DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

LEIC-A, LEIC-T, LETI

Engenharia de Software

2º Semestre – 2018/2019

Enunciado Geral do Projecto

O objetivo do projeto é melhorar o sistema ADVENTURE BUILDER. Este sistema deverá permitir a integração de diferentes aplicações com vista à oferta de atividades de lazer.

O que se segue é uma descrição geral do projeto que será objeto de estudo durante o semestre. Aspetos concretos do projeto a realizar serão descritos em enunciados ao longo do semestre. As partes descritas neste enunciado e que não sejam pedidas nas várias fases a realizar não necessitam de ser concretizadas e apenas servem para contextualizar o problema.

1. Introdução

O sistema ADVENTURE BUILDER tem como funcionalidade principal a integração de diversos fornecedores de serviços com vista à oferta de atividades de lazer. Adicionalmente, o sistema permite a reserva de aventuras, nome que iremos usar para referenciar o conjunto de serviços associados a uma atividade de lazer. A concretização deste sistema é alcançada através da integração de 6 aplicações: BROKER, HOTEL, ACTIVITY, CAR, BANK e TAX.

Cada uma destas aplicações é um componente independente e possui uma interface web através da qual os utilizadores podem interagir (ver Figura 1). Adicionalmente, a aplicação BROKER interage com as restantes aplicações por forma a integrar os serviços que elas disponibilizam. Por outro lado, todas as aplicações que prestam serviços, BROKER, ACTIVITY, CAR e HOTEL, interagem com as aplicações BANK e TAX por forma a, respetivamente, fazerem pagamentos e enviarem as faturas dos itens transacionados. Estas interações remotas entre aplicações estão implementadas nos *packages* `services.remote`.

Uma interação típica com início na aplicação BROKER é iniciada por um utilizador, que através da interface web, requer a reserva de uma aventura. Este pedido desencadeia uma interação com as aplicações remotas, iniciando-se pela reserva de uma atividade junto da aplicação ACTIVITY, da reserva de um quarto na aplicação HOTEL, da reserva de um veículo

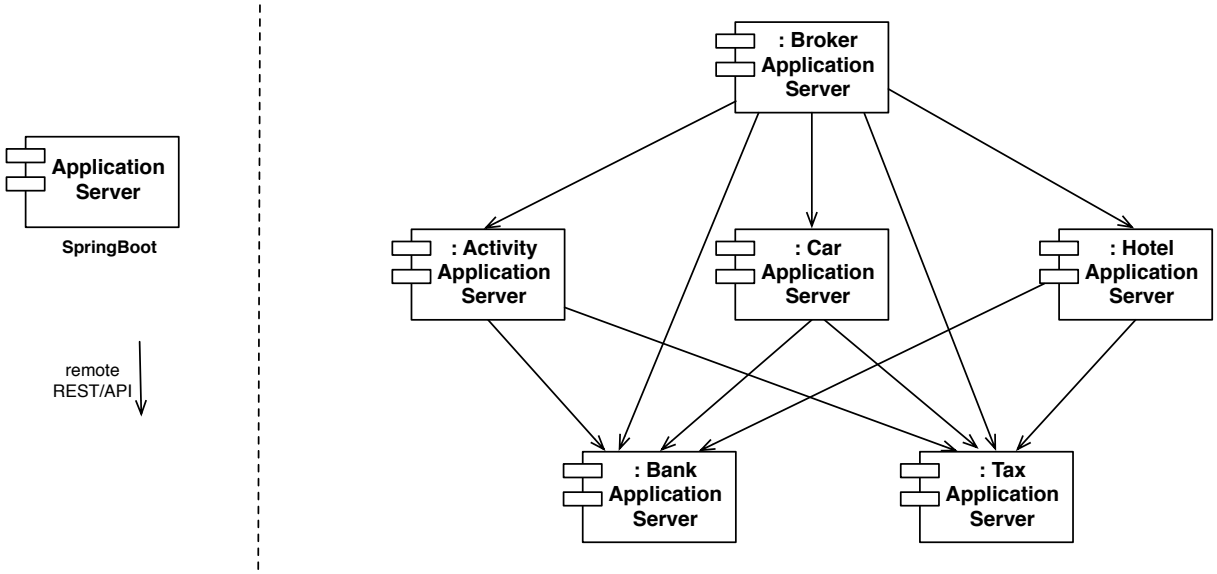


Figura 1: Modelo geral do sistema ADVENTURE BUILDER.

