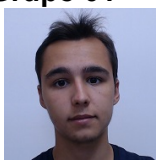
Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de
Desenvolvimento de Sistemas de Software

Ano Letivo de 2022/2023

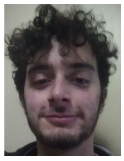
Sistema de Simulação de Campeonatos de Automobilismo
Fase 2

Grupo 01


a84696
Renato Gomes


a83630
Duarte Serrão


a89486
Tomás Dias


a97223
João Castro


a84372
Henrique Paz

URL do Repositório
https://github.com/joaocasr/DSS22-23-GP1

November 22, 2022

Índice

Lista de Figuras

1. Introdução

O presente relatório, da segunda fase do trabalho prático da unidade curricular de Desenvolvimento de Sistemas de Software, do curso de Licenciatura em Engenharia Informática da Universidade do Minho, visa demonstrar o desenvolvimento de quatro diagramas.

O primeiro será o diagrama de componentes, que descreve o layout das interações. O segundo o diagrama de classes, que descreve não só as iterações, mas também os atributos e operações de cada classe. O terceiro será o diagrama de packages que descreve a disposição por package das classes e por fim, o diagrama de sequência, que descreve o comportamento de cada uma das ações dos atores no sistema.

A metodologia aplicada ao longo desta fase foi reuniões frequentes com o grupo todo presente, dividindo tarefas de semana para semana. Também se comunicou bastante com os professores, o que resultou num melhor encaminhamento do projeto.

2. Alterações relativas à fase 1

Após uma análise mais detalhada ao trabalho efetuado na fase 1, detetámos alguns erros quer a nível de modelação de domínio, quer a nível de use cases.

Apesar de estar explícito nos use cases de que um jogador é identificado pelo seu username pensámos que seria conveniente adicionar essa informação no modelo de domínio. Por outro lado, optámos por inserir a entidade híbrido para as categorias específicas que podem ser híbridos (C1Hibrido,C2Hibrido,GTHibrido).Por outro lado, adicionámos uma entidade classificacaoCampeonato que representa a tabela classificativa dos jogadores para o campeonato em causa. Já a classificacaoCorrida representa a classificacão dos jogadores para uma determinada corrida num circuito.

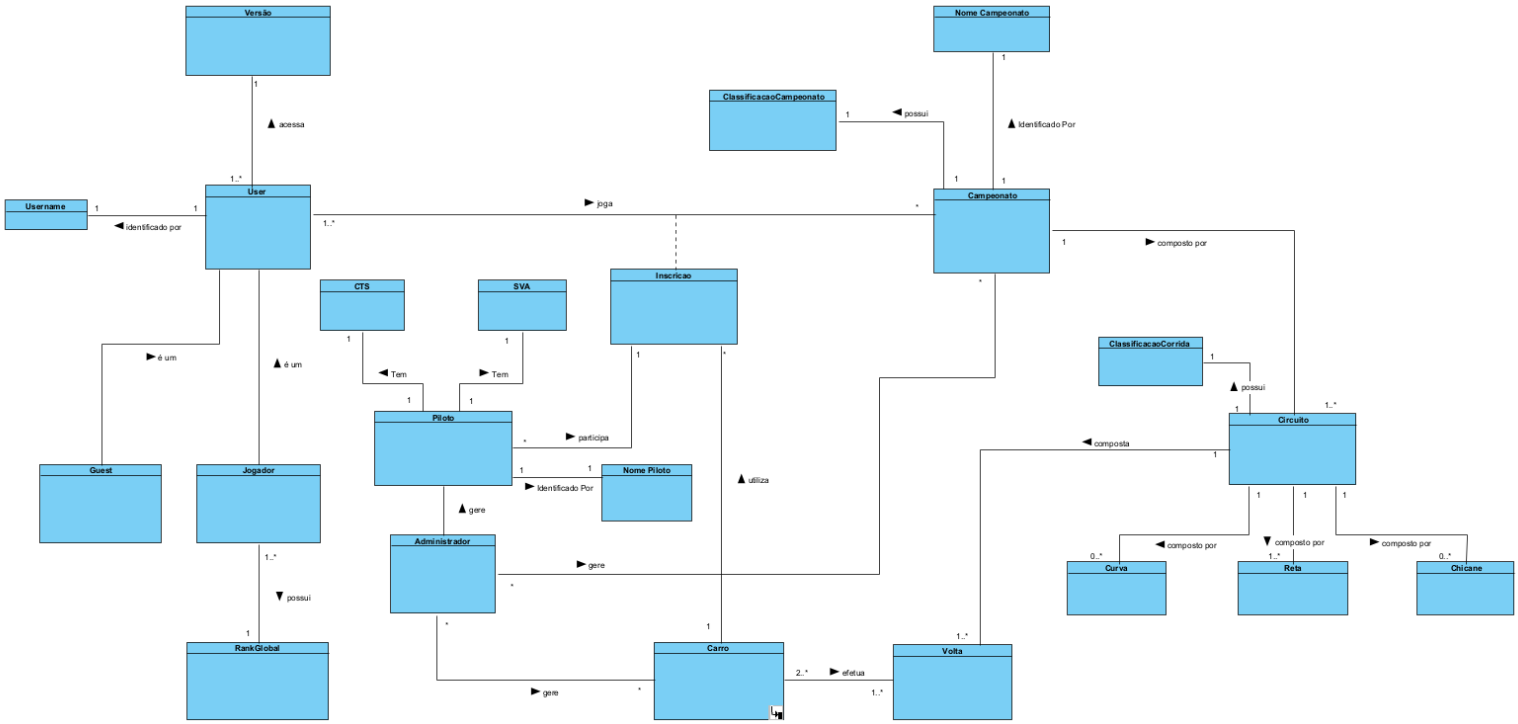


Figura 2.1: Modelo de Domínio simplificado

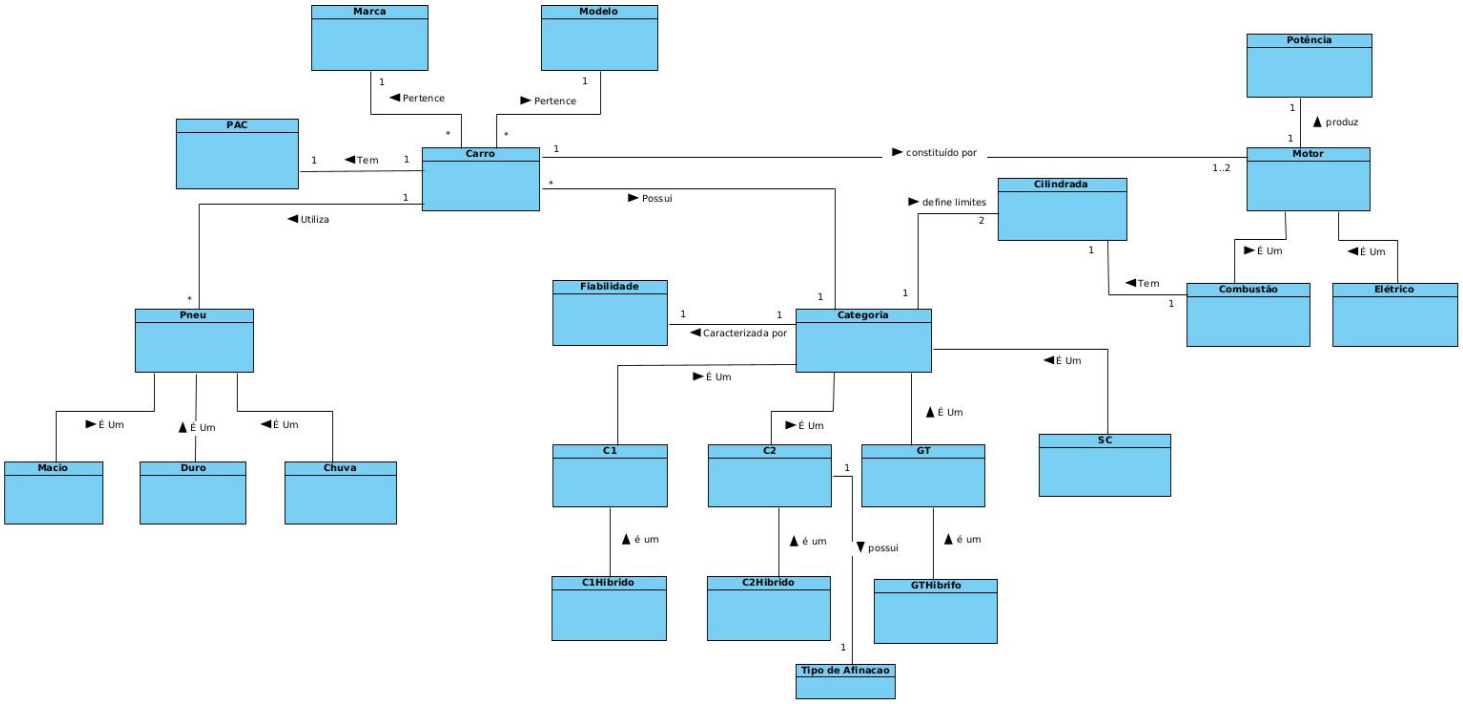


Figura 2.2: Modelo de Domínio centrado na entidade carro

Após algumas discussões detetámos algumas imprecisões no nosso diagrama de use cases:

1. Em primeiro lugar, em termos de sintaxe o Jogador não deverá herdar do Guest, ainda que este herde funcionalidades do mesmo. Desse modo, estaríamos a admitir que um Jogador é um Guest, o que é uma conclusão errada.
2. Relativamente à designação dos use cases, alguns encontram-se pouco explícitos, nomeadamente, o "Regista"e "Consulta Ranking". Deste modo, alterámos para "Registar Conta"e "Consultar Ranking",respetivamente.
3. Detetámos a ausência de um use case no diagrama ("Mudar versão").
4. Em termos de consistência pensámos que seria adequado termos uma funcionalidade de Logout já que possuímos uma funcionalidade de Login.

5. Dividimos o use case "Jogar"em "Inscrever no Jogo","Começar Jogo"e "Alterar configuração do carro", sendo que o use case "Criar Campeonato"agora possui um passo de introdução do número de participantes que vão jogar no campeonato em questão.
6. Adicionamos um novo use case em que consta a etapa de alteração das configurações do carro entre as corridas de um jogador.
7. O use case "Criar carro"foi alterado pois este encontrava-se mal construído.

Ainda relativamente aos Use Cases, decidímos realizar certas modificações, uma vez que, alguns destes use cases encontravam-se mal construídos ou tinham algumas imprecisões na sua constituição.

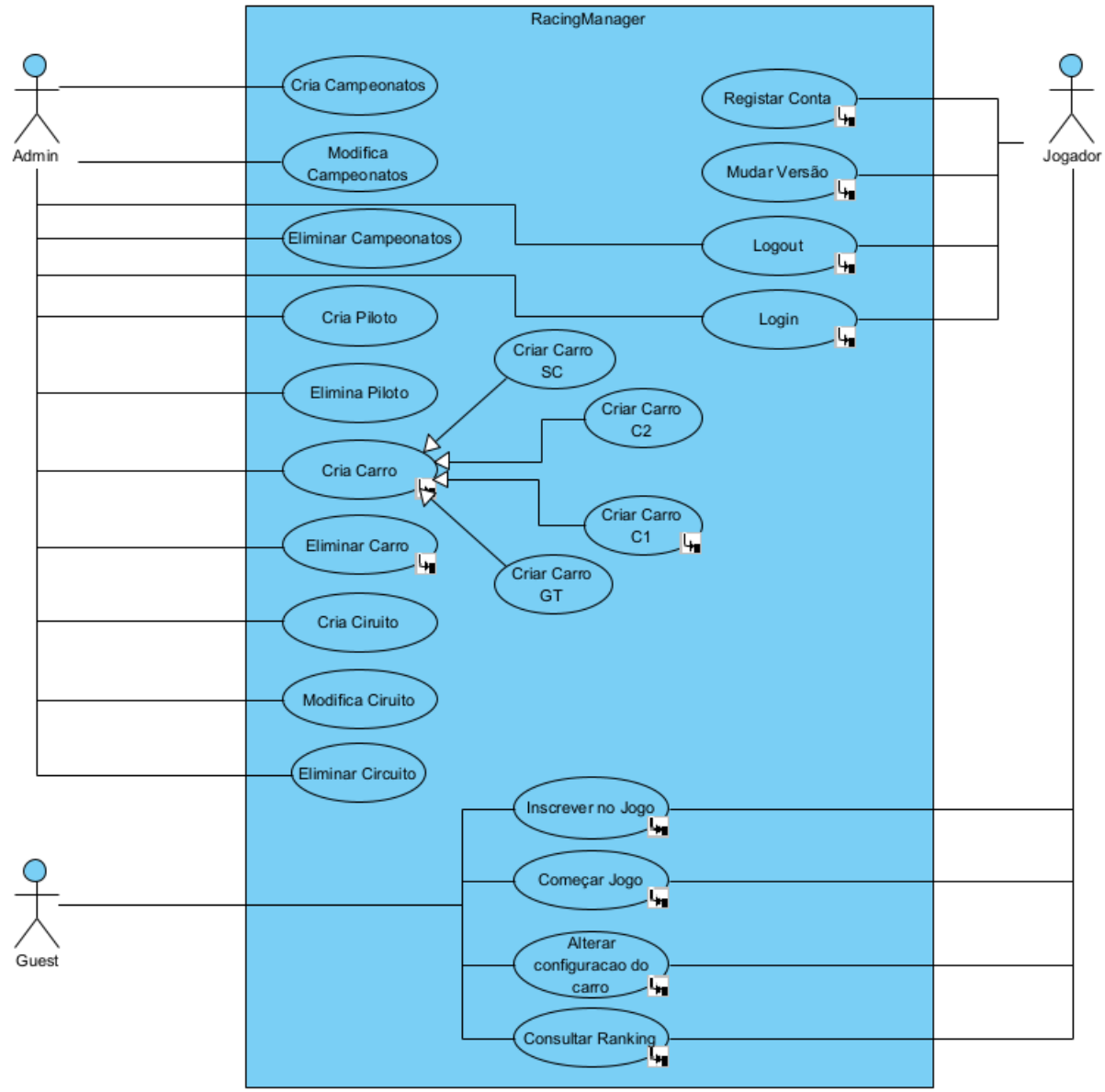


Figura 2.3: Diagrama de Use Case.

