CityStats

Aplicação web e mobile de cidades inteligentes

Diogo Silveira - 85117
Miguel Matos - 89124
Rita Ferrolho - 88822

Introdução

No âmbito do projeto da unidade curricular de IES (Introdução à Engenharia de Software) foi necessário desenvolver uma aplicação, acessível por dispositivos móveis, fixos ou ambos, que utilizasse as tecnologias REST (*REpresentational State Transfer*). Esta tecnologia aborda uma abstração da arquitetura da web, englobando princípios a seguir no desenvolvimento de um projeto, para garantir interfaces bem definidas. O exemplo de utilização mais conhecido desta arquitetura reside no HTTP, o protocolo mais usado da *World Wide Web.*

O software desenvolvido neste trabalho foi evoluindo com base no modelo *Agile Software Development*, uma estratégia que difere dos métodos mais convencionais de engenharia. Neste modelo, o ritmo de trabalho é acelerado. Os requisitos mudam frequentemente, sendo necessário desenvolver código que satisfaça esses requisitos e entregá-lo o mais cedo possível, o que por sua vez altera os requisitos, o que força a desenvolver código que satisfaça esses requisitos e por aí adiante. Resumindo, os requisitos a seguir são instáveis, daí a mudança constante de código, e vice-versa, o que pode explicar o facto de este procedimento ser considerado iterativo. Perante um sistema que tende a possuir imprevistos, não adianta dar tanto ênfase à planificação de um software que, depois de implementado, irá em princípio alterar-se brevemente, portanto uma outra característica deste modelo é dar prioridade ao código em relação ao *design* do software em questão.

Na prática, aplicar a estratégia *Agile* ao nosso projeto significa, por exemplo, realizar *commits* (envios) frequentes numa plataforma (exemplo: Github), mesmo que essas atualizações sejam pouco significativas.

As seguintes etapas foram aplicadas ao modelo *Agile*:

1. Planeamento - associado à prioritização, definição dos sistemas e viabilidade;
2. Análise de requisitos - decidir o que o produto faz por levantamento de requisitos, conhecidos por *user stories* (casos de utilização);
3. *Design* - criar de um plano para a construção do produto:
    - *Frontend* (protótipo);
    - *Backend* (arquitetura).
4. Implementação - desenvolvimento de código e programação:
    - *Frontend* (desenvolvimento web);
    - *Backend* (base de dados).
5. Validação/Testes.

A descrição das etapas e o *walkthrough* da criação do produto em geral serão abordados de seguida.

Planeamento

Para dar início ao projeto, foram atribuídos aos membros da equipa os *roles*. Todos os elementos do grupo desempenharam a função de *developer*, para além dos *roles* que cada um escolheu, como mostra a tabela a seguir:

Diogo	*arquitect*
Miguel	*devOps master*
Rita	*team manager*, *product owner*

Num modo sucinto, o *arquitect* é responsável pelo desenho da arquitetura do sistema a implementar, ou seja, realiza a etapa do design do *backend*; o *devOps master* funciona como uma ponte entre os desenvolvedores e os operadores; a *team manager* executa a gestão da equipa, distribuindo as tarefas, averiguando o progresso do trabalho e ajudando sempre que necessário, de modo a cumprir a *deadline*; o *product owner,* como conhece em detalhe as funcionalidades do produto encarrega-se de criar os seus requisitos e é quem decide aceitar ou não as soluções que vão sendo incrementadas no desenvolvimento do produto.

Com a distribuição dos *roles* concretizada, o instinto imediato do *devOps master* foi criar um repositório no *github,* a anunciar a inicialização do projeto.

O tema do projeto foi-nos opcional, portanto a etapa seguinte consistiu em “atirar para a mesa” ideias soltas, com o intuito de chegar a um consenso acerca do tema final. Devido ao nosso interesse pelo conceito das cidades inteligentes, particularmente na monitorização das concentrações de gases atmosféricos, dos níveis de ruído e da temperatura, decidimos desenvolver um produto que nos fornecesse esses dados estatísticos, medidos em diferentes locais e momentos.

