

## 4 Lab: Modelação de interações

### Enquadramento

Os diagramas de comportamento da UML permitem modelar situações em que se desenrola uma colaboração relevante (e.g.: entre objetos do software). Para além do **tempo/fluxo**, os diagramas mostram também as partes intervenientes. O diagrama de comportamento largamente mais usado é o diagrama de sequência que se adequa especialmente bem ao raciocínio “por objetos”.

#### Objetivos de aprendizagem

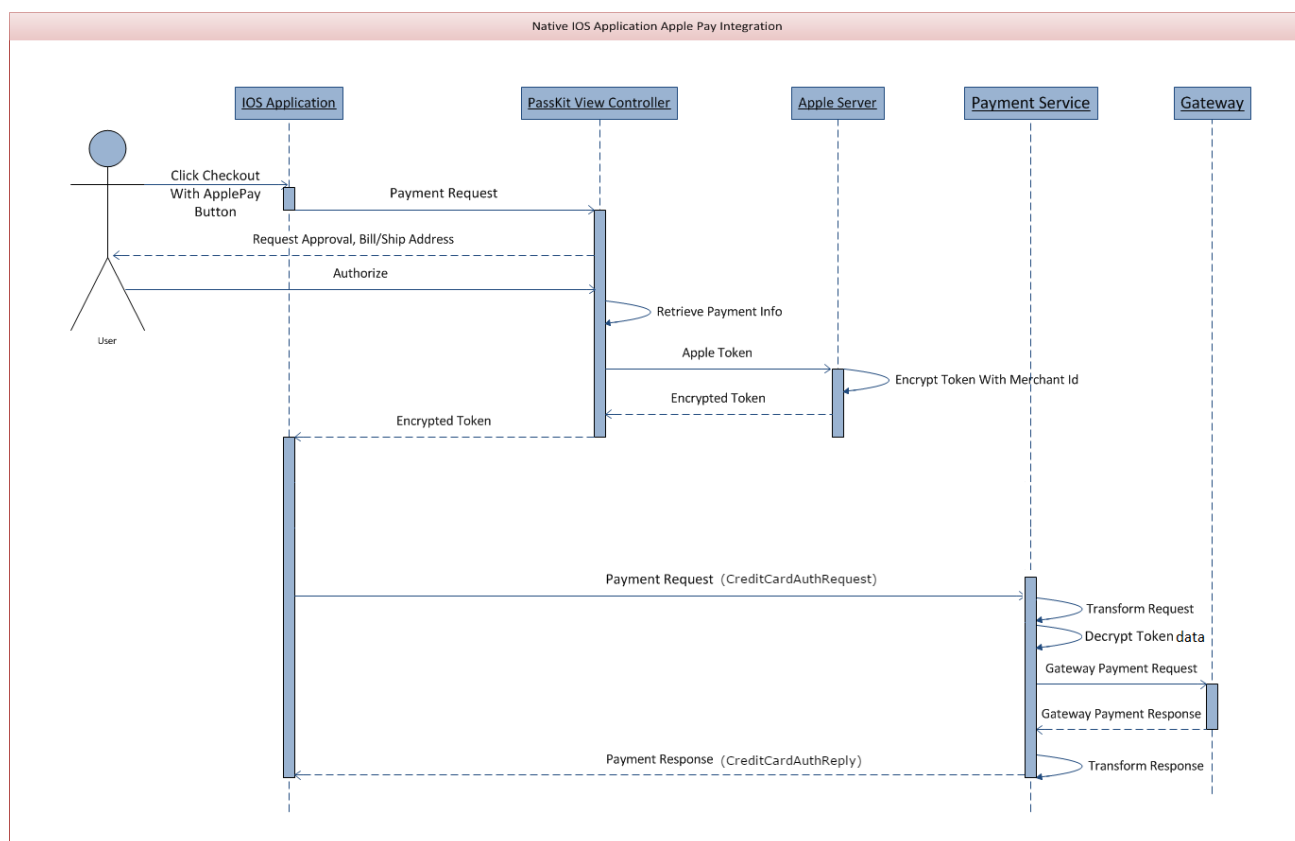
- Explicar a colaboração entre objetos necessária para implementar uma interação de alto nível ou uma funcionalidade de código, recorrendo a diagramas de sequência.
- Representar a evolução de estados de um objeto como uma máquina de estados.