USE CASE: Consultar Ranking
Descrição: O Jogador consulta o ranking.
Cenário: O João consulta o ranking dos jogadores inclusive o seu.
Pré-Condição: True
Pós-Condição:O jogador consulta o Ranking.
Fluxo Normal:

1. O Jogador consulta o ranking geral.

Fluxo Alternativo(1): [O jogador consulta a sua posição] (passo 1)

3.1. O jogador consulta a sua posição na tabela de Ranking.

USE CASE: Mudar versão
Descrição: O Jogador altera a sua versão
Cenário: O Jogador decide alterar a sua versão do jogo.
Pré-Condição: Login feito como Jogador.
Pós-Condição: Versão do Jogador alterada.
Fluxo Normal:

1. O Jogador seleciona a opção de mudança de versão.
2. Sistema regista a mudança de versão.
3. O Jogador tem a sua versão alterada.

USE CASE: Registar conta
Descrição: O Jogador regista-se no sistema
Cenário: O João efetua registo no sistema, de modo a criar uma conta sua.
Pré-Condição:
Pós-Condição: O Jogador cria uma conta associada ao sistema.
Fluxo Normal:

1. Jogador fornece um username e uma password.
2. Sistema valida o username e avisa que a conta foi criada com sucesso.

Fluxo de Excepção(1): [O username já existe] (passo 2)

2.1. Sistema avisa que já existe uma conta corresponde a esse username.
2.2. Sistema cancela registo

USE CASE: Logout
Descrição: O jogador faz logout do sistema.
Cenário: O João efetua logout e sai da sua sessão.
Pré-Condição: O Ator está conectado ao sistema(efetuou login)
Pós-Condição: O Jogador termina a sua sessão.
Fluxo Normal:

1. O ator efetua logout do sistema.
2. Sistema termina sessão do ator.

USE CASE: Inscrever no Jogo.
Descrição:O jogador seleciona e inscreve-se num jogo.
Cenário:O João seleciona um campeonato para começar a jogar.
Pré-Condição: True
Pós-Condição: Inscrição efetuada num campeonato.
Fluxo Normal:

1. O sistema verifica o jogador atual.
2. O jogador seleciona o campeonato no qual irá jogar.
3. O jogador seleciona o carro.
4. O jogador seleciona o piloto.
5. O sistema guarda o campeonato, o piloto e o carro selecionados pelo jogador.
6. O sistema informa o jogador de que este está inscrito no campeonato.

USE CASE: Começar Jogo.
Descrição:O jogador inicia um jogo.
Cenário:O João começa um jogo.
Pré-Condição:O jogador tem de estar inscrito no campeonato.
Pós-Condição:O jogo é simulado.
Fluxo Normal:

1. O jogador informa o sistema de que pretende jogar.
2. O jogador informa o sistema o campeonato que pretende simular.
3. O sistema simula o jogo.

Fluxo Alternativo(1): [O sistema não simula o jogo.] (passo 3)

3.1. O sistema informa que a simulação não irá ocorrer pois ainda não existem jogadores suficientes inscritos no campeonato.

USE CASE: Alterar configuração do carro.
Descrição:O jogador pretende alterar as configurações do carro.
Cenário:O João decide alterar as configurações do seu carro.
Pré-Condição:O jogador não superou o numero máximo de alterações.
Pós-Condição: Configuração do carro é alterada.
Fluxo Normal:

1. O sistema verifica que o jogador ainda não efetuou um número de afinações superior a 2/3 do número de corridas do campeonato.
2. O sistema verifica que o carro do jogador é C2.
3. O jogador altera o tipo de afinação.
4. O jogador escolhe um downforce para o carro.
5. O jogador escolhe o tipo de pneus para o carro.
6. O jogador escolhe o modo do motor.
7. O sistema guarda as configuracões introduzidas pelo jogador.

Fluxo Alternativo(1): [O jogador já efetuou um número de alterações superior a 2/3 das corridas.] (passo 1)

1.1. O sistema informa o jogador que não pode fazer mais afinações ao seu carro.

Fluxo Alternativo(2): [O carro não é do tipo C2.] (passo 2)

2.1. Regressa a 4

USE CASE: Criar Carro C1
Descrição:O jogador cria um novo carro na categoria C1.
Cenário:O João quer criar um novo carro na categoria C1.
Pré-Condição: O login deve ser feito como Administrador.
Pós-Condição: O Administrador cria um novo carro c1 que fica guardado no sistema e pronto para ser usado pelos Jogadores.
Fluxo Normal:

1. O administrador atribui um código ao carro.
2. O sistema verifica que o código não existe no sistema.
3. O sistema pergunta qual a potência de combustão.
4. O administrador introduz a potência de combustão do carro.
5. O sistema pergunta qual a marca do carro.
6. O administrador introduz a marca do carro a adicionar.
7. O sistema pergunta o modelo da marca.
8. O administrador introduz o modelo da marca do carro a adicionar.
9. O sistema pergunta o PAC do carro.
10. O administrador introduz o valor do PAC.
11. O sistema valida o PAC.
12. O sistema pergunta se o carro é híbrido.
13. O administrador responde que não.
14. O carro C1 é registado no sistema

Fluxo Alternativo(1): [O carro é híbrido] (passo 13)

13.1. O carro a adicionar é híbrido.
13.2. O sistema pergunta qual a potência elétrica do carro.
13.3. O administrador atribui uma potência elétrica ao carro.
13.4. Regressa a 14

Fluxo de Exceção(1): [O código do carro já existe] (passo 2)

2.1. O sistema avisa que um carro com esse código já existe.

Fluxo de Exceção(2): [O valor do PAC não é válido] (passo 11)

11.1. O sistema avisa que o valor do PAC não é válido.

USE CASE: Criar Carro C2
Descrição:O jogador cria um novo carro na categoria C2.
Cenário:O João quer criar um novo carro na categoria C2.
Pré-Condição: O login deve ser feito como Administrador.
Pós-Condição: O Administrador cria um novo carro C2 que fica guardado no sistema e pronto para ser usado pelos Jogadores.
Fluxo Normal:

1. O administrador atribui um código ao carro.
2. O sistema verifica que o código não existe no sistema.

3. O sistema pergunta qual a potência de combustão.
4. O administrador introduz a potência de combustão do carro.
5. O sistema pergunta qual a marca do carro.
6. O administrador introduz a marca do carro a adicionar.
7. O sistema pergunta o modelo da marca.
8. O administrador introduz o modelo da marca do carro a adicionar.
9. O sistema pergunta o valor da cilindrada.
10. O administrador atribui uma valor para a cilindrada.
11. O sistema valida o valor da cilindrada.
12. O administrador fornece um tipo de afinação ao carro.
13. O sistema pergunta o PAC do carro.
14. O administrador introduz o valor do PAC.
15. O sistema valida o PAC.
16. O sistema pergunta se o carro é híbrido.
17. O administrador responde que não.
18. O carro C2 é registado no sistema

Fluxo Alternativo(1): [O carro é híbrido] (passo 17)

17.1. O carro a adicionar é híbrido.
17.2. O sistema pergunta qual a potência elétrica do carro.
17.3. O administrador atribui uma potência elétrica ao carro.
17.4. Regressa a 18.

Fluxo de Exceção(1): [O código do carro já existe] (passo 2)

2.1. O sistema avisa que um carro com esse código já existe.

Fluxo de Exceção(2): [Valor de cilindrada não corresponde à categoria escolhida ou o valor digitado está incorreto] (passo 11)

11.1. O sistema avisa que não é possível atribuir esse valor de cilindrada ao carro.

Fluxo de Exceção(3): [O valor do PAC não é válido] (passo 15)

15.1. O sistema avisa que o valor do PAC não é válido.

USE CASE: Criar Carro GT
Descrição:O jogador cria um novo carro na categoria GT.
Cenário:O João quer criar um novo carro na categoria GT.
Pré-Condição: O login deve ser feito como Administrador.
Pós-Condição: O Administrador cria um novo carro GT que fica guardado no sistema e pronto para ser usado pelos Jogadores.
Fluxo Normal:

1. O administrador atribui um código ao carro.
2. O sistema verifica que o código não existe no sistema.
3. O sistema pergunta qual a potência de combustão.
4. O administrador introduz a potência de combustão do carro.
5. O sistema pergunta qual a marca do carro.
6. O administrador introduz a marca do carro a adicionar.
7. O sistema pergunta o modelo da marca.
8. O administrador introduz o modelo da marca do carro a adicionar.

9. O sistema pergunta o valor da cilindrada.
10. O administrador atribui uma valor para a cilindrada.
11. O sistema valida o valor da cilindrada.
12. O sistema pergunta o PAC do carro.
13. O administrador introduz o valor do PAC.
14. O sistema valida o PAC.
15. O sistema pergunta se o carro é híbrido.
16. O administrador responde que não.
17. O carro GT é registado no sistema

Fluxo Alternativo(1): [O carro é híbrido] (passo 16)

16.1. O carro a adicionar é híbrido.
16.2. O sistema pergunta qual a potência elétrica do carro.
16.3. O administrador atribui uma potência elétrica ao carro.
16.4. Regressa a 17.

Fluxo de Exceção(1): [O código do carro já existe] (passo 2)

2.1. O sistema avisa que um carro com esse código já existe.

Fluxo de Exceção(2): [Valor de cilindrada não corresponde à categoria escolhida ou o valor digitado está incorreto] (passo 11)

11.1. O sistema avisa que não é possível atribuir esse valor de cilindrada ao carro.

Fluxo de Exceção(3): [O valor do PAC não é válido] (passo 14)

14.1. O sistema avisa que o valor do PAC não é válido.

USE CASE: Criar Carro SC
Descrição:O jogador cria um novo carro na categoria SC.
Cenário:O João quer criar um novo carro na categoria SC.
Pré-Condição: O login deve ser feito como Administrador.
Pós-Condição: O Administrador cria um novo carro SC que fica guardado no sistema e pronto para ser usado pelos Jogadores.
Fluxo Normal:

1. O administrador atribui um código ao carro.
2. O sistema verifica que o código não existe no sistema.
3. O sistema pergunta qual a potência de combustão.
4. O administrador introduz a potência de combustão do carro.
5. O sistema pergunta qual a marca do carro.
6. O administrador introduz a marca do carro a adicionar.
7. O sistema pergunta o modelo da marca.
8. O administrador introduz o modelo da marca do carro a adicionar.
9. O sistema pergunta o PAC do carro.
10. O administrador introduz o valor do PAC.
11. O sistema valida o PAC.
12. O carro SC é registado no sistema

Fluxo de Exceção(1): [O código do carro já existe] (passo 2)

2.1. O sistema avisa que um carro com esse código já existe.

Fluxo de Exceção(2): [O valor do PAC não é válido] (passo 11)

11.1. O sistema avisa que o valor do PAC não é válido.

USE CASE: Eliminar Carro

Descrição: O administrador elimina um carro do sistema.