Análise de Requisitos

Uma vez selecionado o tema, o *product owner* descreveu em detalhe as funcionalidades do produto através de requisitos, ou mais concretamente, *user stories* (casos de utilização). Basicamente, são propostos cenários minuciosos no estilo de mini-histórias imaginadas, mas realistas. Os *user stories* devem incluir o nome da *persona* envolvida, a sua profissão, os seus objetivos e as tarefas necessárias para satisfazer esses objetivos. É de notar que não convém englobar diversas tarefas num só *user story*; deve-se "partir" as tarefas em porções mais reduzidas até serem praticamente indivisíveis e distribuir essas porções para diferentes *user stories.*

O levantamento dos requisitos foi realizado com o apoio da ferramenta *Pivotal Tracker.* Uma vez criado um novo projeto no qual se incluíram todos os membros do grupo, fez-se o levantamento dos requisitos ilustrados nas figuras x e x:

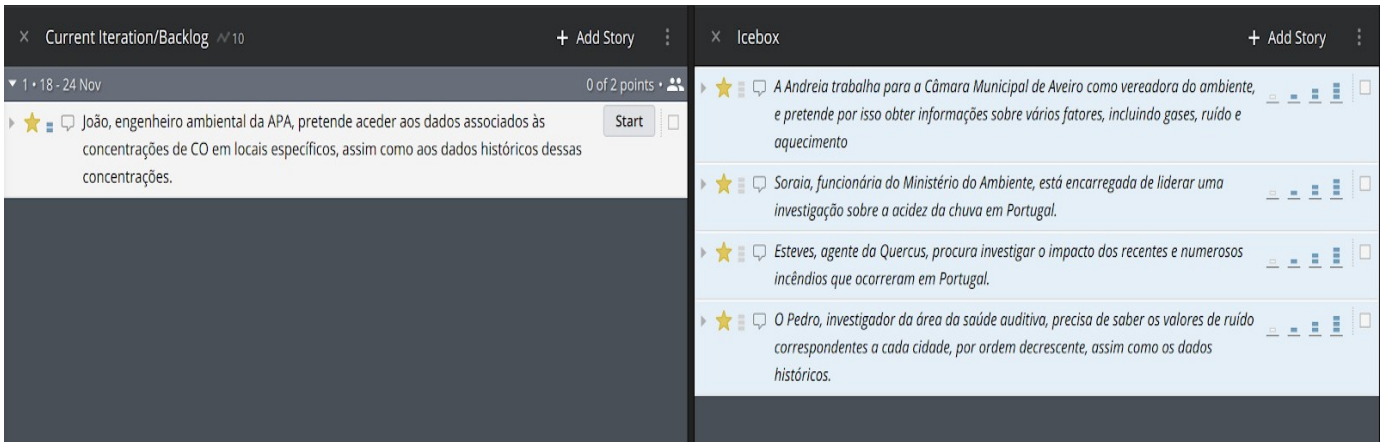


Fig x - *User stories* contidos em *Current Iteration/Backlog.*

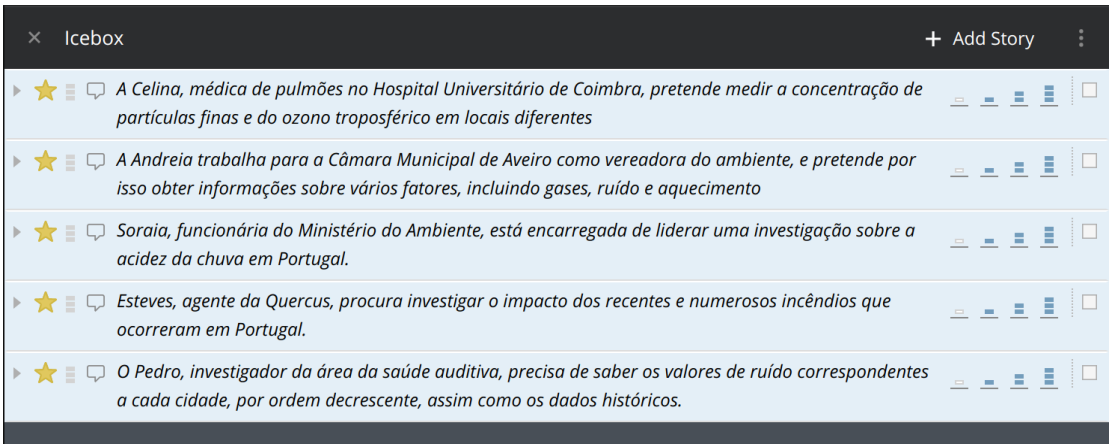


Fig x - *User stories* contidos em *Icebox.*

Os requisitos contidos no *Current Iteration/Backlog* envolve aqueles que estão a ser implementados no momento, e os que estão incluídos no *Icebox* encontram-se pendentes. Cada tarefa a realizar é estruturada com: um título breve com foco na tarefa a realizar; uma escala heurística de zero a três que avalia a dificuldade da tarefa, sendo zero a tarefa mais fácil e três a mais difícil; uma descrição que corresponde ao *user story*.