na aplicação CAR, sendo de seguida efetuado o pagamento na aplicação BANK e o pagamento de IVA na aplicação TAX. Quando esta sequência de interações ocorre com sucesso, a aventura termina no estado `CONFIRMED`, mas se acontecer algum erro termina no estado `CANCELLED`, após ter passado pelo estado `UNDO`. Embora a ordem pela qual os serviços são invocados seja a descrita, pode acontecer que uma aventura não necessite de todos os serviços, permitindo-se várias variações desta sequência. Por exemplo, considerando `PP=PROCESS_PAYMENT`, `RA=RESERVE_ACTIVITY`, `BR=BOOK_ROOM`, `RV=RENT_VEHICLE`, `UN=UNDO`, `CF=CONFIRMED`, `TP=TAX_PAYMENT` e `CC=CANCELLED`, `[RA,BR,RV,PP,TP,CF]` quando todos os serviços são pedidos e confirma, `[RA,BR,PP,TP,CF]` quando não é necessário um veículo e confirma, ou `[RA,BR,UN,CC]` quando não há quarto disponível e cancela. Esta interação está implementada através do padrão de desenho *State*, e o código da implementação está no *package* `broker.domain` da aplicação BROKER.

Uma outra interação que envolve mais do que uma aplicação ocorre entre as aplicações ACTIVITY, CAR e HOTEL e as aplicações BANK e TAX, sendo efetuado primeiro o pagamento no banco e depois o pagamento do IVA. Estas interações estão implementadas pelo padrão *Command* em que o processador de comando está implementado pelas classes `Processor` nos *packages* `domain` de cada uma destas três aplicações.

Por forma a se desenvolver um sistema com este nível de complexidade vamos usar uma abordagem incremental, em que o sistema irá ser desenvolvido através de uma sequência de versões, em que cada uma delas ou aumenta a quantidade total de funcionalidade existente na versão anterior ou altera-se o código para utilizar novas ferramentas ou aumentar a manutenção da solução, e iterativa, em que a complexidade tecnológica do sistema irá também ser introduzida por fases. Focando-nos nos aspetos iterativos do desenvolvimento do sistema devemos introduzir dois conceitos que nos ajudam na sua descrição e compreensão:

- *Módulo* - um módulo é uma unidade de implementação, como por exemplo uma classe ou um pacote
- *Componente* - um componente é uma unidade de execução, como por exemplo um pro-

cesso

Assim, o sistema de partida deste ano é constituído por 6 módulos de código em que cada um é compilado para gerar um componente, as 6 aplicações referidas. Assim, sempre que, por exemplo, nos referimos ao módulo `car` estamo-nos a referir ao código do componente CAR que é uma aplicação web.

Este ano, partindo dos 6 módulos com o código das 6 aplicações, iremos introduzir novas tecnologias (e.g., SPOCK FRAMEWORK, TRAVIS), bem como refatorizar o código para considerar novas funcionalidades e melhorar a sua legibilidade.

De acordo com o plano traçado, o desenvolvimento decorrerá em 4 fases distintas de sensivelmente 2 semanas e em que cada fase corresponde uma etapa do desenvolvimento iterativo. O desenvolvimento incremental acontecerá ortogonalmente e sempre que seja necessário enriquecer a aplicação com mais funcionalidade. De forma a garantir que todos os grupos acompanham o desenvolvimento, após a entrega associada a uma fase será disponibilizado o enunciado e o código de partida para a fase seguinte. Este código conterá a solução da fase anterior eventualmente enriquecido com mais funcionalidade.

Dado que em cada fase é entregue uma nova versão do código como ponto de partida para a fase seguinte, todas as alterações que se efetuem para além do solicitado no enunciado não estarão disponíveis na fase seguinte. Assim sendo, não é aconselhável a realização desse tipo de alterações, pois, para além de não terem impacto na avaliação, podem comprometer a conclusão das tarefas nos prazos definidos.

