela com informações sobre os *Spots* e outra que guarda o histórico dos *spots*. A aplicação quando começa por ler um ficheiro de configuração onde está o IP do broker ao qual se tem que ligar para receber dados no tópico “*Parks*”. A aplicação recebe detalhes do parque, tais como o nome do parque, a quantidade de lugares de estacionamento, desses lugares, quantos são especiais (para pessoas com algum tipo de deficiência), a descrição da localização do parque e a horas de funcionamento. No tópico “*Data*” a aplicação recebe informações acerca dos *Spots*, tais como o nome, o tipo, o valor, o *timestamp*, o estado da bateria, o identificador do parque a que pertence e a localização, divida em latitude e longitude.

Da primeira vez que a aplicação é executada e não existem dados nas tabelas, ou em caso de falha do sistema de base de dados a aplicação cria todas as tabelas necessárias e preenche-a com dados. Seguidamente a aplicação continua a receber dados, visto que existe muita informação estática e para poupar tempo e recursos quando se recebe informação acerca dos *Spots*, na tabela dos *Spots* só existem três valores a serem atualizados que são nomeadamente, o valor (ocupado ou livre), o estado da bateria para o caso do sensor necessitar de trocar a bateria e o *timestamp*.

Para guardar o histórico e para podermos analisar melhor a ocupação dos parques, temos a tabela *History\_Spots* que guarda o identificador único do Spot, o *timestamp* e o valor.

Com esta estrutura de BD não sobrecarregamos o sistema com *updates* grandes e constantes de informação estática e permite-nos monitorizar e guardar um histórico da plataforma ao longo do tempo.

# Smart Park RESTful Web Service

O SmartPark é uma API Rest para que os administradores tenham acesso aos dados da plataforma, da forma mais prática possível, isto é, disponibilizados através de rotas para o acesso aos dados. A API apenas disponibiliza recursos GET, visto que é só para consulta. Todos os recursos suportados pela API estão disponíveis para consulta no [guia de rotas](file:///Users/rubenfilipe/Desktop/Escola/3ºAno/1ºSemestre/IS/Projeto-IS/Guia%20de%20Rotas.xlsx).

Internamente a API está dividida em dois controladores, o de parques e o de *spots*. Cada controlador tem métodos para consulta de recursos.

O acesso aos recursos depois é feito no ParkDashboard, onde são chamadas as rotas mapeadas pelas funções dos controladores.

# Admin Dashboard (ParkDashboard)

Para o ParkDashboard optámos por um *WindowsFormApplication* para fazer uso dos serviços do SmartPark. Para que as consultas possam ser facilmente feitas e que se obtenham respostas rápidas e precisas.

No Admin Dashboard encontram-se vários separadores com diversas opções para facilitar a procura ao utilizador. São feitos pedidos ao servidor para obter os dados referentes à resposta pretendida. De modo a não sobrecarregar o servidor, após o primeiro pedido ao servidor, a resposta (sendo uma lista ou apenas um só produto (*spot* ou *park*) é armazenado localmente. Dessa forma sempre que precisarmos de apresentar novamente os dados referentes ao mesmo pedido já não se torna necessário fazer conexão com o servidor salvo alguns dos pedidos que têm sempre de ser feitos, visto que podem sofrer alterações com os *updates* à base de dados ao segundo. Existe a possibilidade através de um botão (*refresh*) que torna a atualização dos dados possível. Faz com que a nossa aplicação tenha de fazer novamente pedidos à API, apresentando e armazenando então os possíveis novos dados presentes na nossa base de dados.

# Requisitos não implementados

Todos os requisitos foram implementados.

# Contribuição de cada elemento do grupo

AdminDashboard: Miguel Alendouro, Tiago Batista

BOT-SpotSensor: Tiago Ferreira, Rúben Lopes

ParkDace: Rúben Lopes, Tiago Batista

ParkSS: Rúben Lopes, Miguel Alendouro

SmartPark: Tiago Batista, Tiago Ferreira

Relatório: Rúben Lopes, Tiago Batista

# Conclusão

Este projeto foi bastante desafiador, visto que para além do código ter de ser desenvolvido, também tivemos de tomar decisões estratégicas em relação a cada micro aplicação. Este foi um dos grandes desafios, visto que em projetos anteriores bastou desenvolver o código e não tivemos de tomar decisões.

Pensamos que tenha sido uma mais valia, visto que estas decisões influenciavam a performance e usabilidade da plataforma.

Este projeto teve vários desafios, desde o facto de termos de gerar dados simulados, mas com significado, até ao facto de decidirmos qual a melhor maneira de os armazenar.

A plataforma em geral foi bastante interessante de desenvolver e está preparada para no futuro poder ser expandida, desde ter mais *providers* com dados diferentes até à aplicação final de consumo de dados.

Em geral pensamos que tenha sido um desenvolvimento bastante positivo com um resultado que estamos orgulhosos ainda que tanto a linguagem como o tipo de projeto não fosse um forte de nenhum dos membros do grupo o que provou um desafio interessante de ultrapassar.

#### *Anexo A* – Listagem dos Ficheiro XML utilizados

[Colocar o XSD e respetivo XML para o(s) diverso(s) ficheiro(s) de XML que foram criados no projeto.]

XML:

* [BotDataAfterParkDace](ESTG/IS/Projeto-IS/Ficheiros%20XML/BotDataAfterParkDace.xml)
* [BotSpotSensor](ESTG/IS/Projeto-IS/Ficheiros%20XML/BotSpotSensor.xml)
* [DllAfterParkDace](ESTG/IS/Projeto-IS/Ficheiros%20XML/dllAfterParkDace.xml)
* [ParkingNodesConfig](ESTG/IS/Projeto-IS/Ficheiros%20XML/ParkingNodesConfig.xml)

XSD

* [BotDataAfterParkDace](ESTG/IS/Projeto-IS/Ficheiros%20XSD/BotDataAfterParkDace.xsd)
* [BotSpotSensor](ESTG/IS/Projeto-IS/Ficheiros%20XSD/BotSpotSensor.xsd)
* [DllAfterParkDace](ESTG/IS/Projeto-IS/Ficheiros%20XSD/dllAfterParkDace.xsd)
* [ParkingNodesConfig](ESTG/IS/Projeto-IS/Ficheiros%20XSD/ParkingNodesConfig.xsd)

#### *Anexo B* – Outras informações

[*ConnectionStrings*, Logins e passwords necessárias para testar o trabalho (caso se justifique). Devem também indicar todos os ficheiros de configuração que é necessário alterar para executar o trabalho corretamente.

A base de dados está alojada num servidor [App Harbor](https://appharbor.com), os dados para a BD são:

- Database: dbb1feb347a8564e2a9223a9a7010c7b61

- Hostname: b1feb347-a856-4e2a-9223-a9a7010c7b61.sqlserver.sequelizer.com

- Username: ewvceiluattpbfqh

- Password: 34wKA4rydVBiVZRFhmXf3MuJ5dycnM4YSK8sYpdqJzbx2sbhcpBZwExpGVNxCpkx

[Guia de Rotas](ESTG/IS/Projeto-IS/Guia%20de%20Rotas.xlsx), para consulta;

[Parking Sensor DLL](ESTG/IS/Projeto-IS/BOT-SpotSensors%20(SOAP)/DLL/ParkingSensorNodeDll.dll);

[Ficheiro de Configuração para Broker](ESTG/IS/Projeto-IS/ParkSS/bin/Debug/FicheiroConfBroker.xlsx);

[Ficheiros de criação das tabelas](ESTG/IS/Projeto-IS/Data)