Universidade do Minho
Escola de Engenharia

DESENVOLVIMENTO DE SISTEMAS DE SOFTWARE

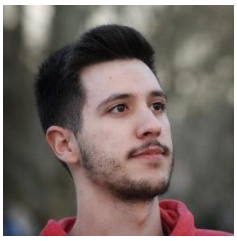
Relatório Projeto Fase II

Sistema de gestão de stocks de um armazém

GRUPO 40

Luís Miguel Pinto A89506

Ana Luísa Carneiro A89533



Pedro Almeida Fernandes A89574

Ana Rita Peixoto A89612

ÍNDICE

INTRODUÇÃO

Na **Fase II** deste projeto propõe-se a **Modelação Conceptual Da Solução** e consequentemente a implementação de:

- Uma arquitetura conceptual do sistema, capaz de suportar o conjunto de Use Cases definido pela equipa docente após a entrega da Fase I, API da lógica de negócio;
- Um Modelo comportamental dos métodos necessários à implementação desses Use Cases, diagramas de sequência.

Deste modo, através da **API da lógica de negócio**, conseguimos ter noção dos métodos que descrevem o sistema, dividindo-os por subsistemas e, por conseguinte, originando o **diagrama de componentes**. Além disso, também foi elaborado o **diagrama de classes** respetivo a cada componente e o **diagrama de packages** que os agrega. Por último, de forma a representar a interação entre os objetos, foram criados **diagramas de sequência** para cada método da API.

ALTERAÇÕES FASE I

Após análise do trabalho e orientações da equipa docente, decidimos efetuar as seguintes mudanças em relação ao proposto na fase I:

- Adição de entidade percurso no modelo de domínio;
- Adição de Use Case “Terminar Sessão”.

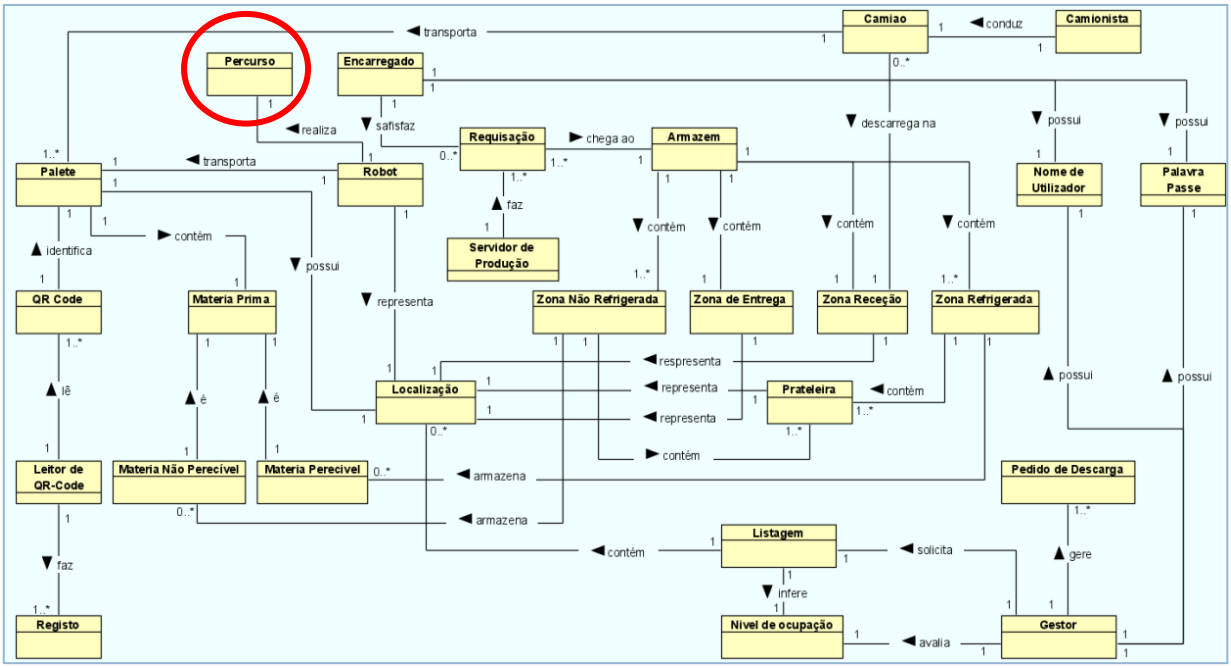


Figura 1: Modelo de domínio alterado

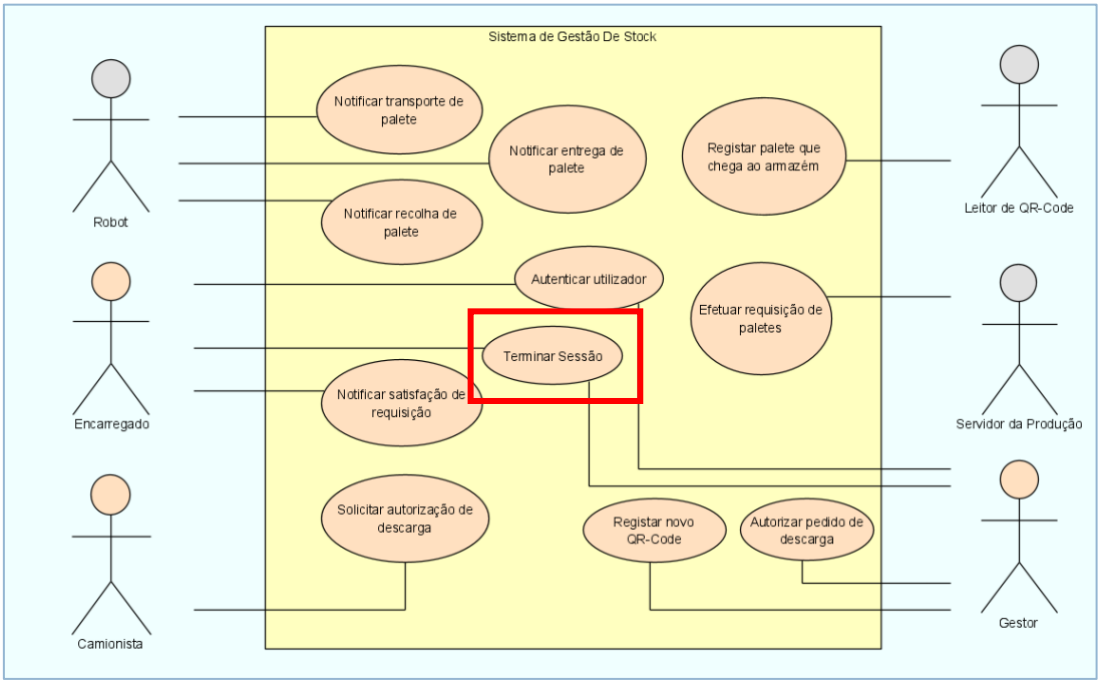


Figura 2: Diagrama de Use Case alterado

API DA LÓGICA DE NEGÓCIO

Tal como foi proposto, foram identificadas as responsabilidades do sistema a partir dos Use Cases sugeridos e a partir destas, os métodos da API.