Cenário: O João faz login no sistema como administrador. O João digita o carro que pretende eliminar e o sistema remove esse carro.

Pré-Condição: O login deve ser feito como Administrador.

Pós-Condição: O administrador seleciona o carro que pretende eliminar e o mesmo é removido do sistema.

Fluxo Normal:

1. O sistema pergunta qual o carro a eliminar.
2. O jogador escolhe o carro que pretende eliminar.
3. O sistema verifica que o carro existe e elimina o carro.

Fluxo de Exceção(1): [O carro não existe] (passo 3)

3.1. O sistema avisa que não é possível eliminar o carro.

USE CASE: Criar Campeonato

Descrição: Administrador cria um novo campeonato no sistema.

Cenário: Administrador entra no jogo e decide criar um novo campeonato, composto por diferentes circuitos.

Pré-Condição: Login feito como Administrador.

Pós-Condição: O novo campeonato será guardado no sistema.

Fluxo Normal:

1. Administrador digita o nome do campeonato.
2. Sistema valida o nome.
3. O administrador digita o número de participantes do campeonato.
4. Administrador escolhe que circuitos usar.
5. Sistema regista campeonato no sistema.

Fluxo de Excepção(1): [Nome do campeonato já existe] (passo 2)

2.1. Sistema avisa que esse campeonato já existe.

2.2. Sistema cancela registo do campeonato.

USE CASE: Modificar Circuito

Descrição: Administrador modifica um circuito já existente no sistema.

Cenário: Administrador entra no jogo e decide alterar o circuito ao 50 metros, adicionar 1 curva, modifcar o GDU da Curva 5 e adicionar 2 voltas.

Pré-Condição: Login feito como Administrador e decide modificar um circuito já esxistente.

Pós-Condição: As alterações do circuito são guardadas no sistema.

Fluxo Normal:

1. Administrador pede para modificar um circuito.

2. Sistema apresenta uma lista de circuitos.
3. Administrador seleciona o circuito a modificar.
4. Sistema apresenta a distancia e pergunta a nova distancia do circuito.
5. Administrador digita a nova distancia.
6. Sistema pergunta se quer adicionar ou retirar partes do percurso da circuito.
7. Administrador seleciona adicionar.
8. Sistema pergunta a quantidade de curvas que quer adicionar ao circuito.
9. Administrador digita a quantidade de curvas que quer adicionar ao circuito.
10. Sistema pergunta a quantidade de chicanes que quer adicionar ao circuito.
11. Administrador digita a quantidade de chicanes que quer adicionar ao circuito.
12. Sistema recalcula a quantidade de retas no circuito e apresenta as curvas, retas e chicanes numeradas.
13. Administrador seleciona qual as secções do circuito com GDU Difícil.
14. Sistema valida o GDU para as secções do circuito.
15. Administrador seleciona qual as secções do circuito com GDU Impossível.
16. Sistema valida o GDU para as secções do circuito.
17. Sistema atribui GDU Possivel para as restantes.
18. Sistema apresenta a quantidade de voltas e pergunta a nova quantidade de voltas do circuito.
19. Administrador digita a nova quantidade de voltas do circuito.
20. Sistema avisa que o circuito foi modificado.

Fluxo alternativo(1): [Administrador quer retirar partes do percurso do circuito] (passo 9)

7.1. Administrador seleciona a opção de retirar partes do percurso do circuito.
7.2. Sistema pergunta a quantidade de curvas que quer retirar ao circuito.
7.3. Administrador digita a quantidade de curvas que quer retirar ao circuito.
7.4. Sistema pergunta a quantidade de chicanes que quer retirar ao circuito.
7.5. Administrador digita a quantidade de chicanes que quer retirar ao circuito.
7.6. Sistema recalcula a quantidade de retas no circuito e apresenta as curvas, retas e chicanes numeradas.
7.7. Regressa a 13

Fluxo de Exceção(1): [O Administrador introduziu uma chicane mas não selecionou pare ter um GDU Difícil] (passo 14)

14.1. Sistema avisa que é necessário introduzir as chicanes para GDU Difícil.
14.2. Regressa a 13.

Os restantes use cases decidímos mantê-los inalteráveis, visto que no nosso ponto de vista, estes encontram-se bem construídos e consistentes.

3. API da Lógica De Negócios

USE CASE: Registar Conta

Tabela 3.1: Registar Conta

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1			
2	Verificar se username existe	existeUser(username:String):boolean	SubGestaoUsers
3	Registar conta	registaUser(username:String,password:String)	SubGestaoUsers

USE CASE: Login

Tabela 3.2: Login

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Faz input dos dados		
2	Verificar se username existe	existeUser(username:String):boolean	SubGestaoUsers
3	Validar password	efetuaLogin(username:String,password:String):boolean	SubGestaoUsers
4	Faz input dos dados		
2.1	Sistema avisa que já existe uma conta corresponde a esse username		
2.2	Sistema cancela login		
3.1	Sistema avisa que a password não é válida		
3.2	Sistema cancela login		

USE CASE: Criar Carro C1

Tabela 3.3: Criar C1

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Verificar se existe carro	existeCarro(idCarro:String):boolean	SubGestaoCP
2	Registar carro C1 hibrido	adicionarC1Hibrido(idCarro:String,marca:String,modelo:String,PotenciaCombustao:int,pac:int,PotenciaEletrica:int)	SubGestaoCP
3	Registar carro C1	adicionarC1(idCarro:String,marca:String,modelo:String,PotenciaCombustao:int,pac:int)	SubGestaoCP
4	Validar PAC	validaPac(pac:int):boolean	SubGestaoCP
5	Sistema verifica o utilizador atual	getCurrentUser():String	SubGestaoCP
6	Sistema verifica se o utilizador atual é administrador	getUser(username:String):User	SubGestaoCP

USE CASE: Criar Carro C2

Tabela 3.4: Criar C2

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Verificar se existe carro	existeCarro(idCarro:String):boolean	SubGestaoCP
2	Registar carro C2 híbrido	adicionarC2Hibrido(idCarro:String,marca:String,modelo:String,PotenciaCombustao:int,pac:int,PotenciaEletrica:int,tipoAfinacao:String,cilindrada:int)	SubGestaoCP
3	Registar carro C2	adicionarC2(idCarro:String,marca:String,modelo:String,PotenciaCombustao:int,tipoAfinacao:String,pac:int,cilindrada:int)	SubGestaoCP
4	Validar PAC	validaPac(pac:int):boolean	SubGestaoCP
5	Validar Cilindrada C2	validaCilindradaC2(cilindrada:int):boolean	SubGestaoCP
6	Sistema verifica o utilizador atual	getCurrentUser():String	SubGestaoCP
7	Sistema verifica se o utilizador atual é administrador	getUser(username:String):User	SubGestaoCP

USE CASE: Criar Carro GT

Tabela 3.5: Criar GT

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Verificar se existe carro	existeCarro(idCarro:String):boolean	SubGestaoCP
2	Registar carro GT híbrido	adicionarGTHibrido(idCarro:String,marca:String,modelo:String,PotenciaCombustao:int,pac:int,PotenciaEletrica:int,cilindrada:int)	SubGestaoCP
3	Registar carro GT	adicionarGT(idCarro:String,marca:String,modelo:String,PotenciaCombustao:int,pac:int,cilindrada:int)	SubGestaoCP
4	Validar PAC	validaPac(pac:int):boolean	SubGestaoCP
5	Validar Cilindrada GT	validaCilindradaGT(cilindrada:int):boolean	SubGestaoCP
6	Sistema verifica o utilizador atual	getCurrentUser():String	SubGestaoCP
7	Sistema verifica se o utilizador atual é administrador	getUser(username:String):User	SubGestaoCP

USE CASE: Criar Carro SC

Tabela 3.6: Criar SC

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Verificar se existe carro	existeCarro(idCarro:String):boolean	SubGestaoCP
2	Registar carro SC	adicionarSC(idCarro:String,marca:String,modelo:String,PotenciaCombustao:int,pac:int)	SubGestaoCP
4	Validar PAC	validaPac(pac:int):boolean	SubGestaoCP
5	Sistema verifica o utilizador atual	getCurrentUser():String	SubGestaoCP
6	Sistema verifica se o utilizador atual é administrador	getUser(username:String):User	SubGestaoCP

USE CASE: Eliminar Carro

Tabela 3.7: Eliminar Carro

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Verificar se existe carro	existeCarro(idCarro:String):boolean	SubGestaoCP
2	Eliminar carro do sistema	removerCarro(idCarro:String)	SubGestaoCP
3	Sistema verifica o utilizador atual	getCurrentUser():String	SubGestaoCP
4	Sistema verifica se o utilizador atual é administrador	getUser(username:String):User	SubGestaoCP

USE CASE: Consultar Ranking

Tabela 3.8: Consultar Ranking

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Seleciona opção		
2	Consultar Ranking Geral	getRanking():List<String>	SubGestaoUsers

USE CASE: Criar Piloto

Tabela 3.9: Criar Piloto

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Administrador introduz nome do Piloto		
2	Sistema valida o nome	validaNomePiloto(nome: String): boolean	SubGestaoCP
3	Administrador introduz níveis de CTS e SVA do Piloto		
4	Sistema valida os níveis de CTS e SVA	validarPericia(cts: Float, sva: Float): boolean	SubGestaoCP
5	Sistema regista o Piloto	registarPiloto(nomePiloto:String,SVA:int,CTS:int)	SubGestaoCP
6	Sistema verifica o utilizador atual	getCurrentUser():String	SubGestaoCP
7	Sistema verifica se o utilizador atual é administrador	getUser(username:String):User	SubGestaoCP

USE CASE: Eliminar Piloto

Tabela 3.10: Eliminar Piloto

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Administrador seleciona piloto que pretende eliminar		
2	Sistema elimina registo do piloto selecionado	removePiloto(nome: String)	SubGestaoCP
3	Sistema verifica o utilizador atual	getCurrentUser():String	SubGestaoCP
4	Sistema verifica se o utilizador atual é administrador	getUser(username:String):User	SubGestaoCP

USE CASE: Criar Campeonato

Tabela 3.11: Criar Campeonato

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1			
2	Validar se o nome é válido	validaNomeCampeonato(nomeCampeonato:String):Boolean	SubGestaoCC
3	Buscar todos os circuitos ao sistema	getAllCircuitos():List<Circuito>	SubGestaoCC
4	Registar Campeonato	guardaCampeonato(nomeCampeonato:String,njogadores:int)	SubGestaoCC
5	Sistema verifica o utilizador atual	getCurrentUser():String	SubGestaoCP
6	Sistema verifica se o utilizador atual é administrador	getUser(username:String):User	SubGestaoCP

USE CASE: Modificar Campeonato

Tabela 3.12: Modificar Campeonato

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Buscar a lista de campeonatos	getCampeonatos():List<Campeonato>	SubGestaoCC
2	Buscar os circuitos pertencentes a um campeonato	getCircuitosDoCampeonato(campNome:String):List<Circuitos>	SubGestaoCC
3			
4			
5			
6			
7	Substitui um circuito por outro no sistema	updateCircuitoCampeonato(nomeCamp:String, circNomeAntigo:String, circNomeNovo:String):Boolean	SubGestaoCC
		existeCircuitoemCampeonato(nomeCircuito:String):Boolean	SubGestaoCC
		existeCircuito(nomeCircuito:String):Boolean	SubGestaoCC
2.1			
2.2			
4.1			
4.2	Retirar circuito da lista de circuitos de um campeonato	apagaCircuitoDoCampeonato(nomeCampeonato:String, nomeCircuito:String):Boolean	SubGestaoCC
5	Sistema verifica o utilizador atual	getCurrentUser():String	SubGestaoCP
6	Sistema verifica se o utilizador atual é administrador	getUser(username:String):User	SubGestaoCP

USE CASE: Eliminar Campeonato

Tabela 3.13: Eliminar Campeonato

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Administrador seleciona campeonato que pretende eliminar		
2	Sistema elimina registo do campeonato selecionado	apagaCampeonato(campNome: String)	SubGestaoCC
3	Sistema verifica o utilizador atual	getCurrentUser():String	SubGestaoCP
4	Sistema verifica se o utilizador atual é administrador	getUser(username:String):User	SubGestaoCP

USE CASE: Consultar Campeonato

Tabela 3.14: Consultar Campeonato

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Utilizador escolhe um campeonato	getCampeonatos():List<Campeonato>	SubGestaoCC
		getCampeonato(nomeCampeonato:String):Campeonato	SubGestaoCC

USE CASE: Logout

Tabela 3.15: Logout

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1			
2	Fazer logout do sistema	logout()	SubGestaoUsers