O João, engenheiro ambiental da APA, pretende aceder aos dados associados às concentrações de CO em locais específicos, assim como aos dados históricos dessas concentrações

Descrição: O João necessita de obter informações sobre as condições ambientais de diversas cidades para um projeto que pretende reduzir as emissões de CO (Monóxido de Carbono) para a atmosfera nas principais cidades portuguesas. Para isso, o João gostaria de ter acesso a uma plataforma que lhe fornecesse informações instantâneas sobre a concentração de CO em localizações específicas, assim como dados históricos para monitorizar a evolução desses valores.

O Pedro, investigador da área da saúde auditiva, precisa de saber os valores de ruído correspondentes a cada cidade, por ordem decrescente, assim como os dados históricos

Descrição: O Pedro está a fazer um projeto sobre a poluição sonora em Portugal, sendo para isso necessário obter informações acerca do estado de diversos locais do país no que toca a emissões de ruído. Para saber onde a situação é mais preocupante, interessa-lhe que os valores correspondentes a cada cidade apareçam ordenados por ordem decrescente, assim como a obtenção de dados históricos e tratados de modo a retirar conclusões relativas à evolução destes valores.

A Andreia trabalha para a Câmara Municipal de Aveiro como vereadora do ambiente, e pretende por isso obter informações sobre vários fatores, incluindo gases, ruído e aquecimento

Descrição: A Andreia preocupa-se com o estado do ambiente na Cidade de Aveiro, pelo que lhe interessa ter acesso a uma plataforma onde pudesse obter informação acerca de diversos fatores relativos ao estado do ambiente, como por exemplo: ruído, poluição do ar (CO, óxidos de enxofre e óxidos de azoto) e aquecimento.

O Esteves, agente da Quercus, procura investigar o impacto dos recentes e numerosos incêndios que ocorreram em Portugal

Descrição: O Esteves procura investigar o impacto dos recentes e numerosos incêndios que ocorreram em Portugal. Quer saber o seu efeito na qualidade do ar (através das quantidades de monóxido de carbono (CO), óxido nítrico (NO3) e dióxido de carbono (CO2)), assim como o tempo que se leva até recuperar dos danos causados pelos incêndios. O seu trabalho seria facilitado com o acesso a dados via web relativos a várias localizações estratégicas relevantes para o seu trabalho.

Soraia, funcionária do Ministério do Ambiente, está encarregada de liderar uma investigação sobre a acidez da chuva em Portugal

Descrição: A Soraia tem o objetivo de recolher dados um pouco por todo o país, não só diretamente relacionados com a acidez da chuva mas também com a concentração de óxidos de enxofre no ar, de modo a perceber quais são os locais onde a situação é mais crítica e a forma como a poluição nuns lugares pode afetar o estado do ambiente noutros.

A Celina, médica de pulmões no Hospital Universitário de Coimbra, pretende medir a concentração de partículas finas e do ozono troposférico em locais diferentes

Descrição: A Celina está a realizar, em conjunto com alguns colegas um estudo sobre a relação entre a concentração de partículas finas e ozono troposférico e a incidência de doenças na região toráxica. A sua estratégia seria medir estas concentrações em diferentes locais e comparar as estatísticas sobre a incidência destas doenças, fazendo depois as devidas comparações para tirar conclusões. No entanto, faltam-lhe dados sobre estes fatores ao longo do país, pelo que seria de grande ajuda obter pelo menos os fatores ambientais via Web.

Design

Protótipo:

Uma vez terminados os requisitos, tornou-se possível desenhar um protótipo com as interfaces gráficas do produto inspirado nos *user stories*, que considera não só o modo como a interface é estruturada, mas também as possíveis interações entre o cliente e a *UI* (*User Interface*).