#### Preparação

- “[sequence diagrams](#)” - informação tutorial.

### 4.1

Explique a interação modelada no Diagrama 1.



**Diagrama 1: Integração de aplicações nativas (iOS) com o serviço Apple Pay.** [In: <https://docs.radial.com/ptf/Content/Topics/payments/apple-integration.htm>]

## 4.2

Considere o caso de utilização que o grupo descreveu com a narrativa detalhada (“*fully-dressed*”) no Lab 2.

A partir desse resultado, prepare um Diagrama de Sequência de Sistema (DSS)<sup>1</sup>. Note que o sistema deve ser visto como uma caixa fechada, procurando-se identificar as operações que o sistema deve expor para realizar o caso de utilização em apreço.

## 4.3

O projeto “AirQuality” permite pesquisar a previsão de qualidade do ar (e da previsão meteorológica) para uma zona. Pode experimentar o [projeto no Heroku](#)<sup>2</sup> e aceder ao código no [GitHub](#).

O *backend* do projeto está foi desenvolvido em Java e usa a “*build tool*” do Maven<sup>3</sup>. A solução disponibiliza uma API que permite interrogar a qualidade do ar, e.g.:

— `http://localhost:8080/today/?address=Aveiro`

O pedido acima levará à invocação do método:

— `tqs.airquality.app.controller.AirQualityRestController#getAirQualityOfTodayFromCoordinates`

- a) apresente um diagrama de sequência para ilustrar a colaboração entre objetos que deve acontecer quando o utilizador invoca o método `getAirQualityOfTodayFromCoordinates`<sup>4</sup>.

Nota: o diagrama foca-se na troca de mensagens entre objetos próprios desta implementação (geralmente, omite-se o envolvimento de tipos de objetos da linguagem/JDK, como String, etc).

- b) visualize as classes diretamente envolvidas na interação anterior num diagrama de classes<sup>5</sup>, mostrando os relacionamentos entre elas que possam existir.

## 4.4

O projeto anterior utiliza o conceito de *cache* para otimizar as operações: guarda os resultados da previsão da qualidade do ar para uma localização, durante um tempo designado (*time to live* – TTL). Surgindo um pedido “idêntico”, é respondido com os resultados anteriores guardados na *cache* (se ainda estiverem válidos), em vez de chamar a(s) API remota(s).

Independentemente dessa implementação, considere que:

- Um *put* [inserção] de um resultado [i.e., uma previsão para uma dada localidade] que ainda não existia, cria uma nova entrada [válida] na *cache*, e atualiza a marca temporal [*timestamp*]
- Um *put* de um resultado que já existia em *cache*, atualiza o *timestamp* daquela entrada.
- Um *get* de um resultado existente na *cache*, mas já expirado, leva à sua remoção da *cache*.
- Um *get* de um resultado que não existe na *cache*, leva a um “*cache miss*” (mas não altera

---

<sup>1</sup> Os DSS não são um conceito da UML, mas uma abordagem proposta por C. Larman, como “preparação” para o desenho por objetos de uma solução.

<sup>2</sup> Pode haver limites à utilização, associadas ao “*free tier*” do Heroku...

<sup>3</sup> Num projeto Maven, o código encontra-se em “**src/main/java/...**”. Para abrir o projeto localmente, deverá fazê-lo num IDE preparado para usar o Maven, e.g.: IntelliJ ou VS Code com as extensões do Java. Embora útil, não é necessário ter um ambiente de desenvolvido no seu computador, para abrir o projeto localmente. Pode “navegar” no código a partir do repositório.

<sup>4</sup> Para a resposta, precisa de analisar a implementação do método e identificar a “colaboração” de outros objetos [que são chamados a partir deste método].

<sup>5</sup> Não se pretende mostrar todas as classes do projeto, mas as que estão implicadas na implementação do método considerado na alínea anterior.

as entradas na *cache*).

- Passado o tempo TTL definido para uma entrada, ela passa a inválida [marcada como “dirty”], mas permanece na *cache*.
- Periodicamente, a *cache* é reavaliada e as entradas inválidas são retiradas.

Modele a máquina de estados associada a cada elemento [i.e., entrada] da *cache*.