API global e Responsabilidades

Responsabilidade	Método da API
Validar autenticação	+validarAutenticacao(codUtilizador : string, palavraPasse : string) : boolean
Terminar sessão do utilizador	+terminarSessao(codUtilizador : string) : void
Determinar listagem	+getListagem() : List<String>
Validar QR-Code	+validarQRCode(qrCode : string) : boolean
Registar palete que chega ao armazém	+registarPalete(qrCode : string) : void
Determinar palete a transportar	+getPalete(estado : string) : Palete
Determinar localização da palete	+getLocalizacao(palete : Palete) : Localizacao
Determinar robot para transportar	+getRobot() : Robot
Determinar percurso do robot	+getPercurso(destino : Localizacao, palete : Palete) : List <Integer>
Notificar robot	+notificarRobot(palete : Palete, percurso : List<Integer>, robot : Robot) : void
Atualizar estado da palete	+atualizarEstadoPalete(palete : Palete) : void
Atualizar estado da palete do robot	+atualizarEstadoPalete(palete : Palete, localizacao : Localizacao) : void
Determinar stock/validar disponibilidade das paletes	+verificaStock(requisicao : List<String>) : List<String>
Registar requisição de paletes	+registarRequisicao(qrCode : List<String>) : void

Divisão dos métodos da API

Os métodos da API global do sistema foram divididos em diferentes subsistemas, de modo a agrupa-los de acordo com a sua funcionalidade.

Abastecimento

Neste subsistema encontram-se os métodos relacionados com o robot e localizações do armazém.

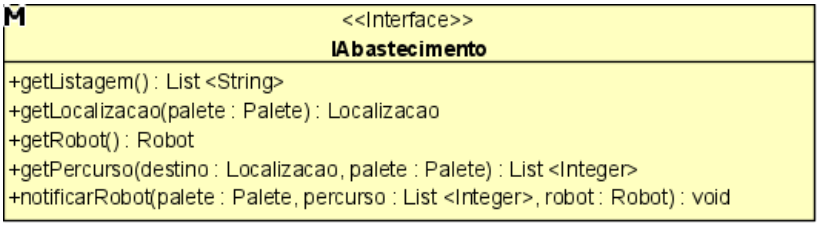


Figura 3: API do subsistema Abastecimento

Catálogo

No substistema catálogo foram identificados os métodos associados às paletes e ao seu armazenamento, assim como todas as ações possíveis sobre elas.

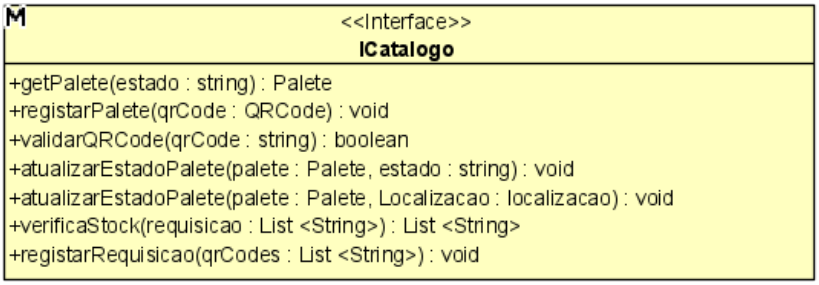


Figura 4: API do subsistema Catálogo

Trabalhador

Neste subsistema trabalhador encontram-se os métodos relacionados com os utilizadores que acedem ao sistema, isto é, a autenticação do mesmo e o terminar sessão.

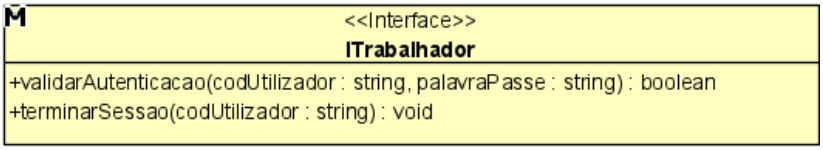


Figura 5: API do subsistema Trabalhador

DIAGRAMA DE COMPONENTES

O **diagrama de componentes** permite descrever os componentes do sistema e as dependências entre eles. Assim, é possível identificar o que é necessário para construir o sistema. Cada componente comunica com outra através de interfaces, pois apenas sabem que tem uma certa API.

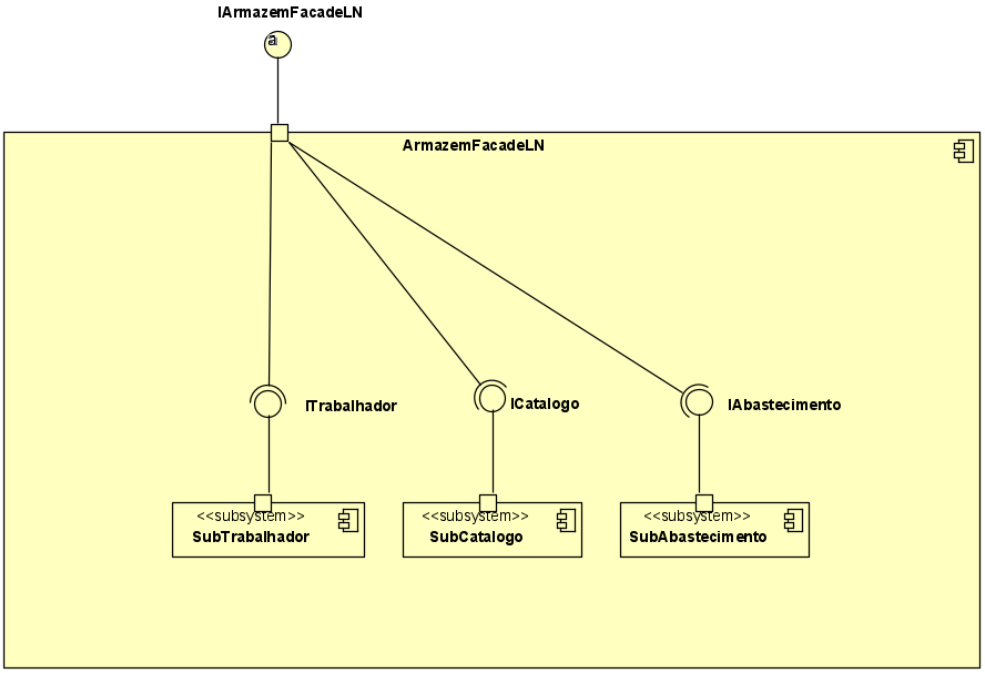


Figura 6: Diagrama de componentes

DIAGRAMA DE PACKAGES

De modo a identificar as dependências entre as diferentes classes de um sistema, foi elaborado um **diagrama de packages**. Cada package é caracterizado por um agrupamento de classes e representa o subsistema respetivo.