Resumidamente, para além do incremento de funcionalidade das aplicações, o foco de cada fase de desenvolvimento é:

1. Desenvolvimento baseado em testes: testes unitários com a tecnologia SPOCK FRAMEWORK.
2. Testes de integração com as funcionalidades de *mocks* da tecnologia SPOCK FRAMEWORK. Utilização do TRAVIS e CODECOV.
3. Introdução da persistência por refatorização do código e utilização da tecnologia MYSQL e FENIXFRAMEWORK.
4. Desenvolvimento de aplicações web com a tecnologia SPRINGBOOT, assim como testes de integração e de carga com a tecnologia JMETER.

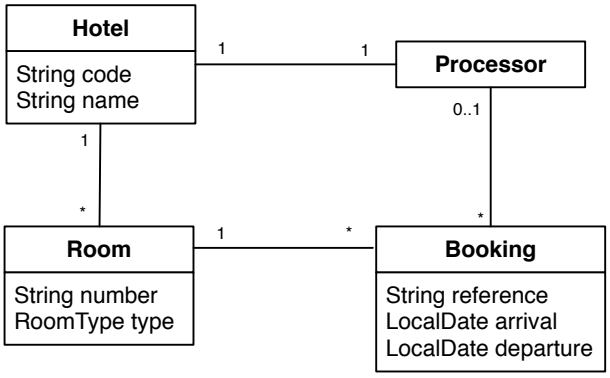
2. Módulos do Sistema

O sistema é constituído por 6 módulos que implementam a funcionalidade das aplicações: HOTEL, ACTIVITY, CAR, BROKER, BANK e TAX.

Sempre que for referida a existência de identificadores associados a entidades do domínio, quer dizer que eles são únicos, i.e., permitem identificar univocamente cada instância dessa entidade. Existem dois tipos de identificador, aqueles que são explicitamente dados como argumentos do construtor da instância, e os que são sequencialmente gerados durante a criação da instância. Sempre que for necessário visualizar uma lista de instâncias de entidade do domínio, por omissão, a ordem é alfabética crescente dos identificadores das entidades presentes na lista.

Note que nos seguintes modelos podem não estar representados todos os atributos das entidades.

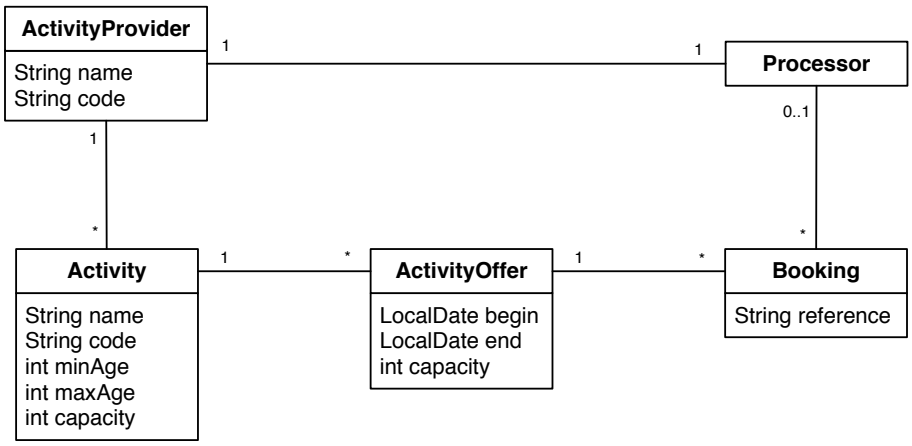
2.1. HOTEL



O módulo HOTEL é responsável por suportar a funcionalidade associada à reserva de quartos. Possui as seguintes entidades: `Hotel`, `Room`, `Booking` e `Processor`. O hotel é identificado por um identificador explícito, assim como o quarto, sendo no caso do quarto o número, ou seja, os quartos de um hotel são identificados pelo seu número. É possível ter várias instâncias de `Hotel` e cada hotel possui várias instâncias de `Room`. Os quartos podem ser singulares ou de casal.

A entidade `Booking` representa uma reserva de um quarto num determinado período e tem associada uma referência, que serve de identificação da reserva, e que é gerada sequencialmente. Naturalmente, não é possível ter a sobreposição de reservas de um quarto num mesmo período. Adicionalmente, a entidade `Booking` também implementa um *command*, possuindo por isso a entidade `Processor` uma lista das reservas a processar.