Um protótipo pode ser esboçado em papel ou usando um software adequado, como o *proto.io*. Não existe um protótipo "melhor" que outro, ambos os estilos têm os seus benefícios e contratempos. O protótipo em papel é mais rápido e fácil de ser esboçado, possui um menor custo ao ser criado e são captados com maior facilidade certos erros de prototipagem. Já o protótipo digital, embora exija mais tempo e investimento para ser construído, consegue filtrar um maior número erros de design, inclusive problemas mais minuciosos, que o anterior.

Familarizados com estas diferenças de prototipagem, optámos pelo protótipo digital, como é mostrado na figura x.

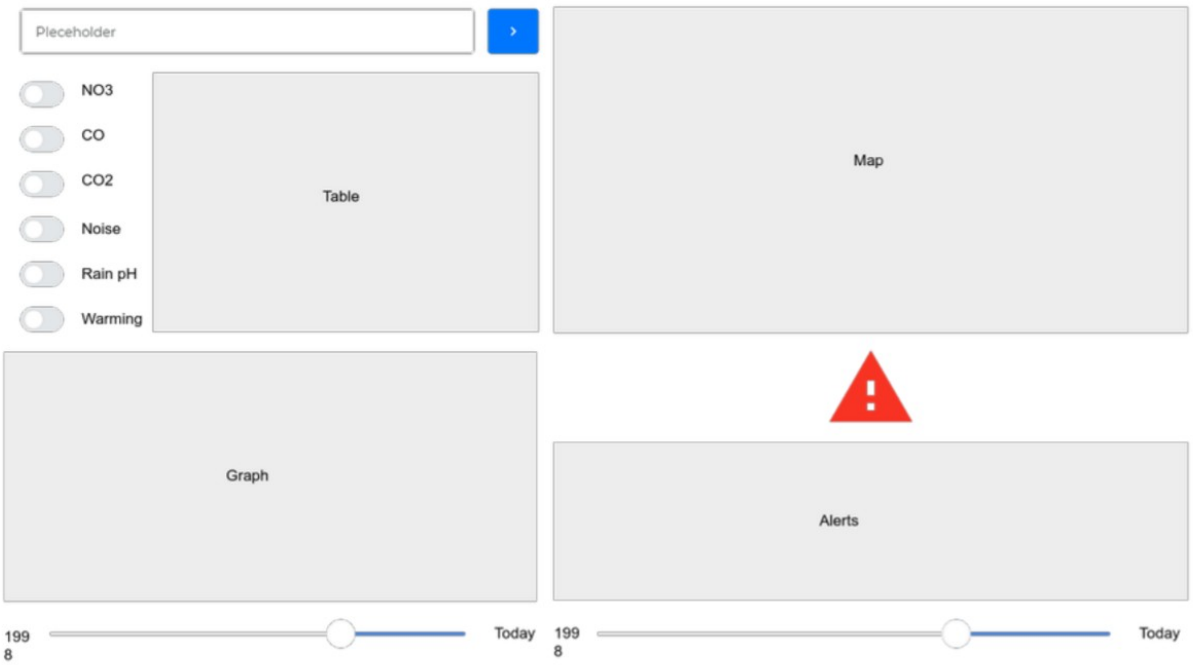


Fig x - Protótipo do produto *CityStats.*

Arquitetura:

Em paralelo com o protótipo, o arquiteto efetuou o design da arquitetura do produto, cuja implementação será a componente mais relevante deste trabalho. O que se pretende é obter dados de sensores externos localizados ao longo do país, de modo a juntá-los no servidor, guardá-los e tratá-los. Após esse processo, os dados ficam disponíveis para serem acedidos via REST ou através de qualquer Browser. Esta foi a arquitetura pensada inicialmente:

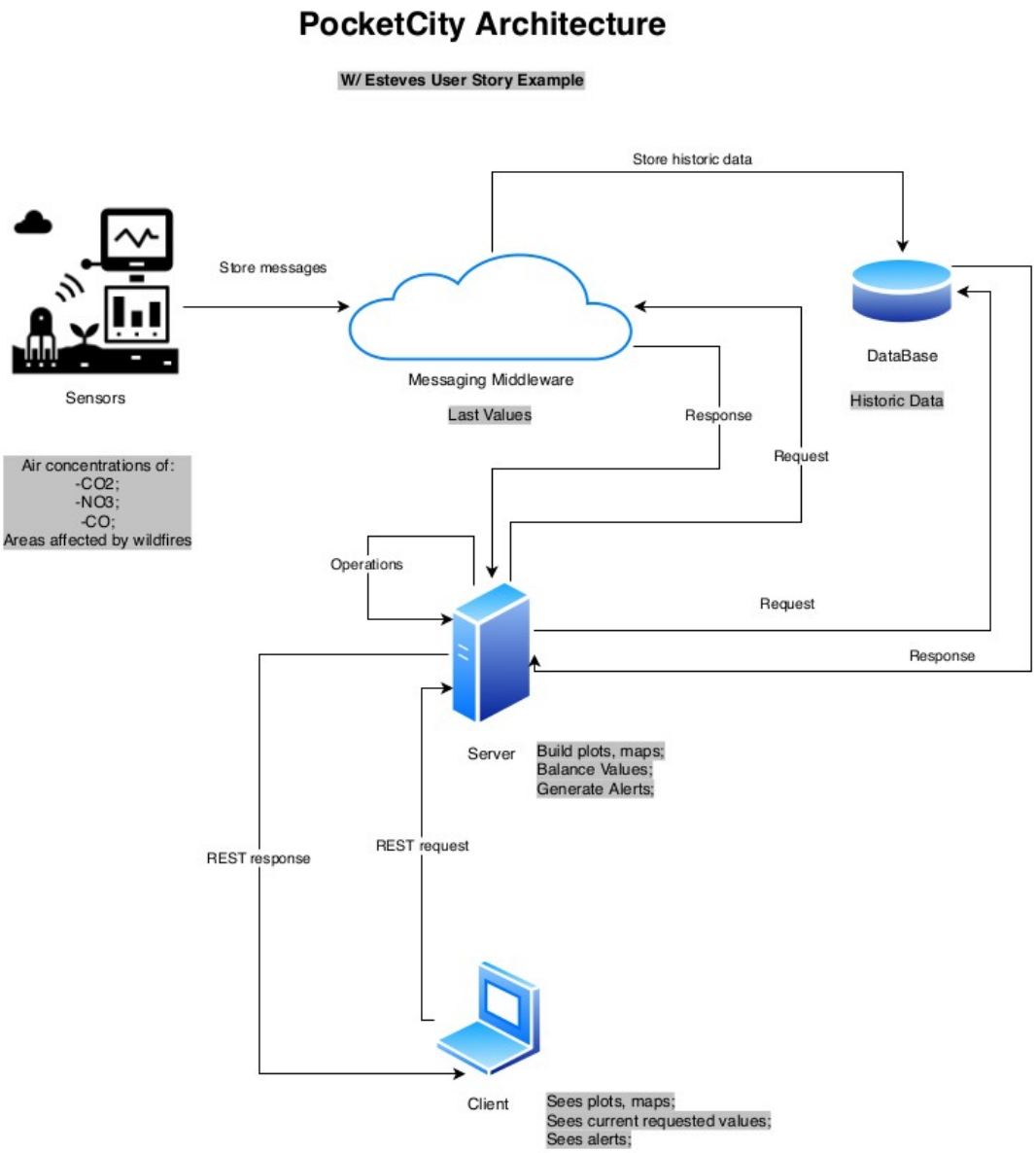


Fig x - Arquitetura do produto.

Implementação

Persistência de dados:

Como Sistema de Gestão de Base de Dados é optou-se pelo MySQL, que corre num container individual. A interação com o MySQL é implementada usando o módulo JPA do Spring. Existem duas entidades na aplicação: as cidades (City) e as estatísticas (Stat)

Ambas são unicamente identificadas pelo seu ID numérico, gerado no caso das estatísticas e atribuído pelo sensor no caso das cidades.