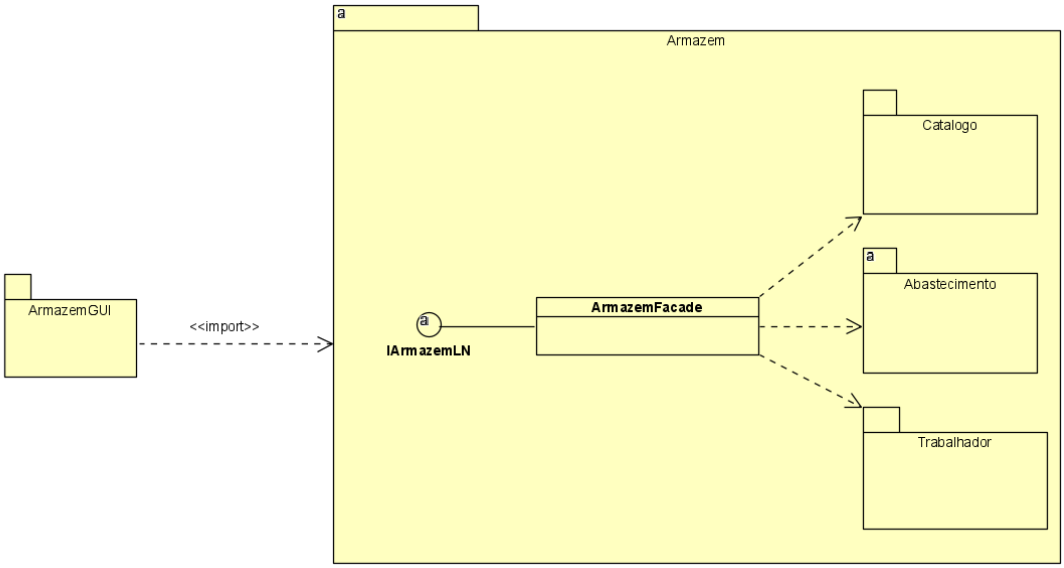


Figura 7: Diagrama de packages

DIAGRAMA DE CLASSES

Um **diagrama de classes** tem como função agrupar classes de forma a criar um subsistema. Uma classe tem dois objetivos fundamentais: facilitar a reutilização — através da reutilização de classes previamente desenvolvidas em novos sistemas; facilitar a manutenção — o sistema deverá ser desenvolvido de forma a que a alteração de uma classe tenha o menor impacto possível no resto do sistema.

Lógica de negócio do armazém

A **lógica de negócio** contém os métodos que representam a API e que serão divididos pelos respetivos subsistemas. Deste modo, criamos a interface **IArmazemFacadeLN** e a respetiva classe **ArmazemFacade** que implementa os seus métodos.

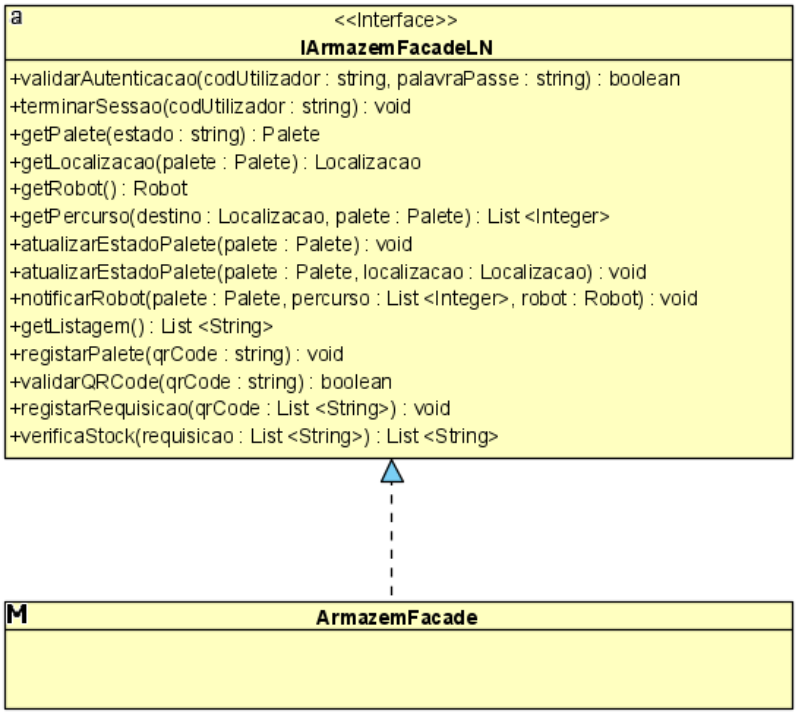


Figura 8: Lógica de negócio

Abastecimento

O diagrama de classes do subsistema Abastecimento é constituído pela interface **IAbastecimento** e pela classe **AbastecimentoFacade** que implementa os seus métodos. Adicionalmente, existem também as classes **Corredor** e **Robot**. Deste modo, existem estruturas que permitem armazenar a informação sobre os robots do sistema e os diferentes corredores na classe **AbastecimentoFacade**.

Além disso, também é implementado neste subsistema a classe **Mapa** que representa o mapa do nosso armazém (figura 10), em que cada vértice representa o código do corredor. Nesta classe esta contida uma matriz de adjacência, isto é, para cada vértice do mapa existe uma lista de vértices adjacentes, para posteriormente conseguirmos determinar o percurso do robot, que será uma lista de vértices.

