Universidade do Minho

Licenciatura em Engenharia Informática

GitHub do Projeto

Desenvolvimento de Sistemas de Software- Entrega Intermédia 2

Grupo 24

Eduardo Pereira (A94881)
Gonçalo Freitas (A96136)
Gonçalo Vale (A96923)
José Martins (A97903)

Ano Letivo 2022/2023




Objetivos da Fase

Nesta fase o objetivo principal seria desenvolver mais três tipos de diagramas, que ajudariam na perceção e realização da última fase do trabalho. Começou-se, assim, por fazer um diagrama de comandos através de um ficheiro de Excel, que permitiu identificar os métodos mais importantes de cada subsistema tendo em conta as necessidades encontradas nas especificações dos use cases. Começamos assim por fazer um diagrama de componentes, seguido de diagramas de classes e, por fim, diagramas de sequência.

Trabalho realizado

Primeiramente o grupo começou por desenvolver o Excel, partindo dos use cases que teriam sido elaborados na etapa anterior. Através desses use cases, identificamos métodos que serão importantes nas etapas futuras, etapas estas que já envolvem trabalho de codificação, e através desses métodos identificamos subsistemas que nos irão facilitar algum trabalho nas etapas futuras, pois o código ficará mais acessível e compreensível para todos aqueles que integram a equipa de trabalho.

Posteriormente, desenvolvemos o diagrama de componentes, com o auxílio da ferramenta **Visual Paradigm**.

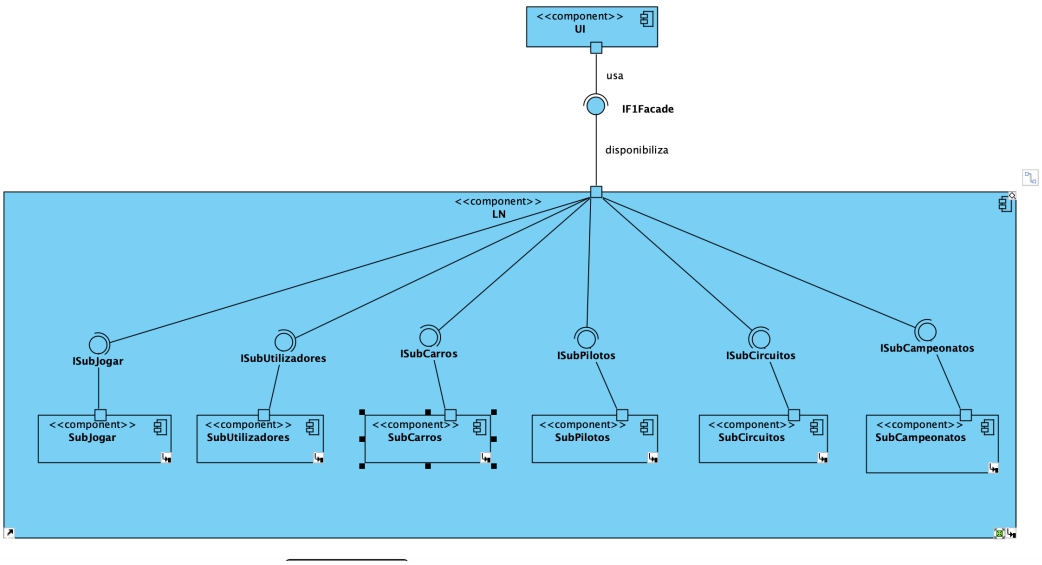

Figura 1: Diagrama de Componentes

O diagrama de componentes descreve os componentes do sistema e as dependências entre eles. Permite identificar, em cada nível, o que é necessário para construir o sistema. Um componente caracteriza-se por um pedaço de software reutilizável, encapsulado e que pode ser facilmente substituído. Todos os componentes juntos formam o sistema pretendido e possuem o comportamento definido pelas interfaces requeridas. Na Figura 1, a **UI** utiliza os métodos da interface **IF1Facade**, por sua vez a **LN** fornece os métodos da interface **IF1Facade**.

Findada esta distribuição, o atribuiu-se responsabilidades(métodos) a cada subsistema, apresenta-se de seguida um exemplo, de um diagrama de classes de um subsistema escolhido aleatoriamente.