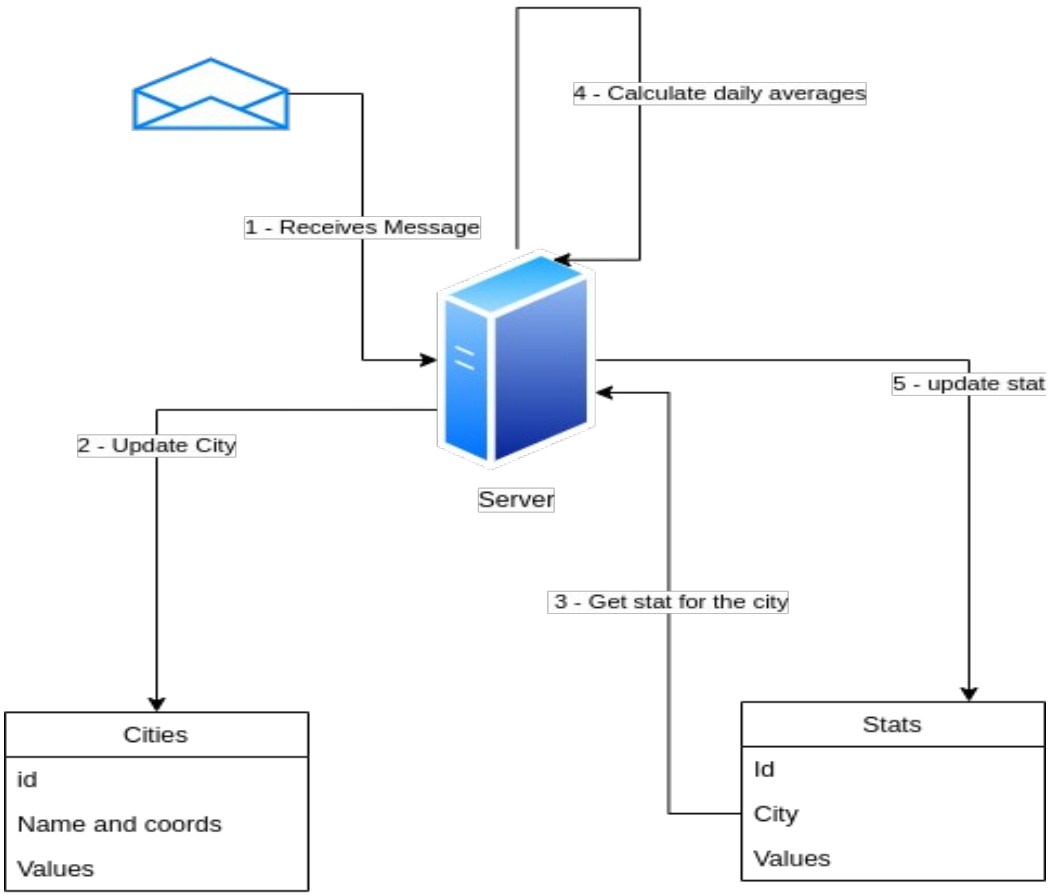
As cidades guardam a sua identificação (id interno da aplicação, nome e distrito), as suas coordenadas e os seus dados relativos à poluição atmosférica e sonora. Os dados ambientais guardados em cada cidade são os últimos a terem sido recebidos dos sensores.

Às estatísticas cabe guardar informação histórica.

Existe uma estatística para cada cidade e dia. Ao longo do dia, à medida que vão sendo recebidos dados relativos a uma cidade, é calculada a média dos valores ao longo desse dia para essa cidade. Quando o dia muda é acrescentada uma nova *row* na tabela para a respetiva cidade e o novo dia. Assim é possível guardar dados úteis para fazer cálculos e analisar o estado do ambiente a longo prazo, guardando apenas os valores para cada cidade e cada dia, sem criar uma nova estatística cada vez que são recebidas novas atualizações.

Existem portanto duas tabelas na base de dados.

Criámos um repositório JPA para cada uma delas. Estes repositórios pertencem à camada de persistência.



Métodos do StatsRepository:

• findById(Long id) → obter estatísticas pelo id; herdado;

• findByDayAndCity(Date day, City city) → obter estatísticas para um dado dia numa dada cidade;

• findByCity(City city) → obter todas as estatísticas para determinada cidade;

• findAllByDayAndCity(Date day, City city) → obter todas as estatísticas para um dado dia numa dada cidade (incluindo repetidas);

Métodos do CityRepository:

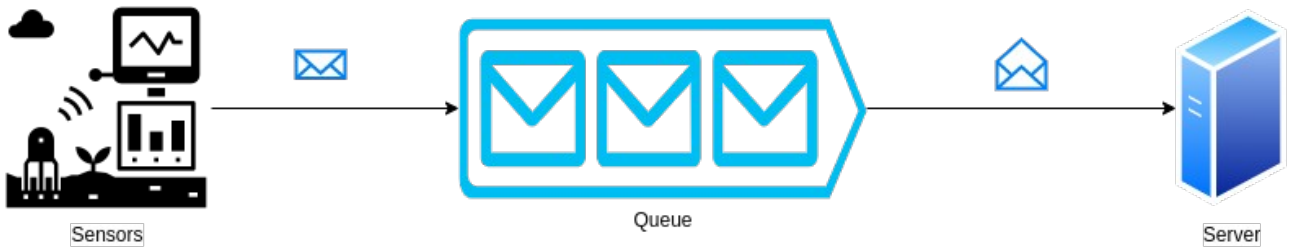
• findById(Long id) → obter cidade com base no seu id; herdado;