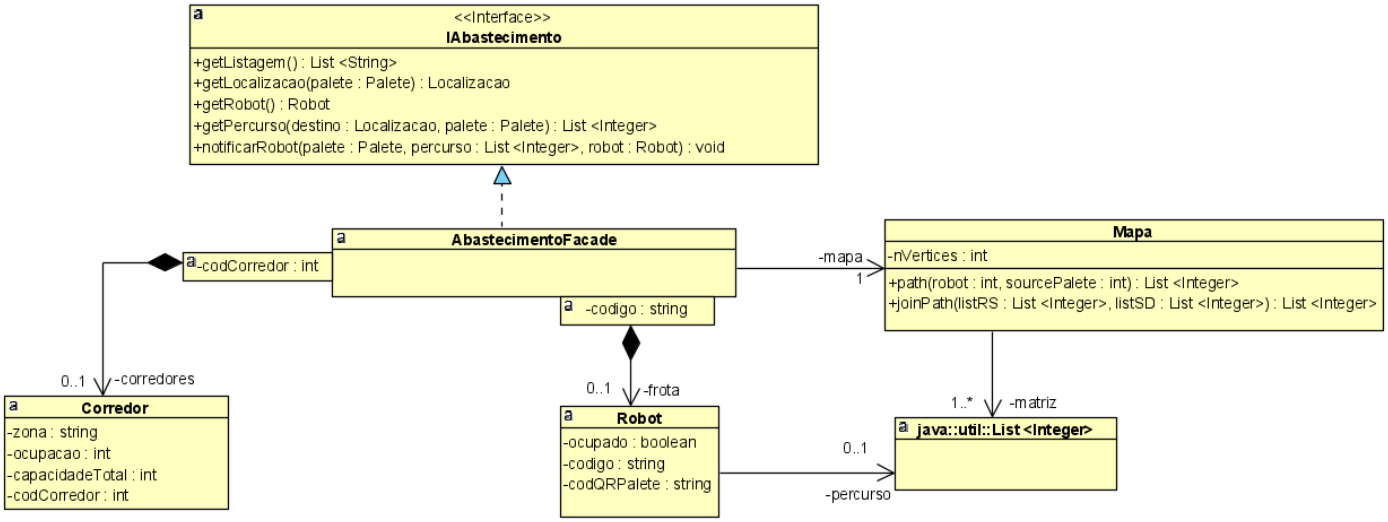


Figura 9: Diagrama de classes do Abastecimento

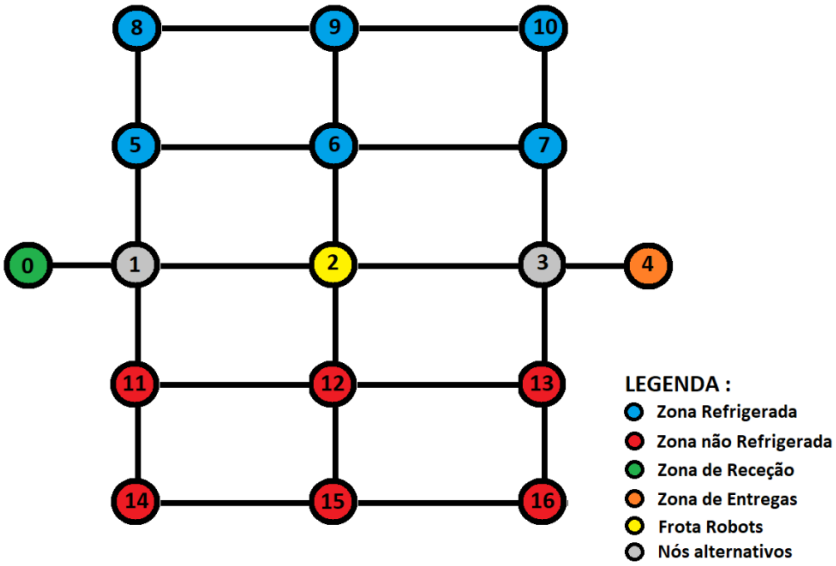


Figura 10: Representação do mapa do armazém

Trabalhador

Neste diagrama de classes está presente a classe **TrabalhadorFacade** que implementa os métodos da interface **ITrabalhador**. Foi implementada uma classe **Utilizador** que representa todas as entidades que podem utilizar o sistema, tais como o gestor e o encarregado.

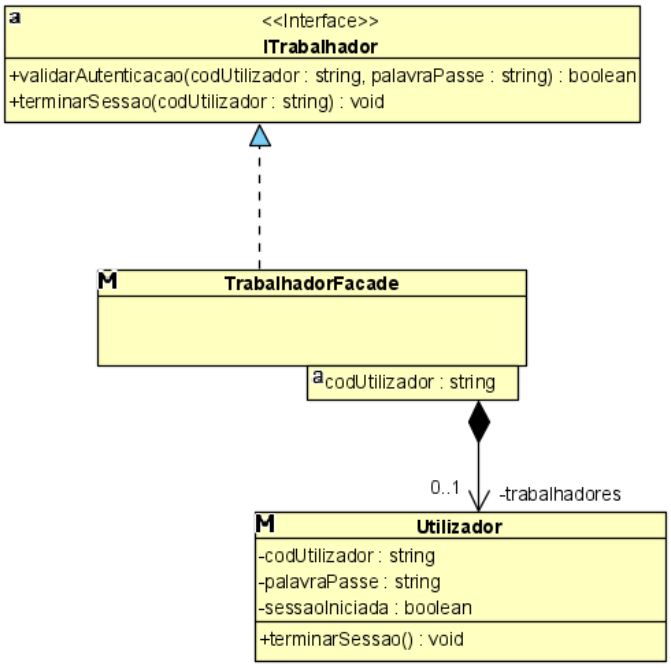


Figura 11: Diagrama de classes do Trabalhador

Catálogo

Neste subsistema está presente a interface **ICatalogo** e a respetiva classe **CatalogoFacade** que a implementa. Além disso, estão presentes classes adicionais como **Palete** que é constituída pelas classes **Localização** e **QRCode.** A classe **CatalogoFacade** contém diferentes estruturas de dados que permitem identificar o estado da palete em cada momento. Assim, as listas requisicao, rececao, aguardaTransporte e robot, referem-se a paletes requisitadas, que se encontram na receção, que aguardam transporte e paletes que se encontram em transporte pelos robots, respetivamente. Adicionalmente, implementamos também um *map* que se refere ao **armazenamento** total do armazém (zonas perecíveis e não perecíveis), contendo todos os QRCodes aceites e as respetivas listas de paletes associadas a cada código.