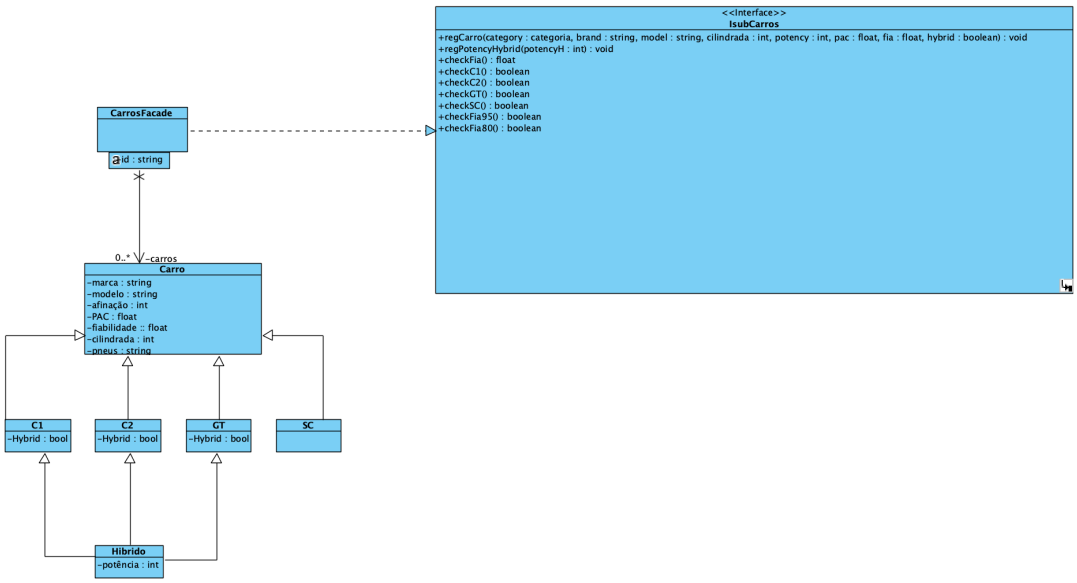

Figura 2: Diagrama de Classes - SubCarros

O subsistema subCarros é uma interface, que permitirá uma melhor relação entre subclasses, sendo que estas "se usam"em vários métodos, e algumas sendo atributos de outras. Cada uma destas tem a si associada um Facade, neste caso, a "CarrosFacade", seguindo por o diagrama abaixo, o facade tem a si associado um Map$< id : string, carro : Carro >$, que irá listar todos os carros dentro do nosso sistema, cada um com um id associado, este id será o seu modelo, que será único. A si associado está o carro em si, com os seus atributos, sendo que o tipo de carro é definido por classes também, pois estas também têm valores a si associados e mediante esses valores uma outra classe poderá ou não ser definida.

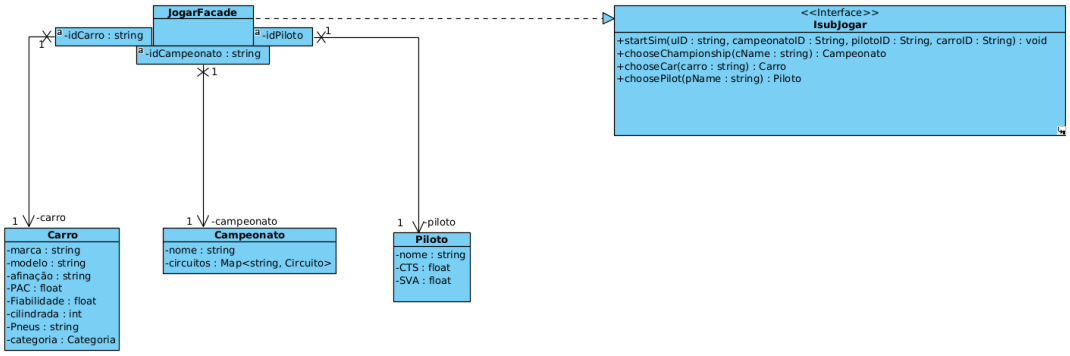

Figura 3: Diagrama de Classes - SubJogar

O subsistema subJogar é uma interface composta por vários métodos, que irá permitir uma melhor relação entre subclasses. Tem a si associado o Facade "JogarFacade". Como métodos este subsistema tem associado os métodos "choose"para piloto, carro e campeonato, que auxiliam na escolha dos argumentos do método "startSim".