USE CASE: Começar jogo

Tabela 3.16: Começar jogo

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1			
2	escolher o campeonato	getCampeonato(campeonato:String):Campeonato	SubGestaoCC
		getInscricoesCampeonato(campeonato:String):List<Inscricoes>	SubGestaoJogos

USE CASE: Inscrever no jogo

Tabela 3.17: Inscrever no jogo

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1			
2	O sistema verifica o jogador que quer jogar	getCurrentUser():String	SubGestaoUsersFacade
3		getUser(username:String):User	SubGestaoUsersFacade
4	O sistema apresenta os campeonatos disponíveis	getNomeCampeonatos():List<String>	SubGestaoCCFacade
5	O jogador seleciona o campeonato.	getCampenato(nomeCampeonato:String):Campeonato	SubGestaoCCFacade
6	O sistema apresenta os carros disponíveis	getCarros():List<String>	SubGestaoCPFacade
7	O jogador seleciona o carro.	getCarro(idCarro):Carro	SubGestaoCPFacade
8	O sistema apresenta os pilotos disponíveis.	getNomePilotos():List<String>	SubGestaoCPFacade
9	O jogador seleciona o piloto.	getPiloto(nomePiloto):Piloto	SubGestaoCPFacade
10	O sistema guarda a inscrição feita pelo jogador.	guardaEscolhasUser(user:User,campeonato:Campeonato,carro:Carro,piloto:Piloto)	SubGestaoJogos
11	O sistema verifica se já é possível simular o campeonato	validaNumeroInscricoes(campeonato:String):boolean	SubGestaoJogos

USE CASE: Criar Circuito

Tabela 3.18: Criar Circuito

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Administrador introduz nome do circuito		
2	Sistema valida o nome	existeCircuito(nomeCircuito:String):boolean	SubGestaoCC
3	Administrador introduz distância do circuito		
9	Sistema calcula a quantidade de retas	setPercurso(numCurva:int , numChicane:int):int	SubGestaoCC
10	Sistema apresenta a lista de curvas e retas	criaRetas():List<Reta>;criaCurvas():List<Curva>	SubGestaoCC
17	Registar Circuito	registarCircuito(nomeCircuito:String,distancia:int,voltas:int,numRetas:int,numChicanes:int,numCurvas:int,curvas:List<Curva>,retas:List<Reta>,chicanes:List<Chicane>)	SubGestaoCC
18	Sistema verifica o utilizador atual	getCurrentUser():String	SubGestaoCP
19	Sistema verifica se o utilizador atual é administrador	getUser(username:String):User	SubGestaoCP

USE CASE: Modificar Circuito

Tabela 3.19: Modificar Circuito

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Administrador requisita a lista dos circuitos		
2	Sistema apresenta a lista dos circuitos	getCircuitos():List<Circuito>	SubGestaoCC
3	Administrador seleciona o circuito		
4	Administrador digita o novo nome do circuito		
5	Administrador digita o número de retas,chicanes e curvas do circuito		
9	Sistema calcula a quantidade de retas	setPercurso(numCurva:int , numChicane:int):int	SubGestaoCC
10	Sistema apresenta a lista de curvas e retas	criaRetas():List<Reta>;criaCurvas():List<Curva>	SubGestaoCC
19	Administrador digita a nova quantidade de voltas		
20	Sistema modifica o circuito	modificaCircuito(antigoCircuito:String,circuito: Circuito)	
8	Sistema verifica o utilizador atual	getCurrentUser():String	SubGestaoCP
9	Sistema verifica se o utilizador atual é administrador	getUser(username:String):User	SubGestaoCP

USE CASE: Eliminar Circuito

Tabela 3.20: Eliminar Circuito

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Administrador seleciona o circuito que pretende eliminar		
2	Sistema elimina o circuito selecionado	removeCircuito(nomeCircuito: String)	SubGestaoCC
3	Sistema verifica o utilizador atual	getCurrentUser():String	SubGestaoCP
4	Sistema verifica se o utilizador atual é administrador	getUser(username:String):User	SubGestaoCP

USE CASE: Alterar configuração do carro

Tabela 3.21:

Nº	Identificar responsabilidades da LN	Definir API	Subsistemas
1	Adiciona configuracões em carros C2	adicionaConfiguracaoC2(username:String,Afinacao:String,downforce:int,tipo:String,String:modo)	SubGestaoConf
2	Adiciona configuracões	adicionaConfiguracao(username:String,downforce:int,tipo:String,String:modo)	SubGestaoJogos
3	Validar se o jogador já realizou um número de configurações superior a 2/3 do número de corridas do campeonato	validaConfiguracao(username:String):boolean	SubGestaoJogos
4	Verifica se o carro do jogador é de categoria C2	verificaInstanceC2(username:String):boolean	SubGestaoJogos

4. Diagrama de Componentes

Face a este problema de grande domínio, procedeu-se à construção do Diagrama de Componentes, de modo a agrupar este software em grupos de subsistemas menos complexos. Em cada um destes subsistemas vão se encontrar agrupados os vários métodos que definimos no capítulo de definição do API da LN. Por sua vez, estes subsistemas irão implementar uma interface, o que irá permitir ao programador invocar os respetivos métodos ou adicionar novos sem alterar a estrutura interna do subsistema, oferecendo desta forma uma maior organização a nível da arquitetura e uma melhor manutenção do código.

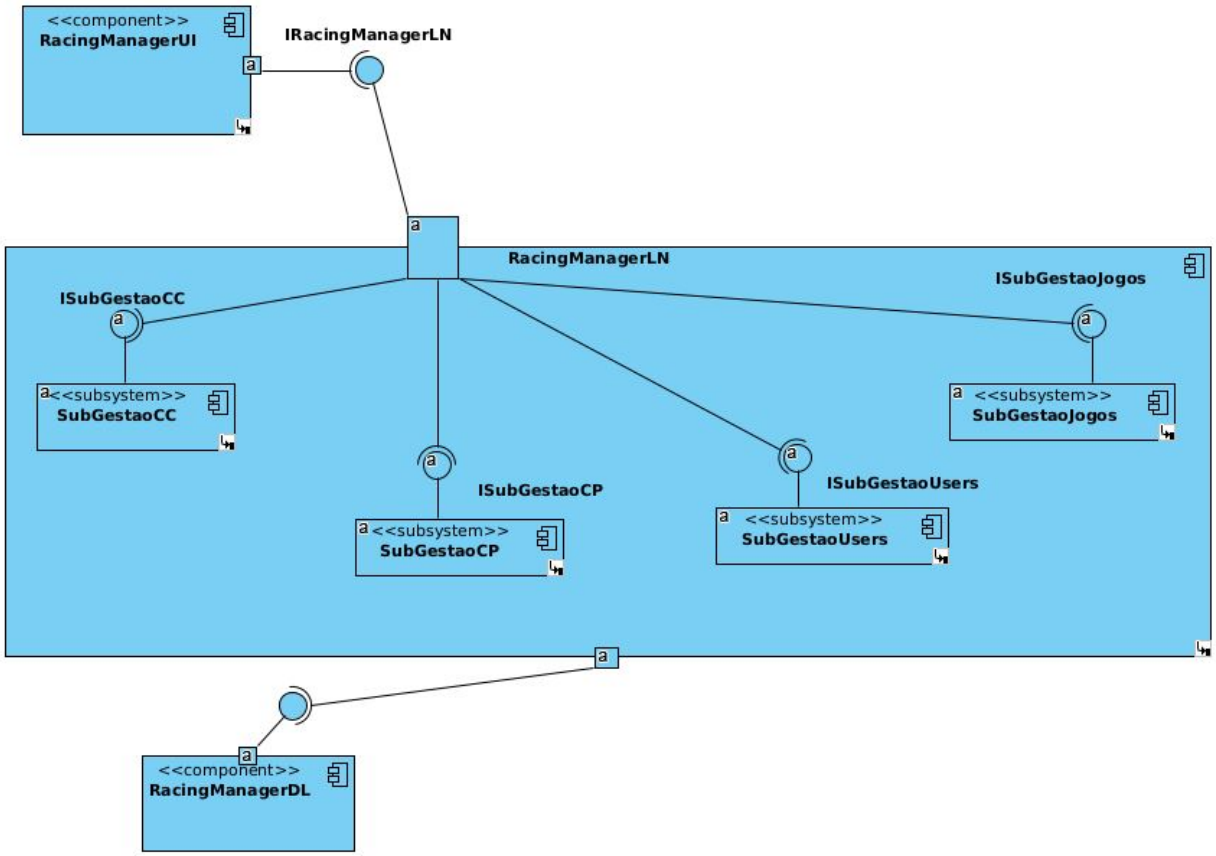


Figura 4.1: Diagrama de componentes.

Através do diagrama de componentes apresentado, podemos visualizar os vários subsistemas da nossa lógica de negócios. A nossa estratégia de partição do sistema consiste na existência de 5 subsistemas:
1- subsistema em que estão agrupados os carros e pilotos (SubGestaoCP)
2- subsistema em que estão agrupados os campeonatos e circuitos (SubGestaoCC)
3- subsistema dos vários utilizadores existentes (SubGestaoUser)
4- subsistema das inscrições dos jogadores para um dado campeonato. (SubGestaoJogos)

A nossa equipa de projeto optou por proceder a esta estratégia de divisão em subsistemas de forma a evitar que o sistema em geral tivesse uma dimensão muito grande, e por outro lado que cumprisse com os requisitos estabelecidos.

5. Diagramas de Classe

De seguida, encontram-se os vários diagramas de classe divididos pelos subsistemas da lógica de negócios. Através deste é possível averiguar como o sistema funciona, visualizar as interações/relacionamentos entre os componentes do sistema, e os métodos que vão ser implementados em cada um deles. Neste diagrama de classes arquiteturámos as várias classes e métodos que o sistema irá carecer para que tenha as funcionalidades desejáveis.

5.1 RacingManagerLN

Neste subsistema é onde se encontra a API global do sistema. Assim, o RacingManagerLN terá uma referência aos 4 subsistemas.