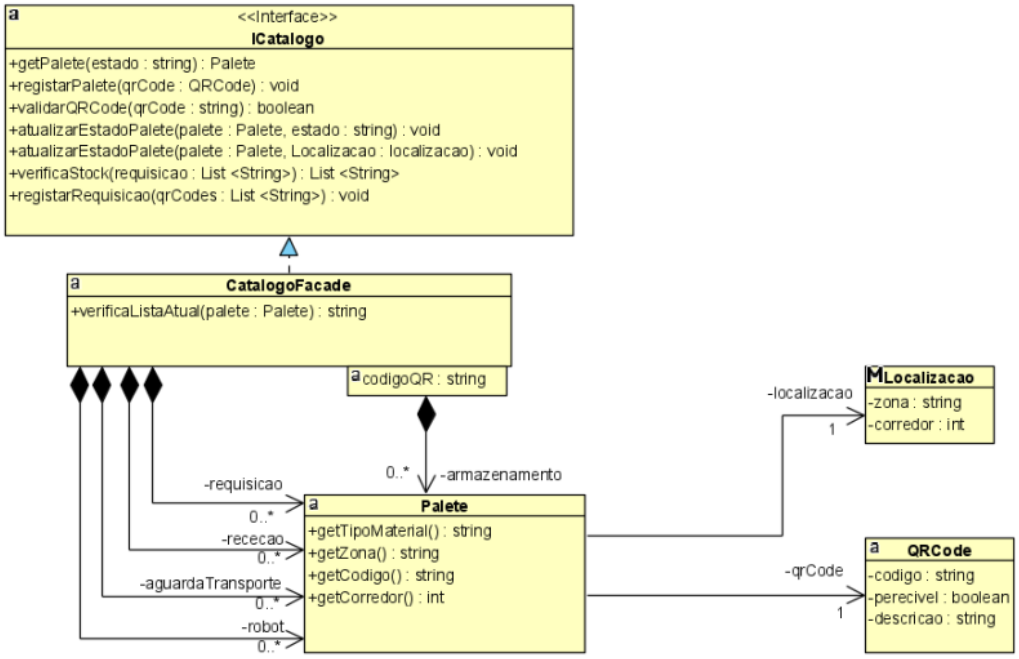


Figura 12: Diagrama de classes do Catálogo

DIAGRAMAS DE SEQUÊNCIA

Os **diagramas de sequência** têm como objetivo focar no ordenamento temporal da troca de mensagens, permitindo observar como é que os objetos comunicam entre si.

Package Trabalhador

Terminar sessão

Este método permite terminar a sessão de um utilizador.

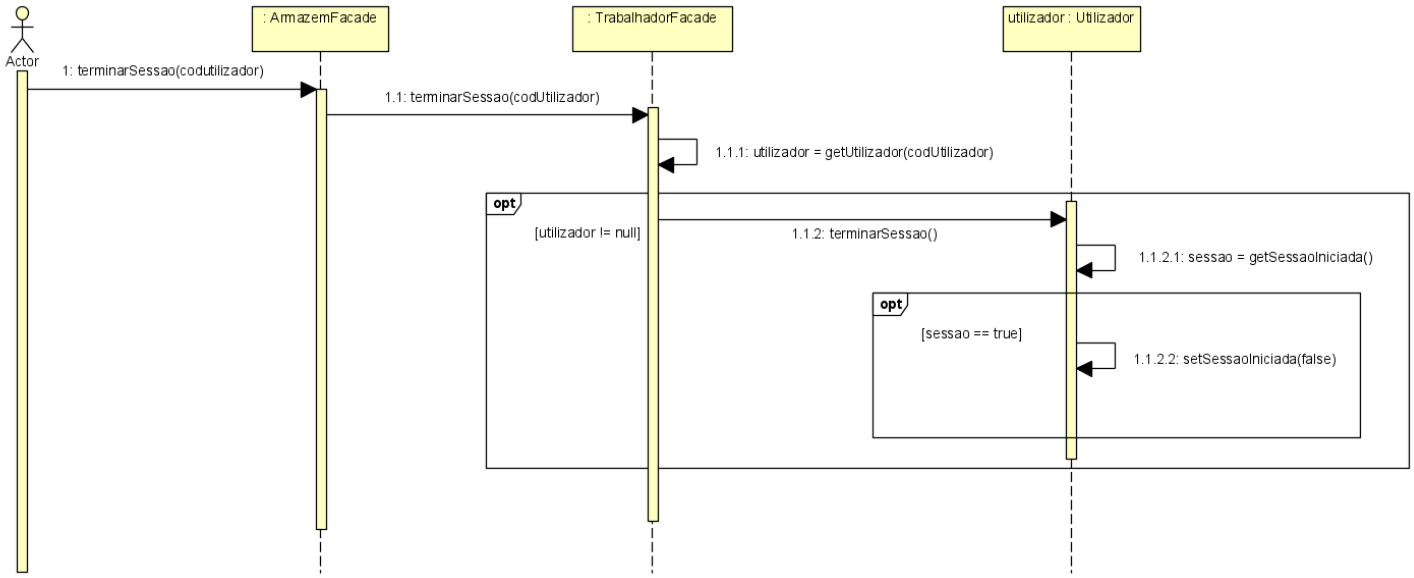


Figura 13: Diagrama do método terminarSessão

Validar autenticação

O método **validarAutenticacao** permite um utilizador do sistema autenticar-se.

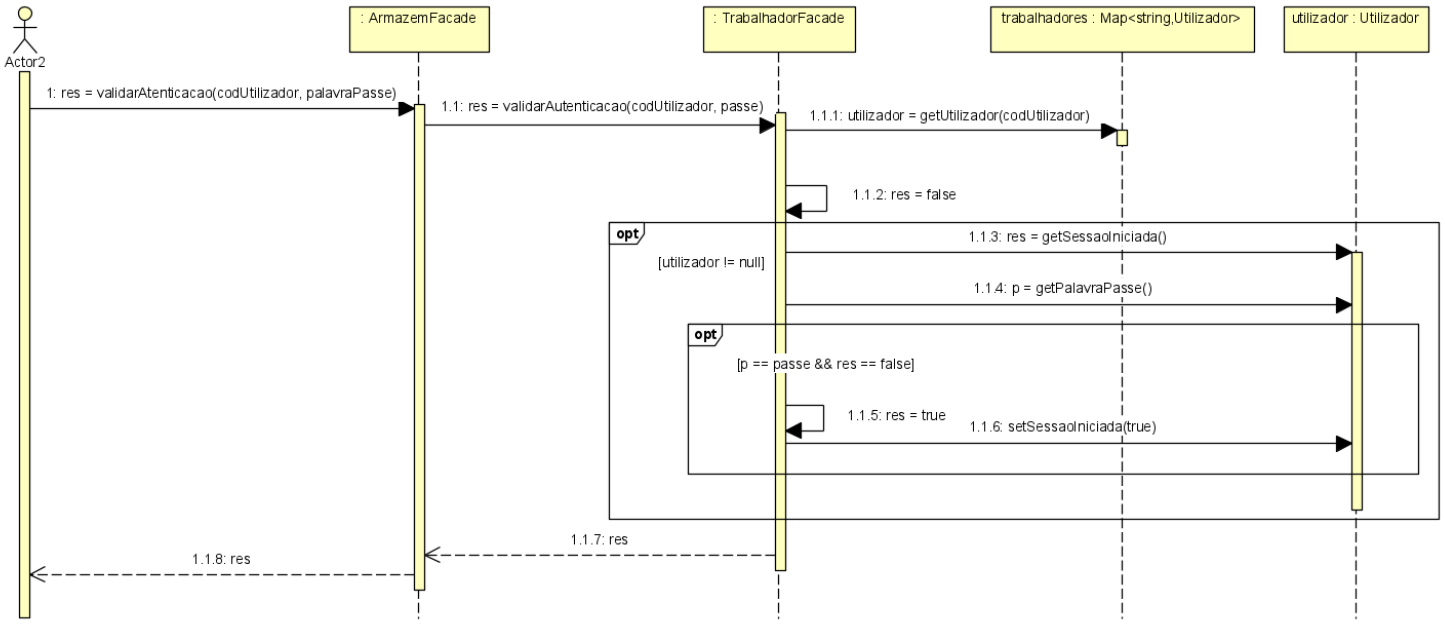


Figura 14: Diagrama de sequência do método Validar Autenticação

Package Abastecimento

Obter Listagem

O método **getListagem** permite obter uma lista com as localizações do sistema e as respetivas ocupações.