• findByNameAndDistrict(String name, String district) → obter cidade pelo seu nome e distrito;

• findByLatAndLong(double lat, double lon) → obter uma cidade pelas suas coordenadas;

Receção de mensagens dos sensores:

Pretende-se que os sensores que recolhem os valores enviem continuamente as últimas atualizações para o servidor que posteriormente processa esses dados e guarda a informação. A estratégia que foi usada na implementação deste sistema de mensagens consistiu na utilização de um *message broker* – RabbitMQ, a correr num *container* individual tal como o SGBD – usando uma única *queue* que recebe mensagens em JSON, relativas às diferentes cidades, todas com a mesma estrutura, das quais o servidor obtém as atualizações ao estado dos fatores ambientais tratados pela aplicação. É usado um único *receiver* para as mensagens dos sensores. Ao receber uma mensagem este usa os dados recebidos para atualizar os valores atuais da cidade e criar uma nova estatística. Essa nova estatística é usada para inicializar o registo do dia para a cidade, se ainda não tiver sido iniciado, ou para fazer média com os valores já existentes, em caso contrário. A *queue* do RabbitMQ é um FIFO, o que significa que as mensagens recebidas são lidas pela ordem em que entram na *queue*, o que garante valores fiáveis e atualizados do ponto de vista do servidor. Tendo em conta a interação com o *message broker* e o facto de haver, por parte do *receiver*, um processamento da informação obtida, podemos dizer que o *receiver* está na camada de negócio/operações tratando também da ligação à *queue* (classe utilitária) que armazena as mensagens do *broker*.



Apresentação:

A camada de apresentação é implementada através de uma REST Api e de uma página Web.

A REST Api é conseguida através de dois controllers, um tratando das cidades (CityController) e outro tratando das estatísticas (StatsController). Tratam-se de RestControllers do Spring, ou seja são controllers construidos especificamente para implementarem o padrão REST.

CityController:

É responsável pelos pedidos diretamente relacionados com a tabela das Cidades. Todos os pedidos a ele direcionados possuem um path que começa com /cities. É possível efetuar os seguintes pedidos:

- GET <*hostname*>/cities – obter todas as cidades e os valores atuais de cada uma;
- GET <*hostname*>/cities/{id} – obter uma cidade e os seus valores atuais com base no seu id;
- GET <*hostname*>/cities/{name}/{district} - obter uma cidade e os seus valores atuais com base no seu nome e distrito;
- GET <*hostname*>/cities/coords/{lat}/{lon} - obter uma cidade e os seus valores atuais com base nas suas coordenadas;
- POST <*hostname*>/cities – adicionar uma nova cidade;
- PUT <*hostname*>/cities/{id} - modificar uma dada cidade;
- DELETE <*hostname*>/cities/{id} – apagar uma dada cidade;

StatsController:

O StatsController tem como função tratar dos pedidos diretamente relacionados com a tabela das Estatísticas. O path dos seus pedidos começa sempre com /stat. É possível efetyar os seguintes pedidos:

- GET <*hostname*>/stats – obter todas as estatísticas existentes;
- GET <*hostname*>/stats/{id} – obter uma única estatística pelo seu id;
- GET <*hostname*>/stats/day/{day} – obter todas as estatísticas relacionadas com um dado dia;
- GET <*hostname*>/stats/city/{id} – obter todas as estatísticas relativas a uma cidade usando o id desta;
- GET <*hostname*>/stats/day/{date}/ city/{id} – obter a estatística de determinado dia para uma dada cidade, com base no id desta;
- GET <*hostname*>/stats/city/{name}/district/{district} – obter todas as estatísticas relativas a uma cidade, usando o seu nome e distrito;
- *GET* <*hostname*>/stats/city/{name}/district/{district}/{date} – obter a estatística relativa a uma cidade para um determinado dia, com base no nome e no distrito da cidade, para além, evidentemente, da data.
- POST <*hostname*>/stats – criar uma nove estatística;

- PUT <*hostname*>/stats/{id} – alterar uma dada estatístca, com base no seu id;
- DELETE <*hostname*>/stats/{id} – apagar uma dada estatística, com base no seu id;

O componente responsável pela página Web é o UIController. Esta não se trata de um RestController mas de um controller "convencional", ou seja retorna o nome do documento a ser apresentado ao utilizador, isto após preencher um model com os dados a serem apresentados nesse documento. Para associar a informação do model aos devidos campos dos documentos HTML foi usado o Thymeleaf.