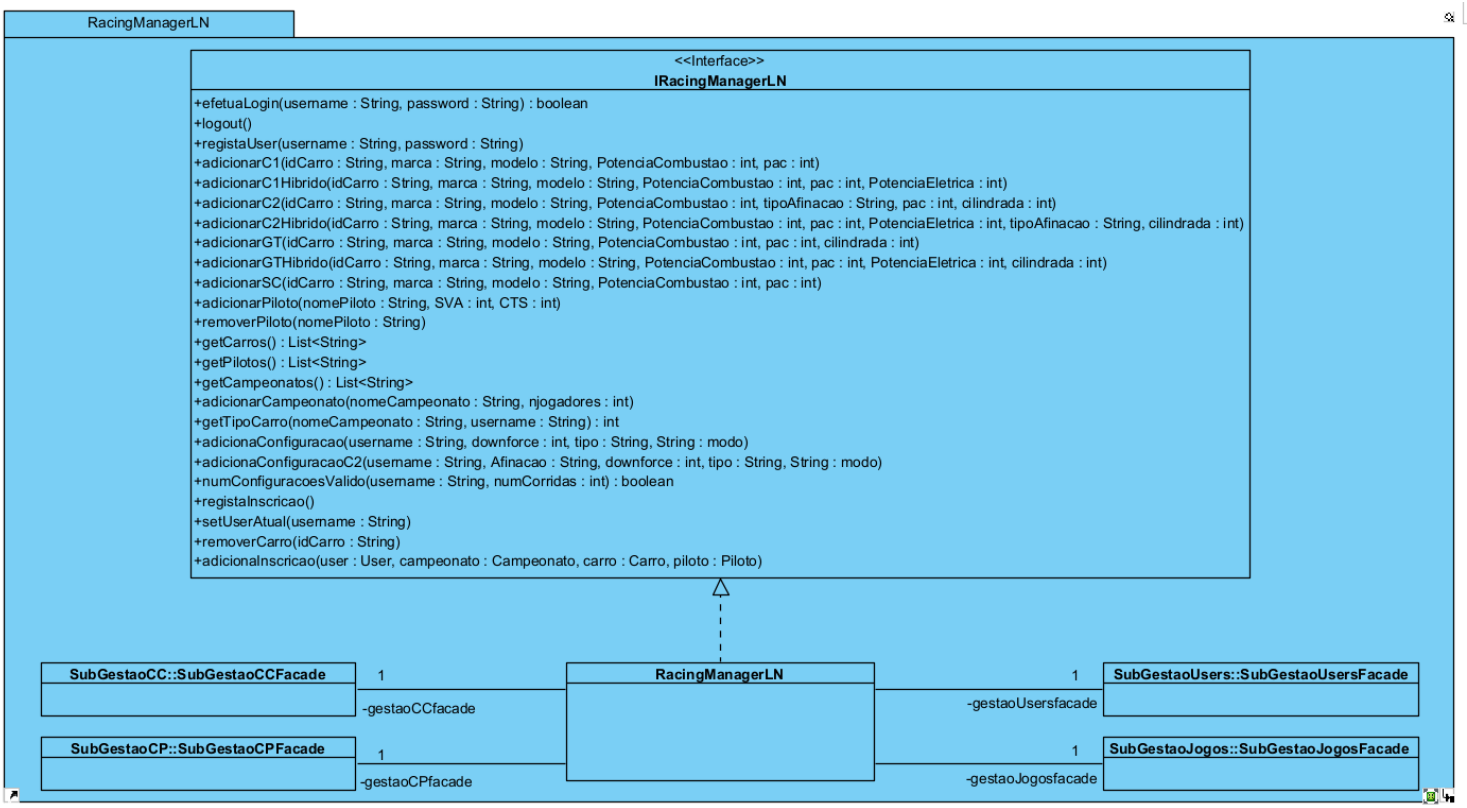


Figura 5.1: Diagrama de classes RacingManagerLN

5.2 SubGestao CC

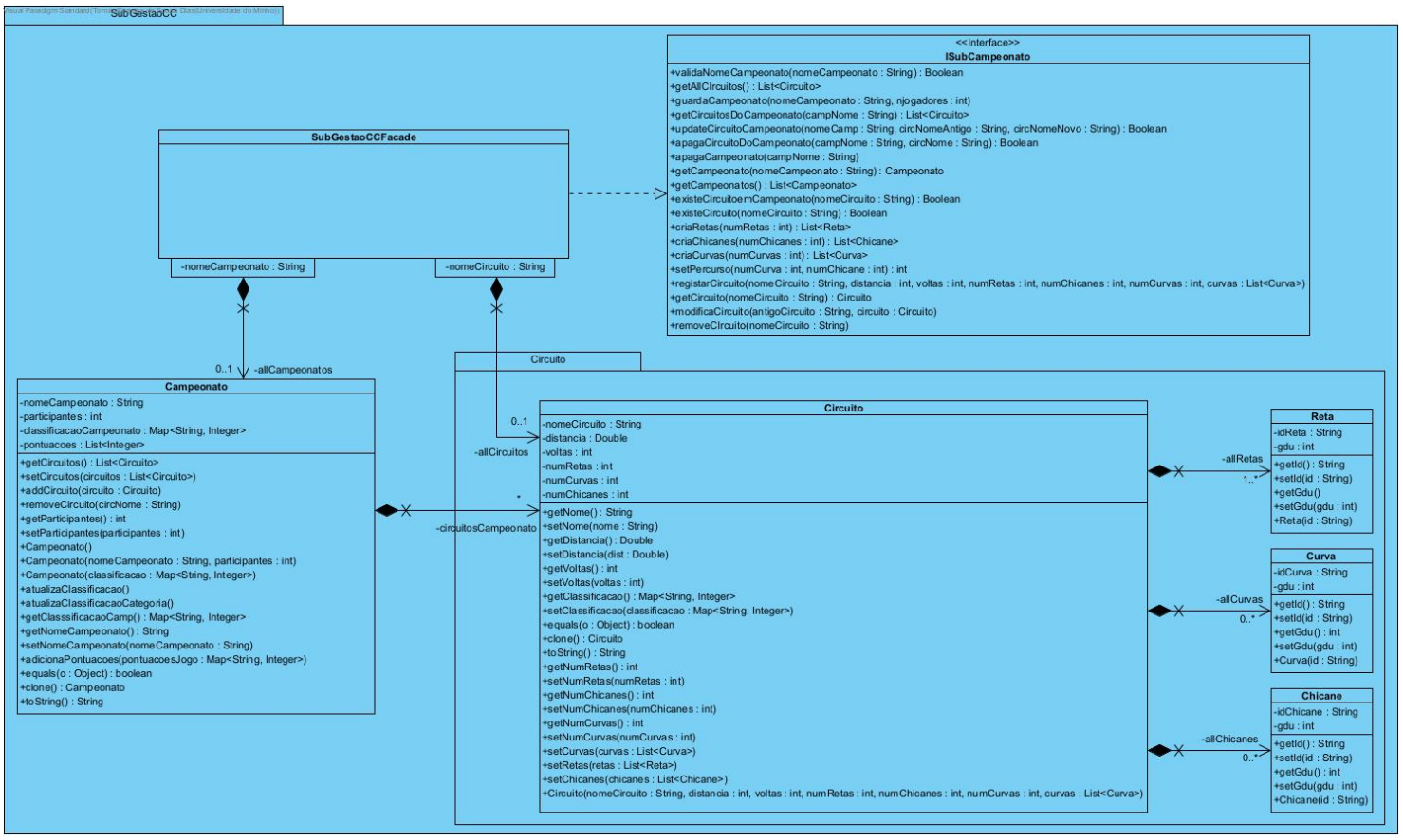


Figura 5.2: Diagrama de classes referente ao subsistema Campeonato-Circuito

5.3 SubGestao CP

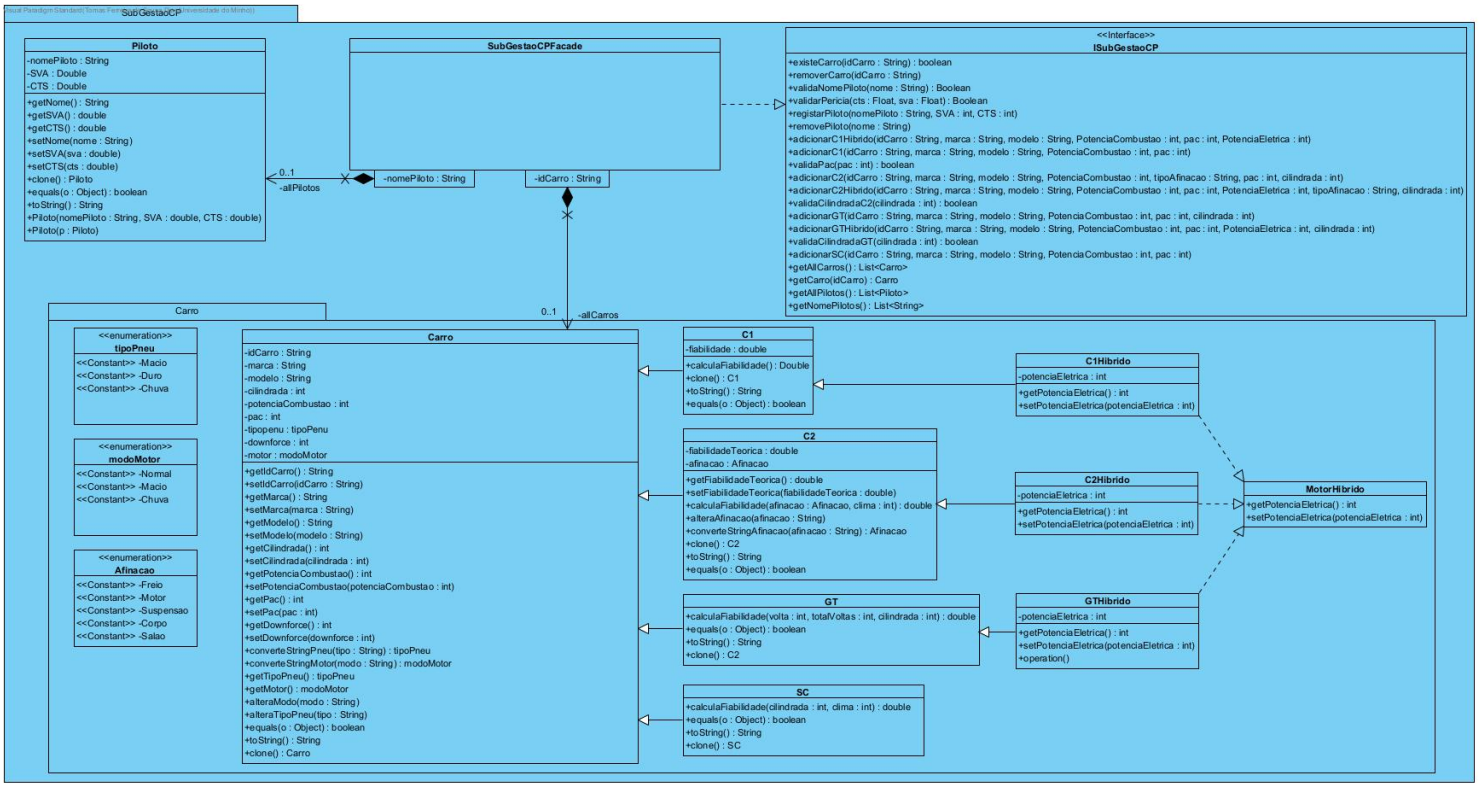


Figura 5.3: Diagrama de classes referente ao subsistema Campeonato-Piloto

5.4 SubGestao Jogos

A classe Inscrição irá possuir uma relação de associação ao User, Carro, Piloto e Campeonato. Cada inscrição corresponde às escolhas feitas por um dos jogadores ao iniciar o jogo. Cada campeonato irá possuir uma lista de inscrições das quais a Simulação irá necessitar para simular o campeonato em causa. Desta forma, a SubGestaoJogosFacade irá possuir a estrutura de dados seguinte: $Map < String, List < Inscricao >>$, em que a string corresponde ao nome do campeonato.