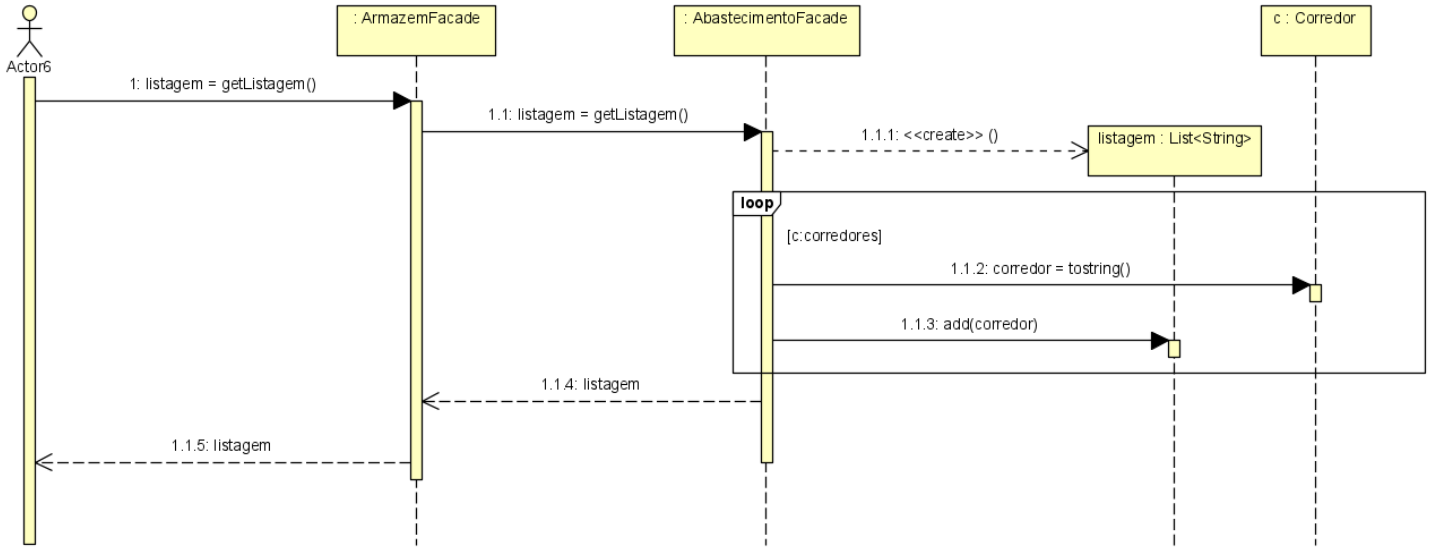


Figura 15: Diagrama de sequência do método getListagem

Obter Robot

O método **getRobot** permite atribuir um robot a uma palete que necessita de transporte.

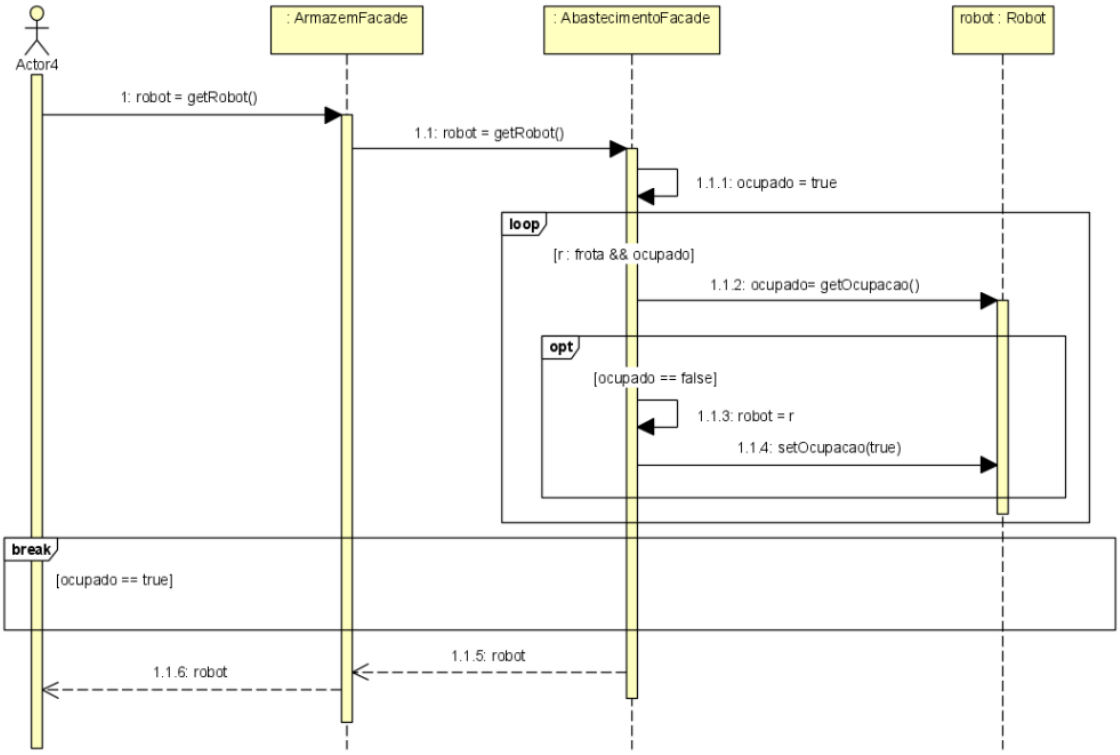


Figura 16: Diagrama de sequência do método getRobot

Obter Localização

O método **getLocalização** permite obter a localização futura de uma palete, dependendo da sua localização anterior. Para isso, foi adicionado um método verificaListaAtual que verifica em que lista se encontra a palete em questão. Adicionalmente, também foi utilizado o método getTipoMaterial que indica se a palete é constituída por matéria perecível ou não perecível.