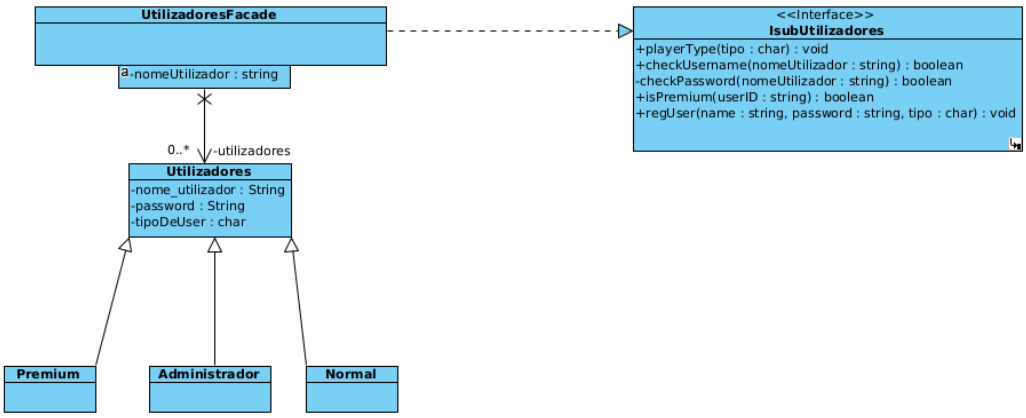

Figura 4: Diagrama de Classes - SubUtilizadores

O subsistema SubUtilizadores é uma interface, que permitirá uma melhor relação entre subclasses. Tem a si associado o Facade "UtilizadoresFacade", seguindo por o diagrama abaixo, o facade tem a si associado um Map$< nomeUtilizador : string, utilizador : Utilizador >$, que irá listar todos os utilizadores dentro do nosso sistema, cada um com um id associado, este id será o seu nome, que será único. O Utilizador em si tem como atributos o seu nome, password e tipo de User, o qual pode ser Premium, Administrador ou Normal.

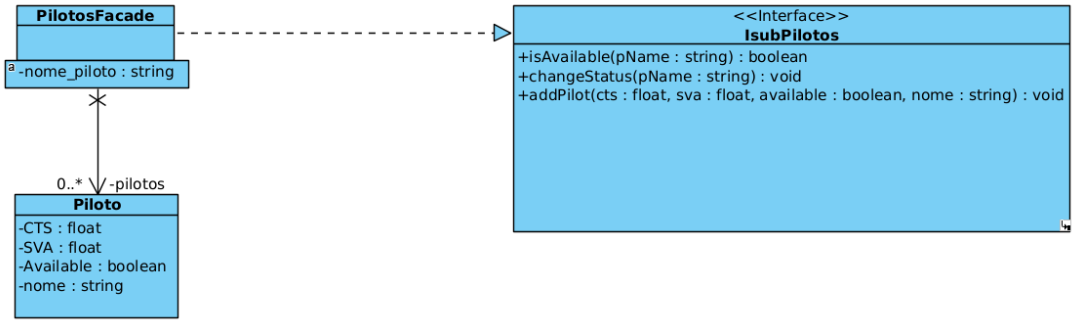

Figura 5: Diagrama de Classes - SubPilotos

O subsistema subPilotos é uma interface, que permitirá uma melhor relação entre subclasses. Este tem a si associado o facade "PilotosFacade", seguindo por o diagrama abaixo, o facade tem a si associado um Map$< id : string, piloto : Piloto >$, que irá listar todos os pilotos dentro do nosso sistema, cada um com um id associado, este id será o seu nome, que será único. A si associado está o piloto em si, com os seus atributos, sendo que o piloto é também definido pelo seu CTS e SVA, sendo que este também poderá estar disponível ou não.