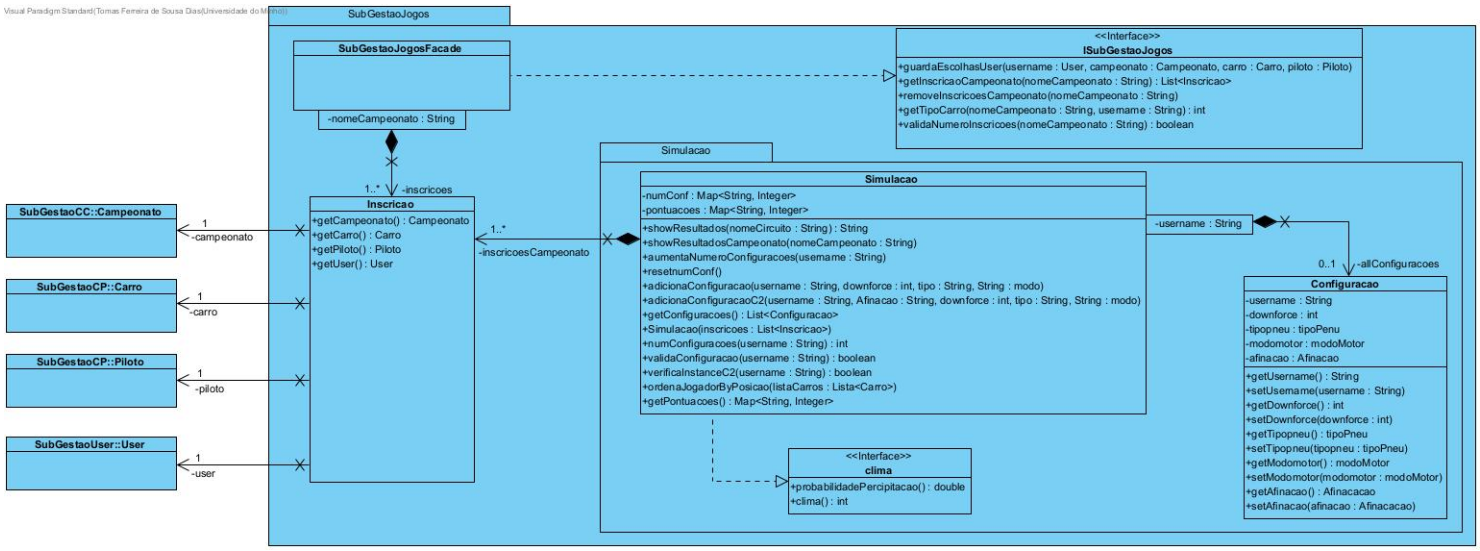


Figura 5.4: Diagrama de classes referente ao subsistema de Gestão de Jogos

5.5 SubGestao Users

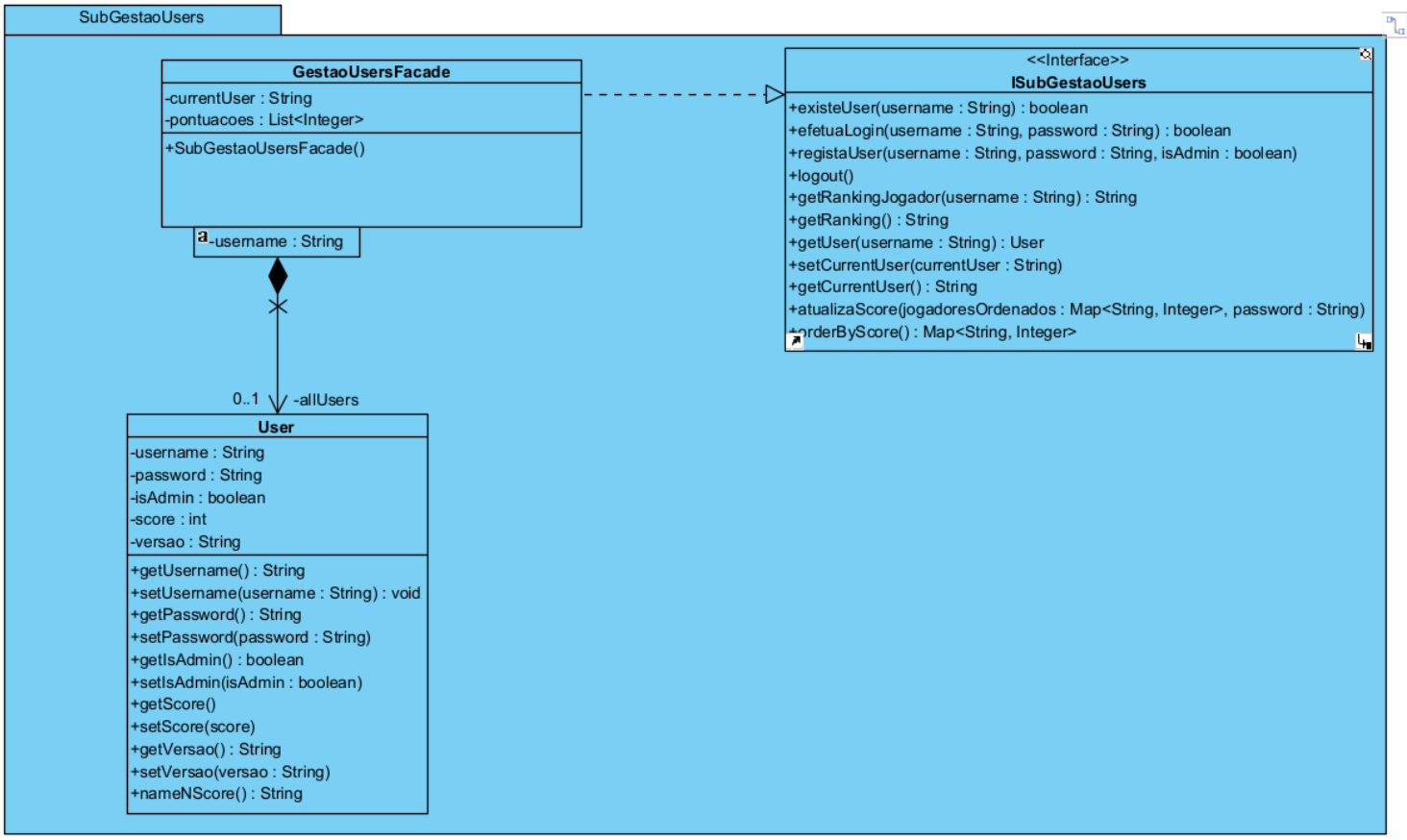


Figura 5.5: Diagrama de classes referente ao subsistema de Gestão de Users

6. Diagramas de Sequência

Através da conceção dos seguintes diagramas de sequência será possível estabelecer desde já as interações entre objetos e métodos que o nosso sistema irá possuir. Desta forma, quando procedermos para uma fase de implementação os vários desenvolvedores poderão analisar estes diagramas e saber de que forma as várias partes do sistema interagem umas com as outras, assim como outras informações relevantes como o tempo de vida dos objetos e a ordem pela qual os métodos deverão ser executados de forma a concretizar uma determinada funcionalidade.

6.1 Consultar Ranking

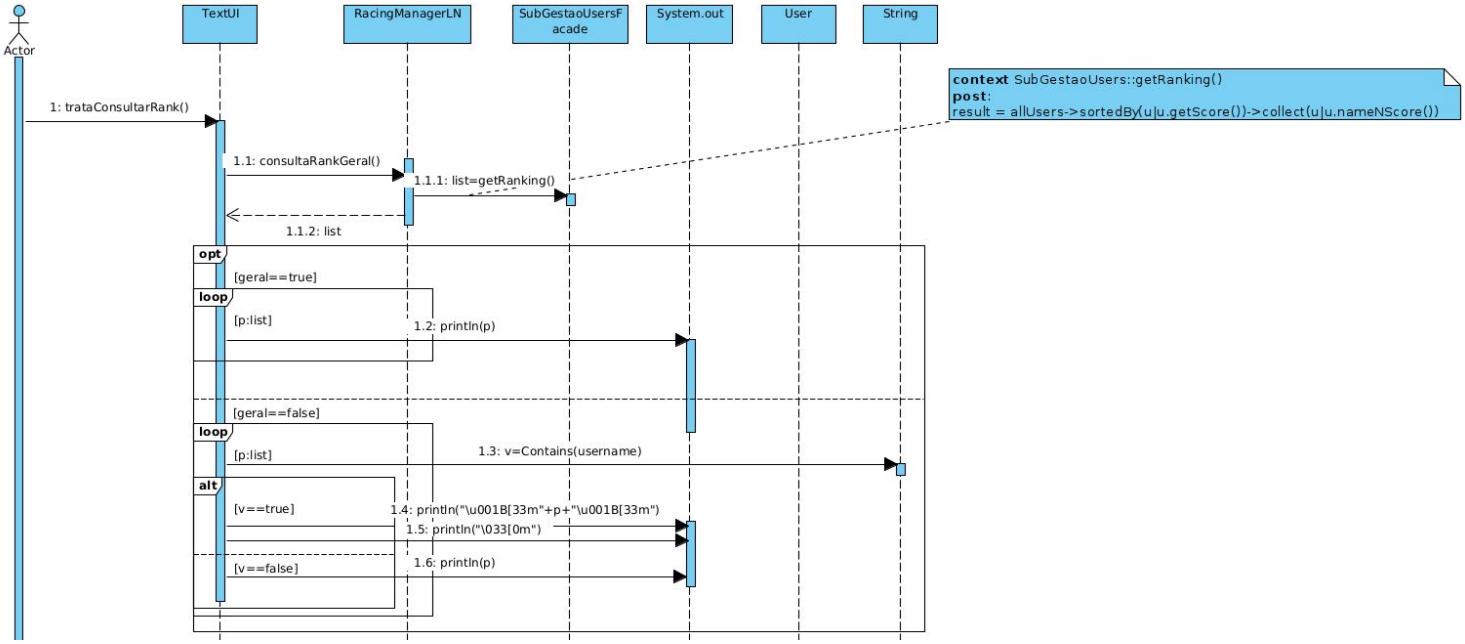


Figura 6.1: Diagrama de sequência referente à consulta do ranking geral

6.2 Mudar Versão

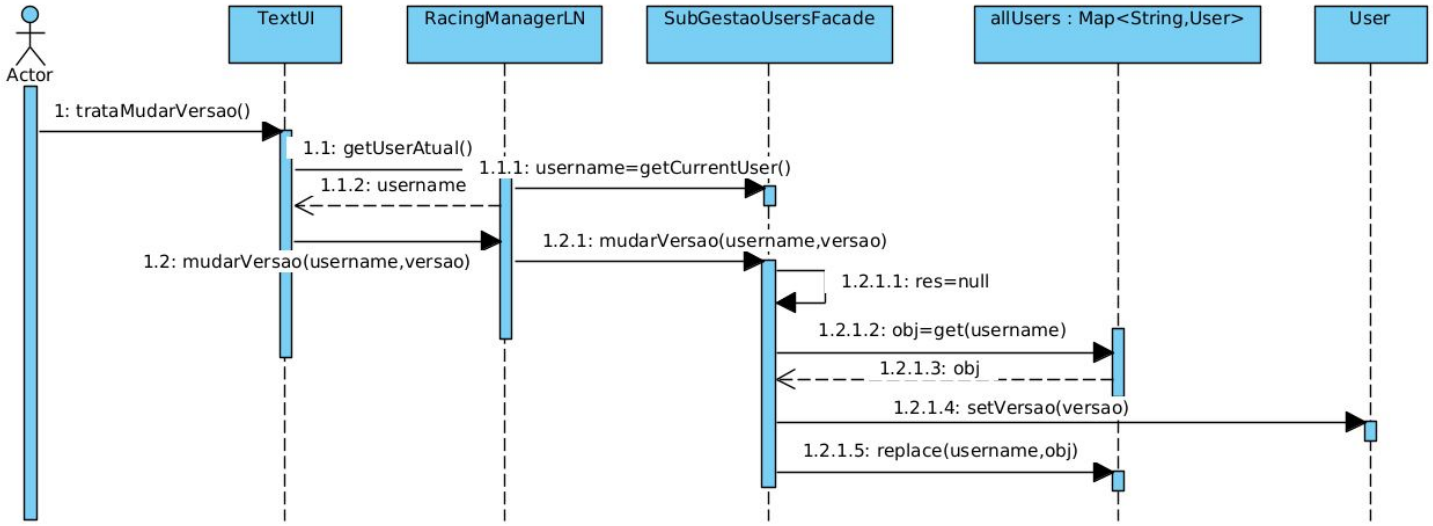


Figura 6.2: Diagrama de sequência referente à mudança de versão

6.3 Inscrever no jogo- Campeonatos disponíveis

A apresentação dos campeonatos, dos carros e dos pilotos do sistema possuem a mesma lógica de representação no diagrama de sequências, pelo que apenas vamos apresentar a dos campeonatos.

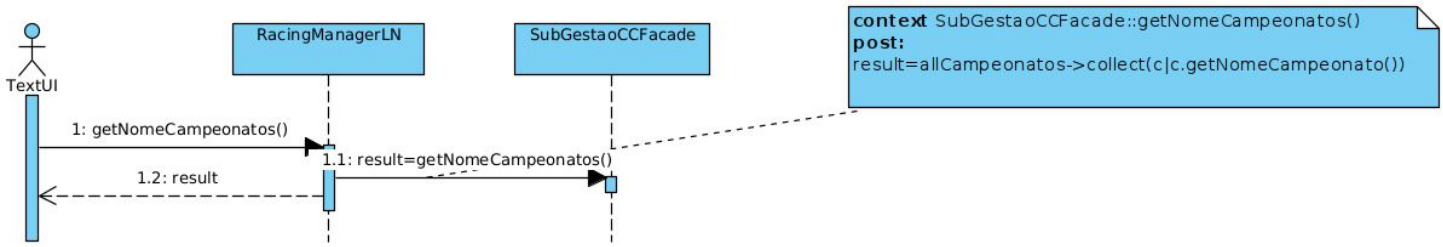


Figura 6.3: Diagrama de sequência referente à apresentação dos campeonatos disponíveis.

6.4 Inscrever no jogo- Selecionar Campeonato

A escolha do campeonato, do carro e do piloto apresentam a mesma lógica de representação no diagrama de sequências, pelo que apenas vamos apresentar a escolha de um campeonato.

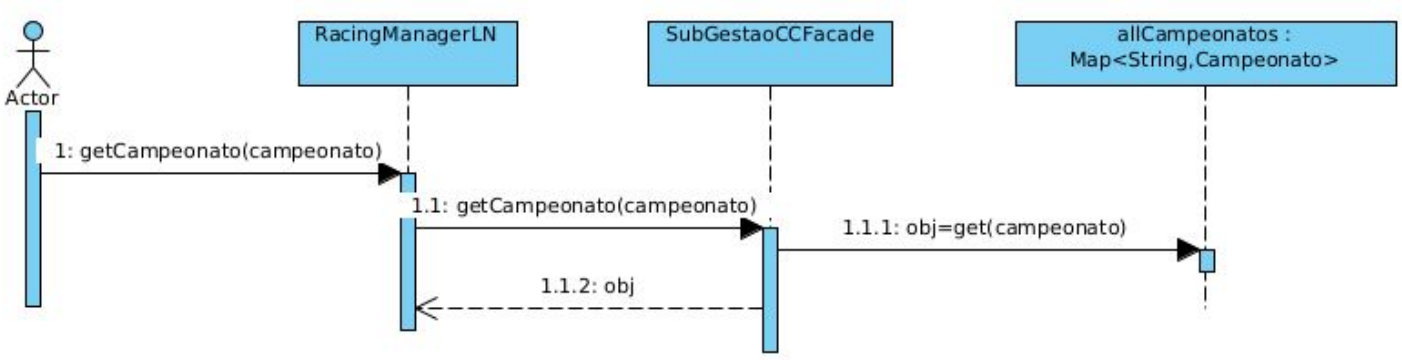


Figura 6.4: Diagrama de sequência referente à apresentação dos campeonatos disponíveis.