2.2. ACTIVITY

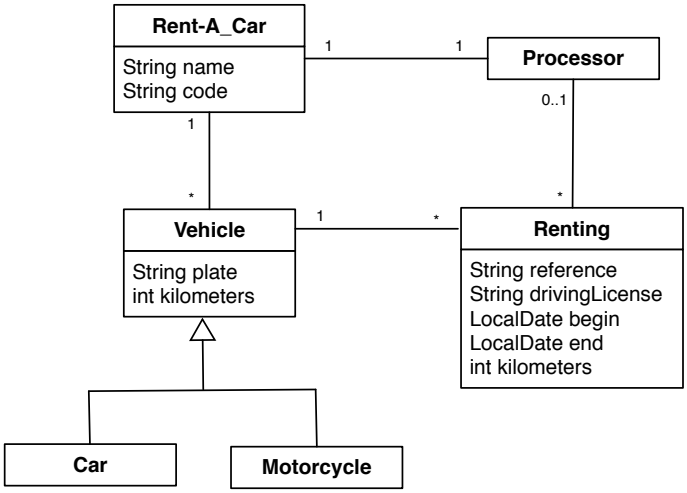


O módulo ACTIVITY é responsável por efetuar a gestão de atividades de lazer. Para isso contém as entidades `ActivityProvider`, `Activity`, `ActivityOffer`, `Booking` e `Processor`. Cada instância de `ActivityProvider` possui o conjunto das suas atividades, representadas pela entidade `Activity`, a qual descreve as idades aconselhadas dos participantes e o número limite de participantes. Uma entidade pode ser oferecida num certo intervalo de

tempo, entidade `ActivityOffer`, podendo ser oferecida várias vezes. As instâncias de `ActivityOffer` possuem associado o conjunto das suas reservas, entidade `Booking`. Tal como no módulo HOTEL a entidade `Processor` processa instâncias da entidade `Booking` como comandos.

A reserva de uma atividade tem que obedecer a um conjunto de condições, como seja, a data e a disponibilidade de vaga. Uma vez feita uma reserva é gerado um identificador, através de geração sequencial, que serve como referência da reserva.

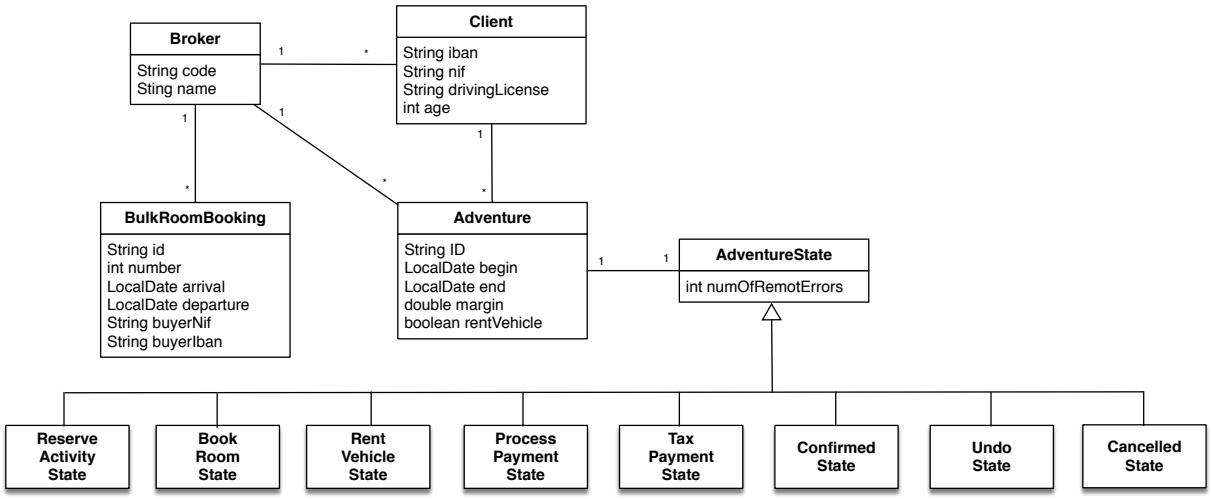
2.3. CAR



O módulo CAR é responsável por efetuar a gestão do aluguer de veículos, incluindo também o aluguer de veículos motorizados de 2 rodas. Para isso contém as entidades `Rent-A-Car`, `Vehicle` com 2 subclasses, `Car` e `Motorcycle`, `Renting`, e `Processor`. Cada instância de `Rent-A-Car` representa uma empresa de aluguer de veículos motorizados, representadas pela entidade `Vehicle`, a qual indica o número de quilómetros que o veículo possui. As instâncias de `Vehicle` possuem associado o conjunto dos seus alugueres, entidade `Renting`, as quais incluem o número de quilómetros que o veículo percorreu durante o aluguer. De igual forma aos dois módulos anteriores, a entidade `Processor` processa instâncias de `Renting`.