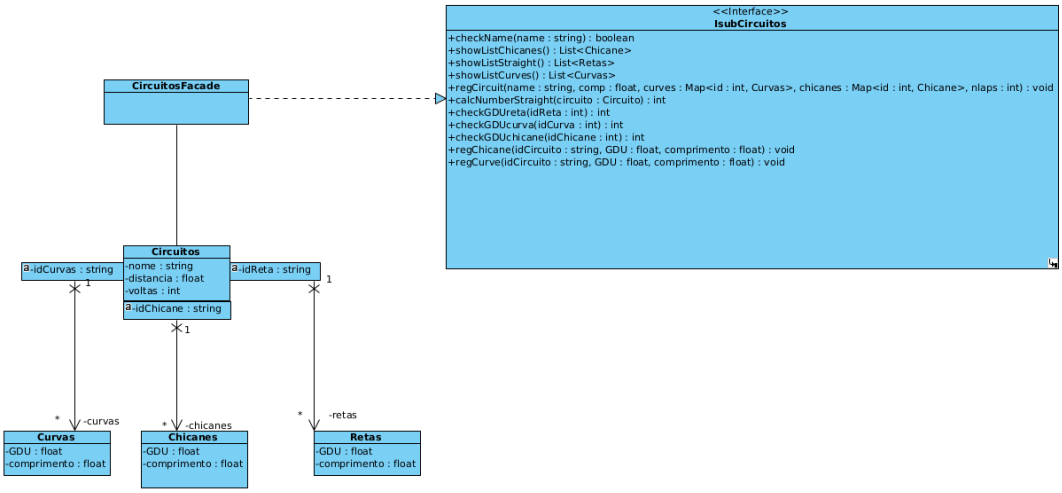

Figura 6: Diagrama de Classes - SubCircuitos

O subsistema subCircuitos é uma interface, que permitirá uma melhor relação entre subclasses. Este tem a si associado o facade "CircuitosFacade", seguindo por o diagrama abaixo, o facade tem a si associado uma coleção de curvas, chicanes e retas, que permitirão obter a forma da pista. Para além disso, o circuito tem a si associados três Maps, um para retas, um para curvas e um para chicanes, que associará um id ao respetivo GDU da entidade.

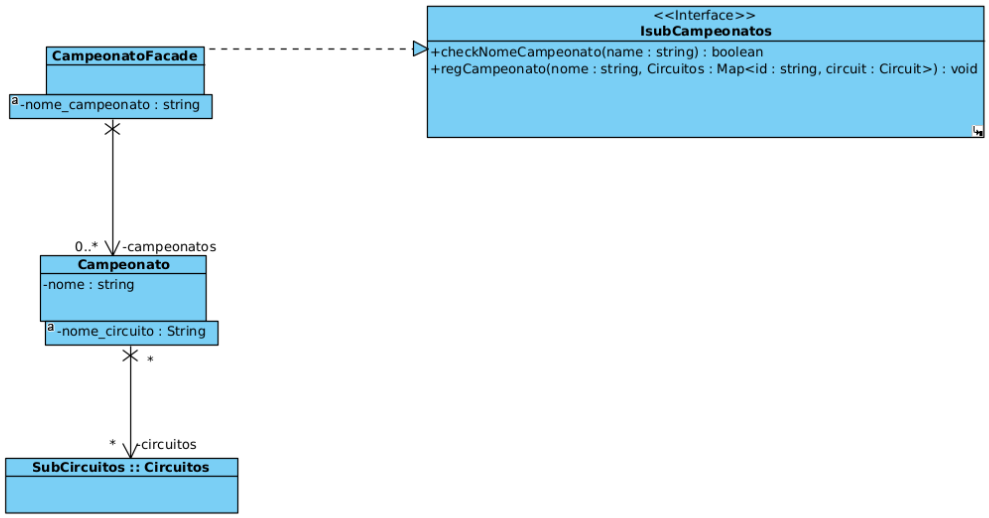

Figura 7: Diagrama de Classes - SubCampeonatos

O subsistema SubCampeonatos é uma interface, que permitirá uma melhor relação entre subclasses. Este tem a si associado o facade "CampeonatoFacade", seguindo por o diagrama abaixo, o facade tem a si associado o objeto Campeonato em si, o qual é identificado pelo seu nome, e qual está por si associado a um Map$< idCircuito : string, circuito : Circuito >$, que irá listar todos os circuitos disponíveis para serem utilizados, cada um com um id único.

Para cada método elaboramos um diagrama de sequência.