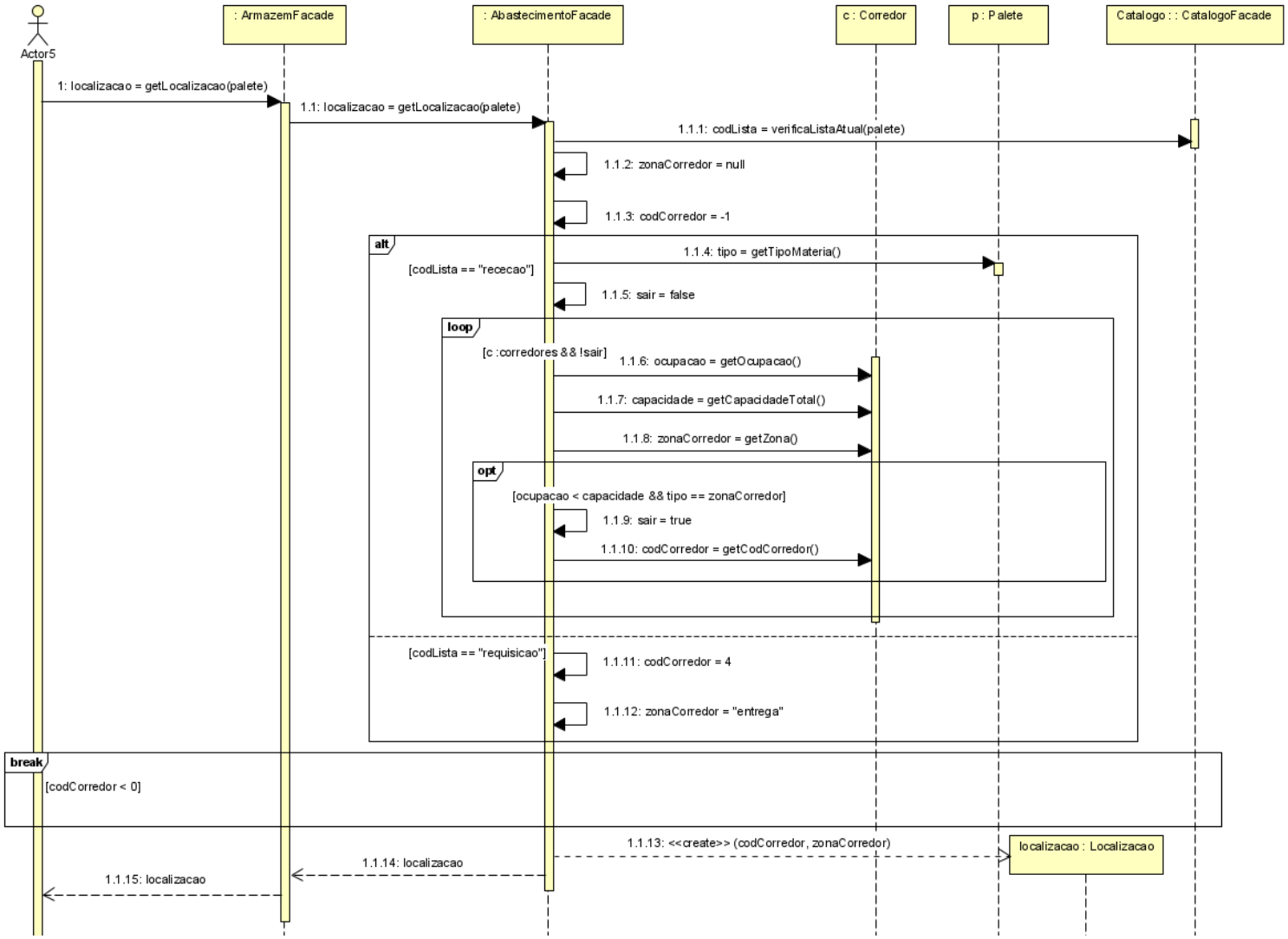


Figura 17: Diagrama de sequência do método getLocalizacao

Obter Percurso

No método **getPercurso** é calculado o percurso do robot de acordo com os seus pontos inicial e final.

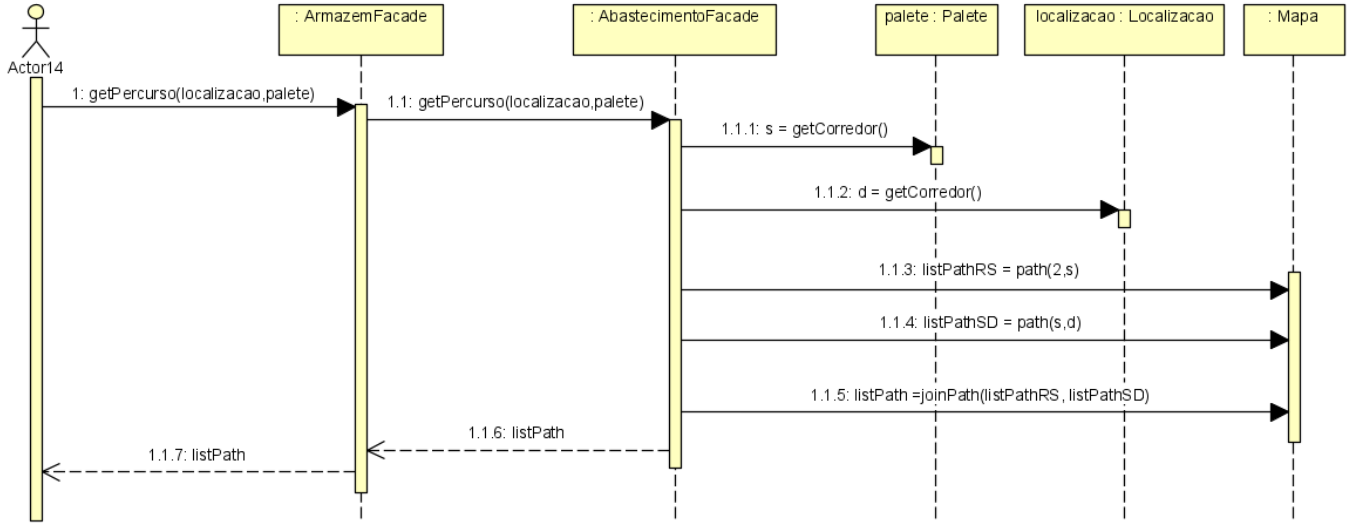


Figura 18: Diagrama de sequência do método getPercurso

Neste diagrama de sequência, foi necessário adicionar novos métodos: *path* e *joinPath*. O *path* permite calcular um caminho entre 2 nós, através de uma matriz de adjacência. Por exemplo, se o robot precisar de armazenar uma palete proveniente da receção, é necessário que este se desloque primeiro à receção (2->0) e posteriormente ao local de armazenamento (0->6, por exemplo). A junção destes dois caminhos é a função do método *joinPath*.