O aluguer de um veículo tem que obedecer a um conjunto de condições, como seja, a data e a disponibilidade do veículo. Uma vez efetuada uma reserva é gerado um identificador, através de geração sequencial, que serve como referência. Quando ocorre o fim do aluguer é calculado o número de quilómetros percorridos durante o aluguer e atualizada a quilometragem do veículo.

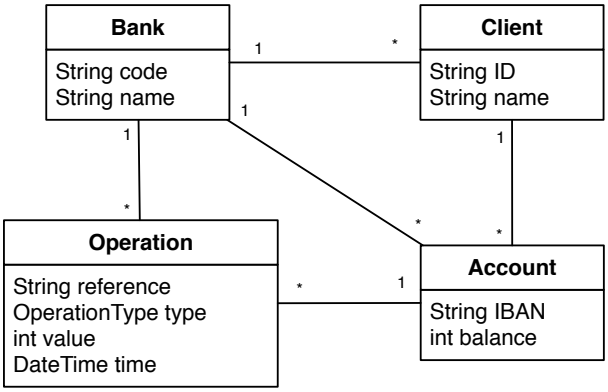
2.4. BROKER



O módulo BROKER é responsável pela gestão das aventuras integrando os serviços dos outros módulos. As suas entidades principais são: `Broker`, que representa uma empresa que fornece aos seus clientes pacotes integrados de atividades de lazer, `BulkRoomBooking`, que representa um conjunto de reservas em bloco de quartos efetuadas por um *broker*, `Client`, que representa o cliente de um *broker*, e `Adventure`, que representa o processo de marcação de uma atividade de lazer para um determinado cliente.

O processo de marcação de uma aventura é um processo complexo que ocorre em diversas fases, desde a sua criação inicial até aos pedidos de pagamento e reserva como descrito anteriormente. As subclasses the `AdventureState` representam cada um dos estados por que passa o processo de marcação.

2.5. BANK

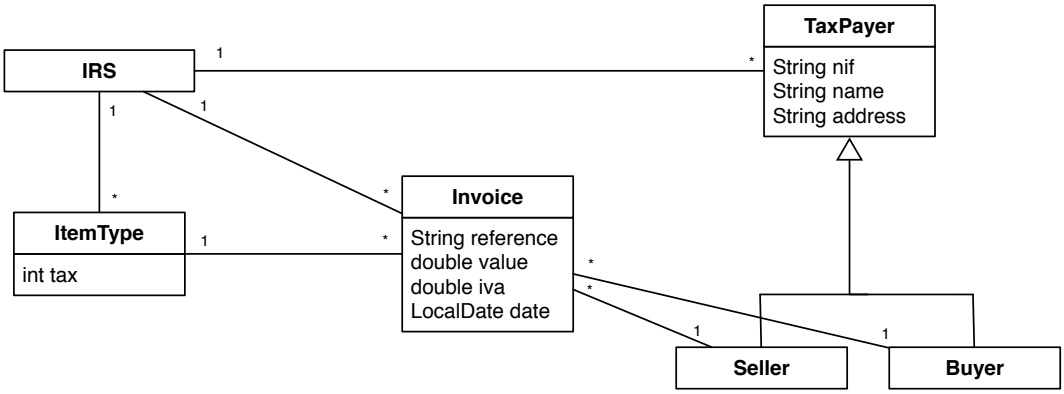


O módulo BANK é responsável por suportar a funcionalidade associada aos pagamentos das aventuras. Permite a criação de instâncias da entidade `Bank`, e cada banco possui associadas instâncias de `Client` e `Account`. O identificador de `Bank` é explícito enquanto que os identificadores de `Client` e `Account` são sequencialmente gerados.

Adicionalmente, sempre que ocorre uma operação de depósito, ou de levantamento, é criada uma instância da entidade `Operation` como forma de registar a sua ocorrência. Os identificadores das instâncias de `Operation`, que são gerados sequencialmente, servem como referência

da ocorrência da operação. Por exemplo, quando é efetuado o pagamento de uma aventura é enviada a referência da operação de pagamento para as entidades do módulo `Broker`. As instâncias de `Bank` possuem a lista de todas as operações que ocorreram no seu contexto.