6.5 Inscrever no jogo - O sistema guarda a inscrição

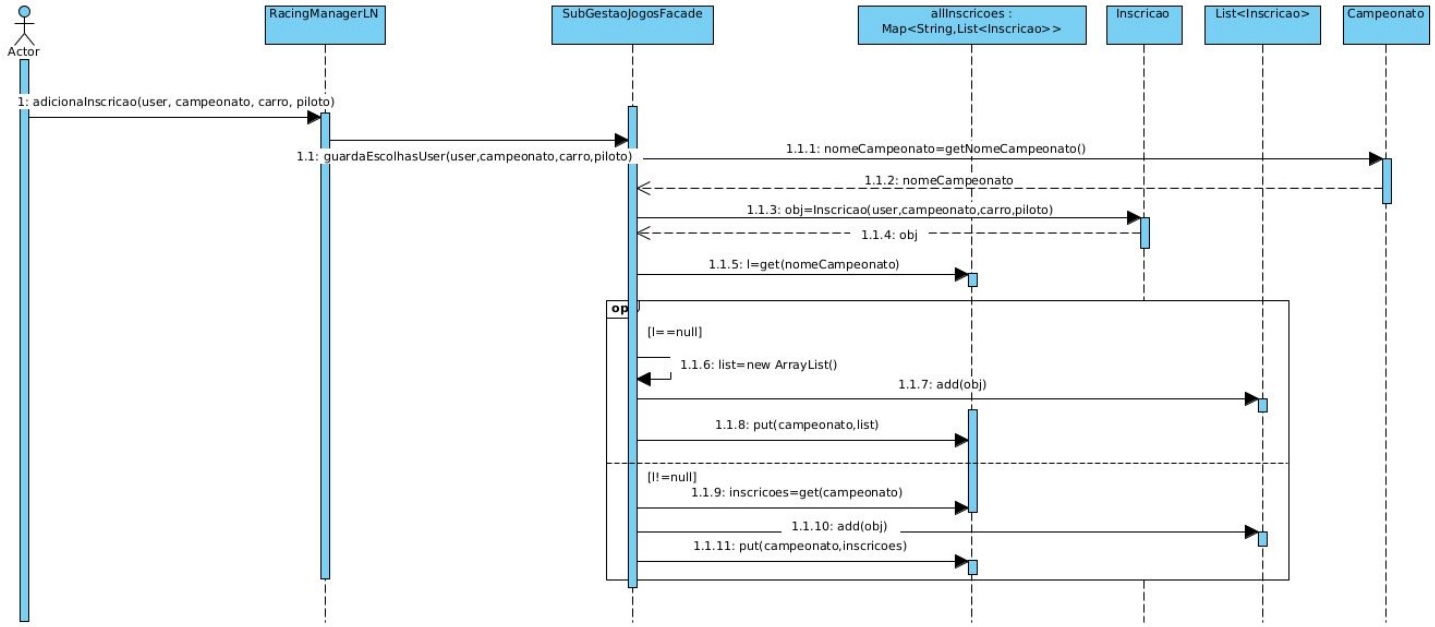


Figura 6.5: Diagrama de sequência relativo à inscrição de um jogador.

6.6 Inscrever no jogo

Elaborámos um diagrama de sequência em que representámos a funcionalidade de inscrição no jogo no seu todo. Por um lado, como já especificámos o getNomesCampeonatos e o getCampeonato decidímos não desenvolver a um nível muito baixo os restantes métodos que se assemelhavam a estes últimos, como é o exemplo do "getNomePilotos"e "getPiloto"; "getCarros"e "getCarro".

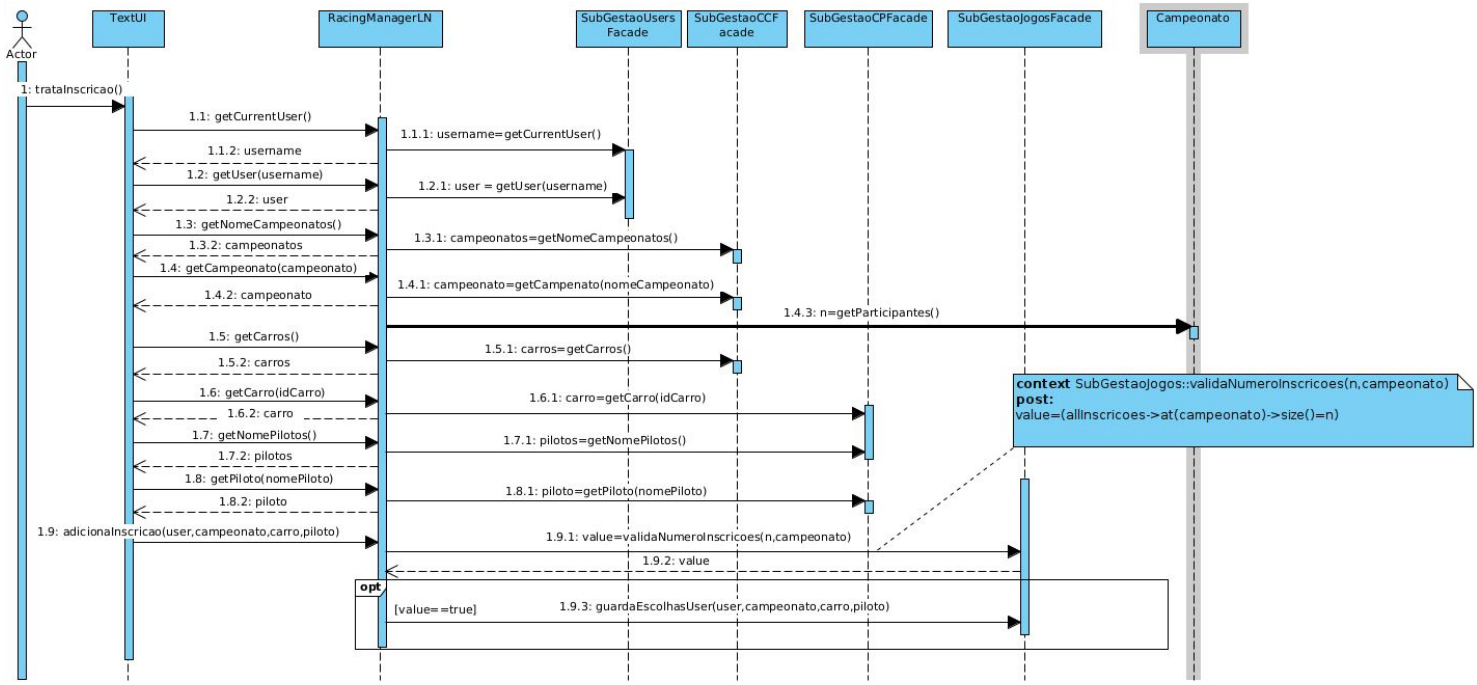


Figura 6.6: Diagrama de sequência relativo à inscrição de um jogador.

6.7 Começar jogo- Início da Simulação

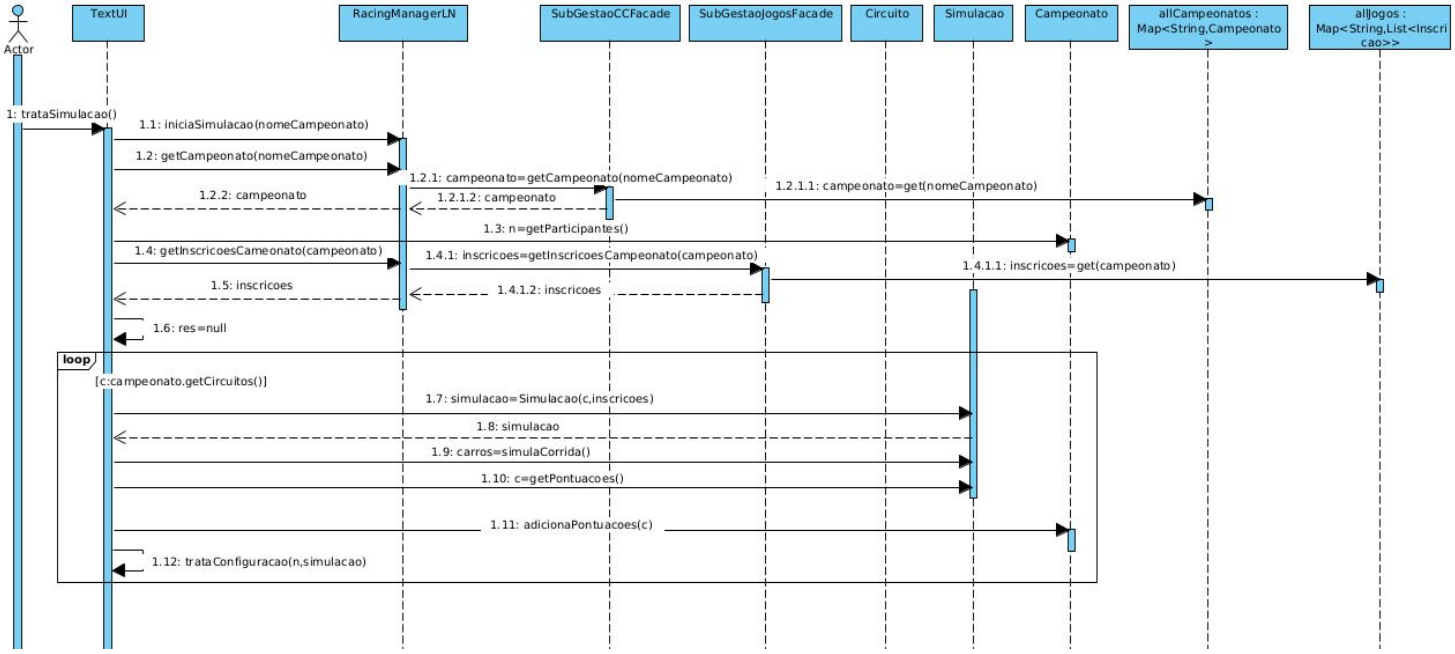


Figura 6.7: Diagrama de sequência relativo à simulação.

6.8 Começar jogo- Atribuição de Pontos

Após a simulação do campeonato irá ocorrer a atribuição da pontuação. De seguida, encontra-se o diagrama de sequências correspondente à distribuição de pontos pelos vários jogadores.

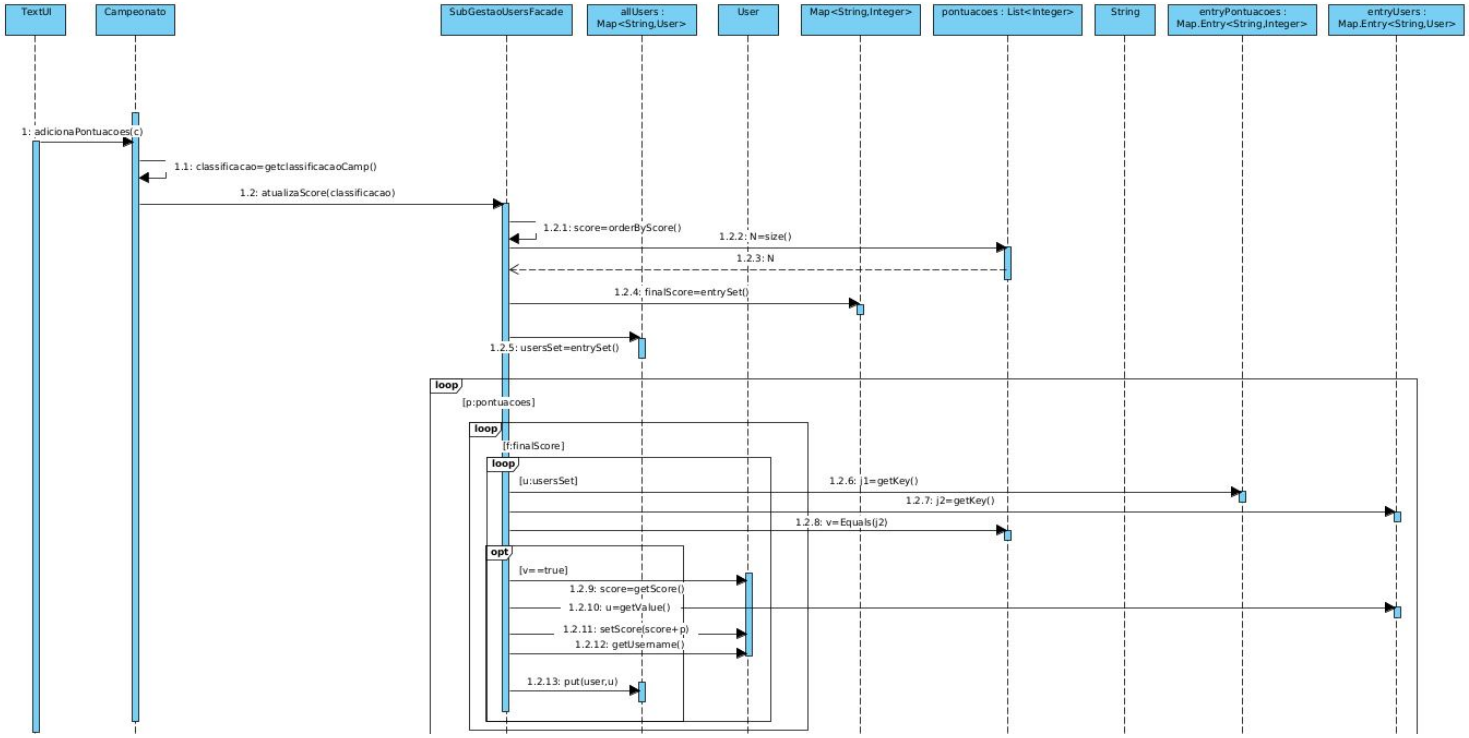


Figura 6.8: Diagrama de sequência relativo à atribuição de pontos.