Beans:

Os *beans* instanciados pelo IOC Container são os *controllers* (ambos os *RestControllers* e o UIController), usados para apresentar informação a utilizadores, e o Receiver e a Queue, usados para receber e tratar as mensagens dos sensores.

Containers:

Cada um dos três serviços – a base de dados, o *message broker* e o servidor - que constituem esta aplicação corre num *container* em separado. Foi usado o Docker como ferramenta de virtualização em *containers*.

Sensores:

Os sensores enviam dados para o servidor, em formato JSON, através do RabbitMQ. São tratadas as seguintes variáveis:

- CO (monóxido de carbono);
- CO2 (dióxido de carbono);
- NO2 (dióxido de azoto);
- SO2 (dióxido de enxofre);
- O3 (ozono troposférico);
- Noise (poluição sonora);
- RainPh (pH da chuva);
- Temperature (temperatura);

NOTA: para simplificar essa parte do projeto, os sensores foram simulados através de um *script* em python3 que, de 20 em 20 segundos, gera valores aleatórios para cada uma destas variáveis e coloca-os, em formato JSON na Queue do RabbitMQ.

Framework

Como suporte para este projeto foi usado o Spring Framework, e a sua extensão, o Spring Boot, que simplifica muito o trabalho de quem está a desenvolver um serviço deste tipo, fornecendo toda uma infraestrutura, sendo apenas necessário construir código relevante, sem necessidade de grandes configurações. É de notar o conceito de convenção em vez de configuração – o uso de convenções que não só evitam a necessidade de perder tempo a configurar o ambiente de desenvolvimento como uniformiza diversos projetos construidos usando a mesma ferramenta, facilitando a sua compreensão, a extensibilidade, implementação e a correção de eventuais erros. O Spring Framework fornece também um servidor embutido, o que poupa trabalho a implementar o código desenvolvido num servidor standalone. É apenas necessário criar um ficheiro .jar e não um ficheiro .war.

Maven

Para compilar o código e criar um .jar executável foi usado o Maven, que se baseia nas configurações do pom.xml para construir o projeto de forma automática.

Dependências usadas:

- **Starters:**
  - spring-boot-starter-amqp
  - spring-boot-starter-thymeleaf 2.1.9
  - spring-boot-starter-data-jpa
  - spring-boot-starter-web
  - spring-boot-starter-test
- mysql-connector-java
- junit-vintage-engine
- spring-rabbit-test
- jackson-databind 2.10.1

Resultado

Como resultado deste trabalho obteve-se um serviço Web, com uma REST Api e uma interface gráfica relativamente simples.

Através da REST Api é possível obter diversos dados e estatísticas em formato JSON, possibilitando a integração com outros projetos de tema semelhante.

A interface gráfica, que consiste numa página Web, apresenta uma tabela com os valores mais recentes obtidos para cada uma das cidades abrangidas pelo projeto, e, ao clicar no nome de uma das cidades, permite navegar para uma outra página que mostra valores históricos relacionados com essa cidade, permitindo fazer uma análise estatística e tomar medidas se necessário.

Conclusão

Com a realização deste trabalho ficamos a conhecer melhor o processo de implementação e o funcionamento de um serviço Web.

Ganhámos noções relativamente às ferramentas as serem usadas, à infraestrutura dos projetos, e aos modelos de desenvolvimento e de negócio associados a estes serviços.

Ficámos, portanto melhor preparados para no futuro trabalhar numa empresa, em equipa, para desenvolver uma aplicação deste género, o que era o objetivo primário deste trabalho.

Ferramentas e Bibliografia

Pivotal Tracker:

- https://www.pivotaltracker.com

Noções de REST:

- https://becode.com.br/o-que-e-api-rest-e-restful/
- https://pt.wikipedia.org/wiki/REST

Spring Boot:

- https://spring.io/projects/spring-boot

RabbitMQ:

- https://www.rabbitmq.com

MySQL:

- https://www.mysql.com