Estes diagramas são importantes pois representam as interações entre objetos através das mensagens, colocadas por ordem temporal, que são trocadas entre eles. Estes diagramas permitem também analisar a distribuição de responsabilidades pelas diferentes entidades, ou seja, analisar onde está a ser efetuado o processamento.

De seguida, apresentamos estereótipos de diagramas de sequência para cada "tipo"de método.

Em primeiro lugar, apresentamos como foi feito o diagrama de sequência de cada função de check, ou seja, verificar se certo elemento está contido no map de cada subsistema ou, no caso do subsistema dos carros, verificar o tipo de cada carro, fiabilidade. Para além disso, ao longo do projeto, utilizamos funções de registo para cada subsistema, tais funções que servem para adicionar novos elementos ás coleções respetivas, recebendo como argumentos os campos de cada classe, nomeadamente *regUser, regCarro, addPilot, regCircuit e regCampeonato*. Por fim, para o subsistema "subJogar"utilizamos uma função que inicia a simulação, que recebe como argumento um campeonato, um piloto e um carro, para além de funções que servem para escolher esses argumentos.

De seguida seguem exemplos de cada tipo de diagrama de sequência que utilizamos.

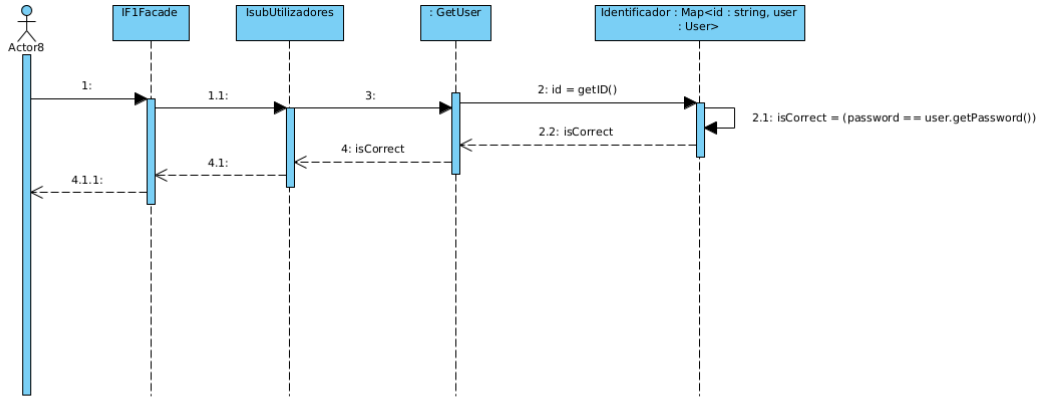

Figura 8: Diagrama de Sequência para funções "check- Exemplo

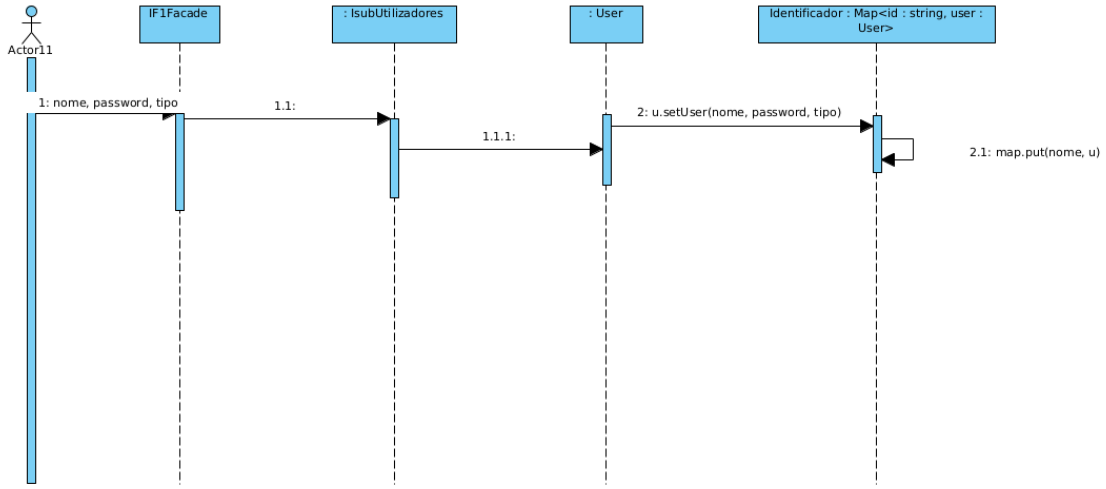

Figura 9: Diagrama de Sequência para funções "regX- Exemplo

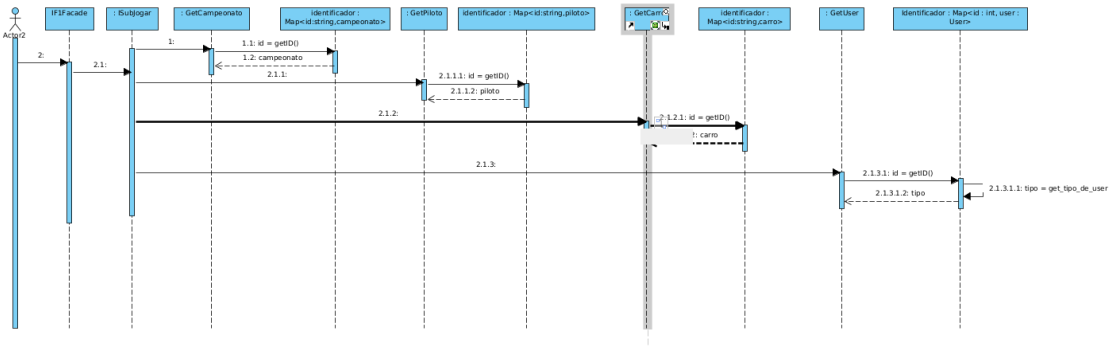

Figura 10: Diagrama de Sequência da função "startSim- Exemplo

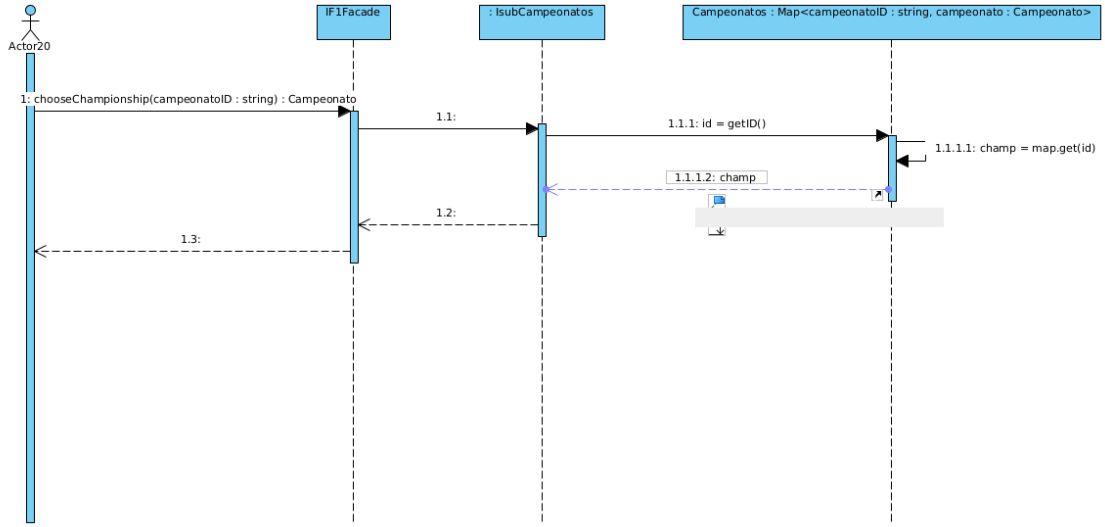

Figura 11: Diagrama de Sequência para funções "chooseX- Exemplo

Análise dos resultados obtidos

Ao longo da execução desta fase do projeto, o nosso grupo viu-se capaz de realizar todas as tarefas propostas para a correta realização deste. Os obstáculos com que nos deparámos nesta fase do trabalho foram, nomeadamente, a gerência de tempo, a conceção de certos diagramas de sequência e na decisão de certos aspetos da arquitetura da aplicação.

Findada esta fase do trabalho, podemos concluir que terminamos com um modelo de arquitetura do sistema sólido, que, a nosso ver, cumpre com os objetivos propostos pela equipa docente e que, acima de tudo, é capaz de suportar os desafios da próxima fase, a implementação.