6.9 Registar conta

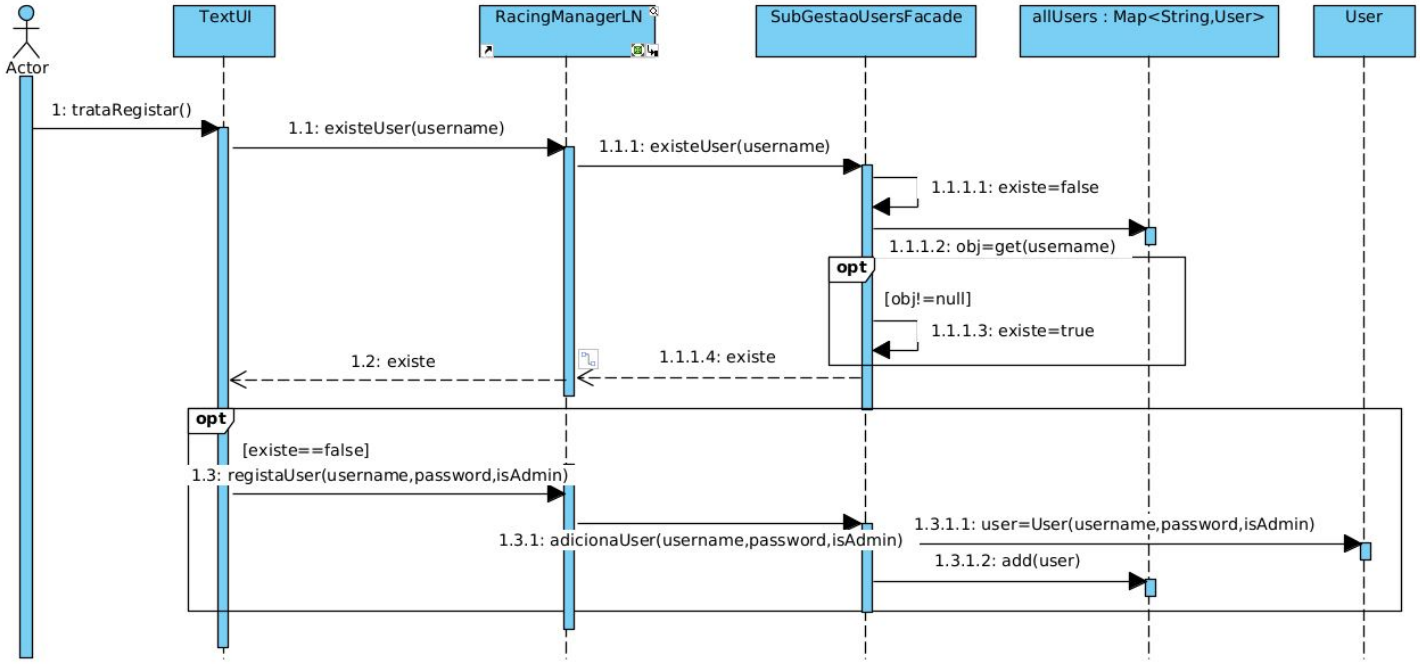


Figura 6.9: Diagrama de sequência relativo ao registo de uma conta.

6.10 Efetuar Login

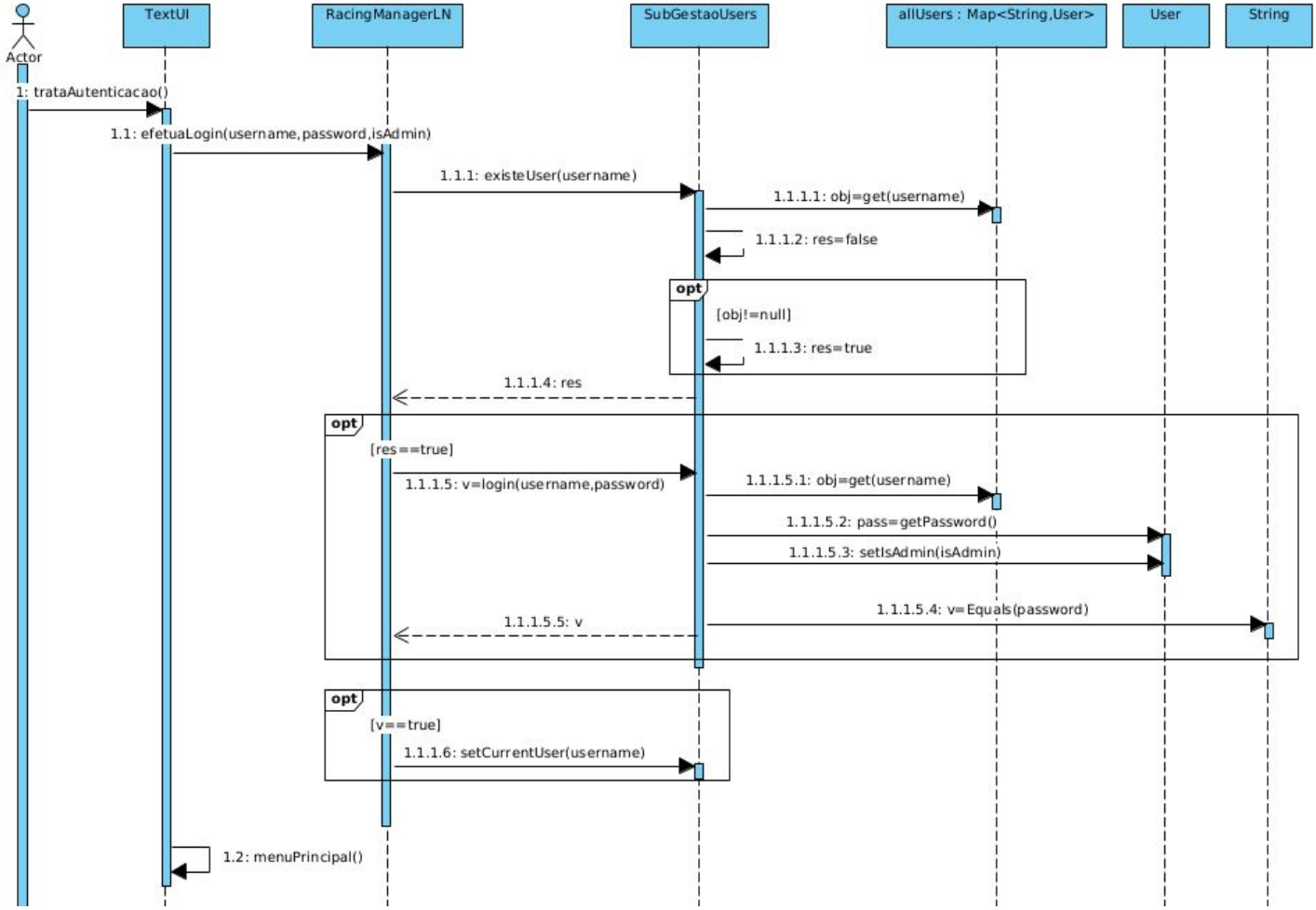


Figura 6.10: Diagrama de sequência relativo à autenticação a uma conta.

6.11 Adicionar Carro C1

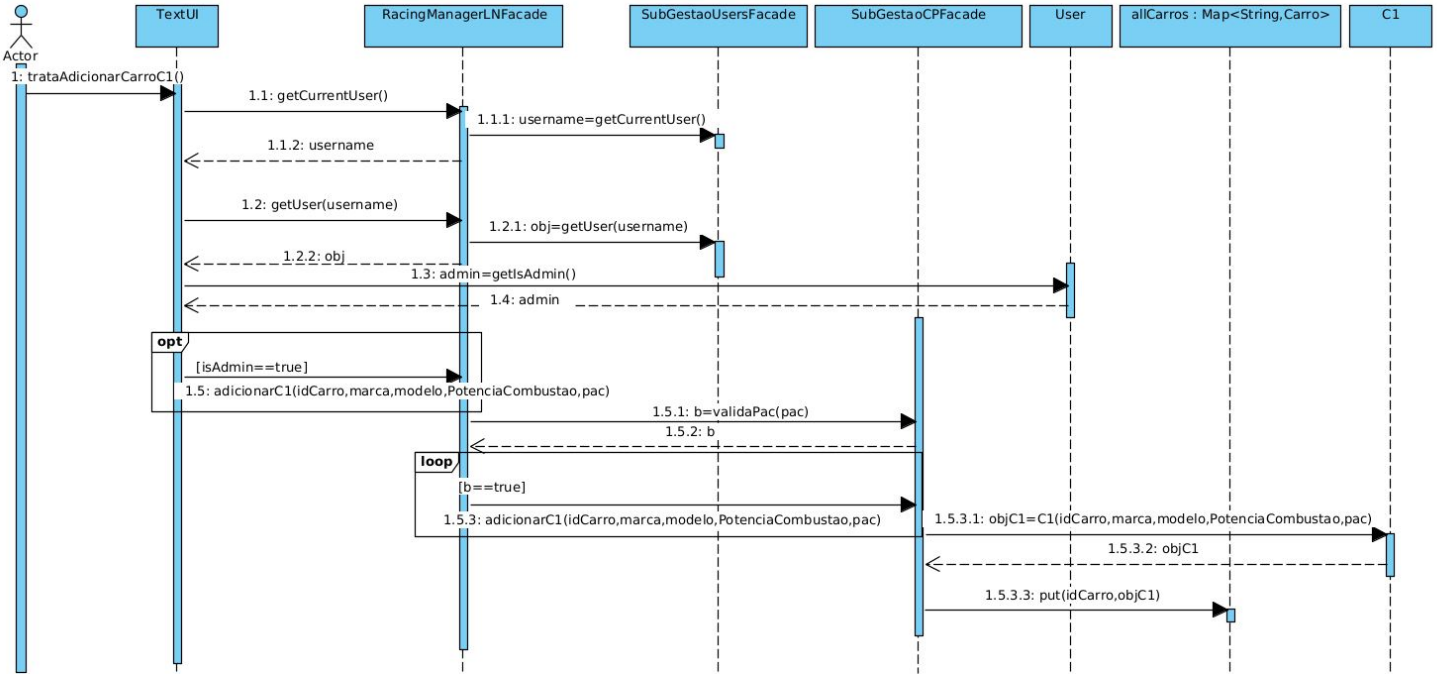


Figura 6.11: Diagrama de sequência relativo à adição de um carro C1.

7. Conclusões e Trabalho Futuro

São várias as empresas que se dedicam ao desenvolvimento de softwares complexos e a projetos de grande escala. Estas necessitam de documentar o seu código, assegurar os prazos de entrega, assim como garantir que as necessidades dos utilizadores finais são satisfeitas. Através deste processo de modelação de software, as equipas que desenvolvem estes softwares evitam perdas de tempo no processo de construção e apenas terão de seguir o plano que foi definido. Os vários diagramas procuram satisfazer os requisitos levantados na fase inicial e desenhar uma arquitetura que procure facilitar a manutenção do sistema.

Após esta fase do projeto, o grupo apercebeu-se da importância que a modelação de software possui, e com o auxílio da ferramenta Visual Paragdim, este processo revelou-se muito mais prático. Com o término desta fase, a equipa sente-se muito mais confiante e capaz para desenvolver a fase seguinte do projeto.