2.6. TAX



O módulo TAX é responsável por calcular o IVA gerado em cada compra com dois objetivos: (1) tributar as transações comerciais e (2) calcular os benefícios fiscais que os compradores usufruem por solicitarem a emissão de faturas. Existe uma única instância da entidade `IRS` a qual tem associada todos os contribuintes, considerando-se, neste caso, dois tipos, disjuntos, de contribuintes, `Seller` e `Buyer`, os quais estão associados a uma transação comercial. Cada transação comercial corresponde à venda de um `Item`, que possui obrigatoriamente associada um comprador e um vendedor. O cálculo do valor de IVA de um `Item` é efetuado com base no seu valor e na taxa de IVA que está depende do tipo de item, `ItemType`. Anualmente são calculados os benefícios fiscais dos compradores e a tributação fiscal dos vendedores.

3. Avaliação do projeto

Semanalmente haverá uma avaliação da gestão do projeto. Cada aluno deve contribuir semanalmente com tarefas que impliquem o desenvolvimento de código, atualizar o repositório em conformidade e manter o planeamento semanal atualizado (trabalho realizado na semana anterior e trabalho previsto para a semana seguinte). Esta avaliação será feita durante a aula de laboratório de cada equipa. **A quantidade de trabalho deve ser divida equitativamente pelas semanas disponíveis para a sua realização.** Assim, a nota final penalizará os grupos, e cada aluno individualmente, que realizem a maior parte do seu trabalho apenas na última semana; no caso de todo o trabalho ser realizado na primeira semana não há penalização. Para fazer esta avaliação, o corpo docente irá usar a ferramenta *gitinspector*, que se encontra disponível em `https://github.com/ejwa/gitinspector`, pelo que aconselhamos a sua instalação. Grupos que não compareçam ao laboratório terão uma avaliação de 0 (zero) nesta componente. Trabalhadores estudantes estão isentos de comparecer nas aulas de avaliação mas devem fornecer, semanalmente, o mesmo material que os restantes colegas. A avaliação terá em conta a qualidade do planeamento do projeto e da gestão do projeto que está a ser aplicada.

O código produzido deve ser guardado no repositório do `github.com` atribuído a cada equipa.

Apenas a informação pedida deve ser impressa no terminal. Qualquer informação de depuração (*debug* ou *trace*) deve ser impressa no terminal através de uma instância da classe `Logger`

do **log4j2** (`http://logging.apache.org/log4j/2.x/`).

Todo o código e respetiva documentação, comentários no código, *issues* ou páginas *wiki* devem ser escritos em língua inglesa.

4. Fases de desenvolvimento

O projeto será realizado em equipas de 6 alunos de forma faseada. Cada equipa desenvolve o código no seu repositório `github.com`.

Os passos para se criar um *workspace* local do projeto do grupo são:

1. `git clone https://github.com/tecnico-softeng/es19CC_GG-project.git`, onde, em `es19CC_GG-project`, `CC` pode ser "al"ou "tg", e `GG` é o número do grupo
2. `git branch --all`, para verificar que existem *branches* `remotes/origin`

A fase final do projeto (após a quarta entrega) é avaliada com visualização e discussão a realizar no final do semestre. Após as discussões, será atribuído a cada aluno uma nota individual que procurará refletir a sua participação no projeto ao longo do semestre.

O projeto contribui com 30% da nota da disciplina e tem uma nota mínima final individual de 8 valores.

O corpo docente da disciplina de Engenharia de Software prevê que a realização do projeto exigirá cerca de 35 horas de trabalho a cada um dos membros da equipa. Nesta previsão, o corpo docente assume que os alunos já compreenderam o funcionamento das tecnologias utilizadas (JUNIT, JMOCKIT, FENIXFRAMEWORK, ...) antes da realização do trabalho e que existe um planeamento do projeto por forma a definir e compreender o trabalho a realizar, bem como a sua distribuição pelos vários elementos da equipa.

Serão publicados, para cada fase, pequenas adendas ao enunciado a detalhar o que será avaliado em cada semana e quaisquer alterações aos requisitos ao trabalho a desenvolver na fase seguinte.