Notificar Robot

Este método permite notificar o robot da palete a transportar, do percurso e do respetivo robot. O método getCodigo adicionado retorna o código dentro do QRCode da palete.

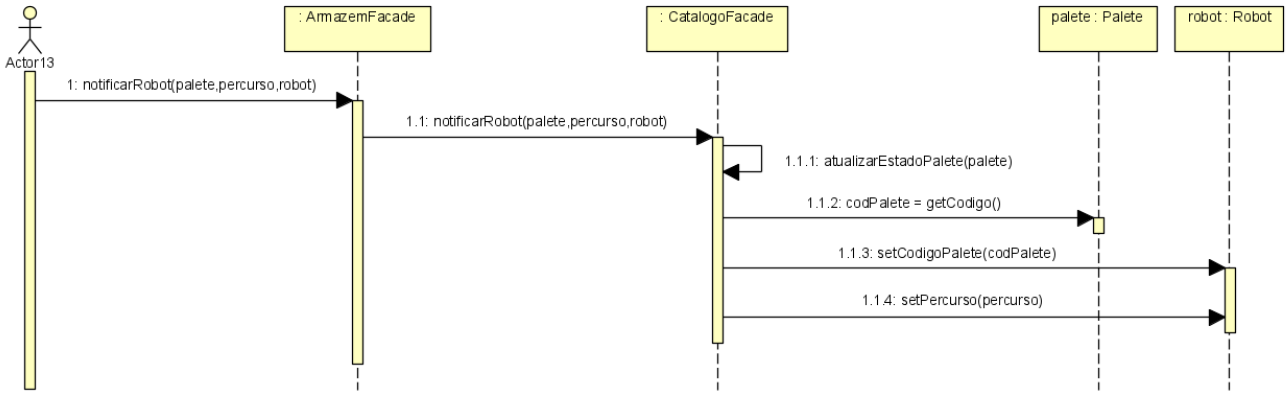


Figura 19: Diagrama de sequência do método notificarRobot

Package Catálogo

Obter Palete

Este método permite obter uma palete para armazenar, que se pode encontrar requisitada ou na zona de receção.

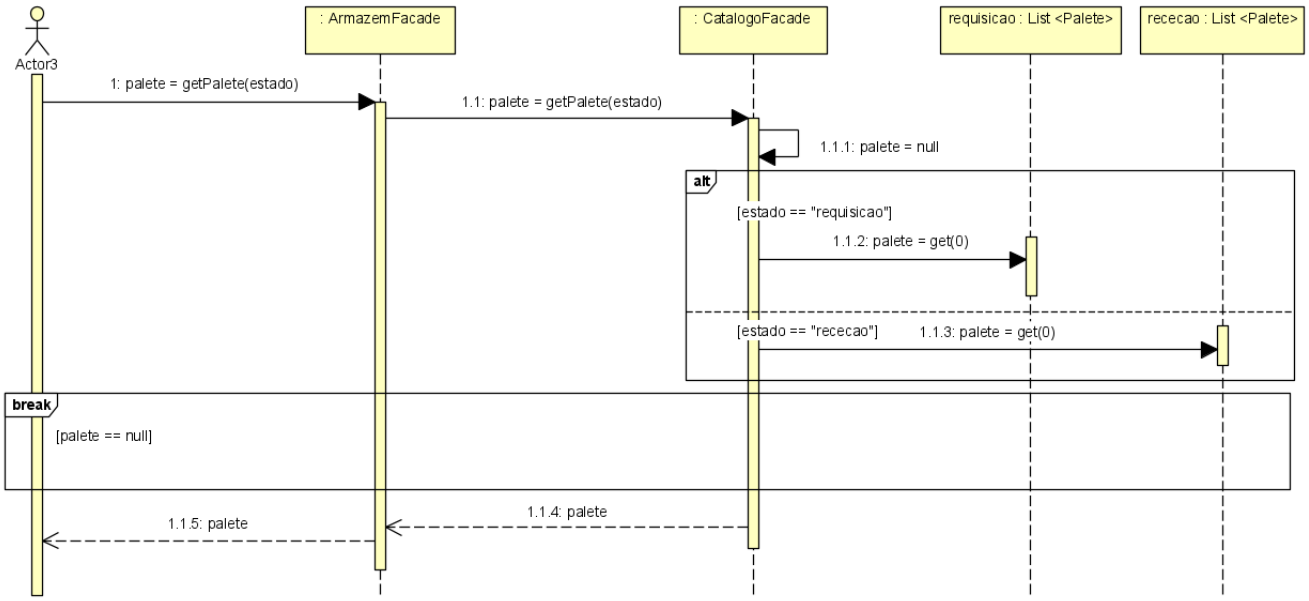


Figura 20: Diagrama de sequência do método getPalete

Registar Palete

Este método permite registar uma nova palete que chega ao armazém.

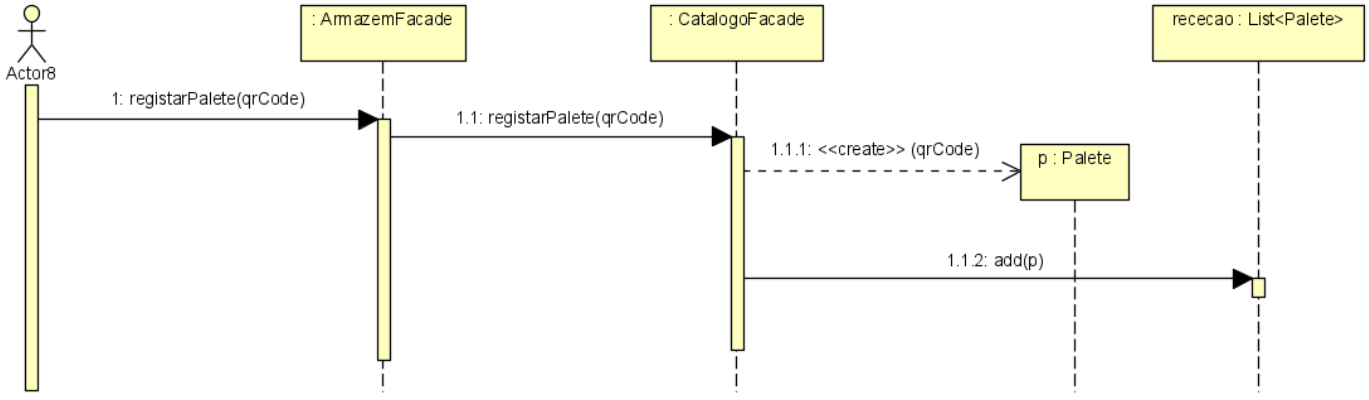


Figura 21: Diagrama de sequência do método registarPalete

Validar QR-Code

Este método permite verificar se um QR-Code é válido.

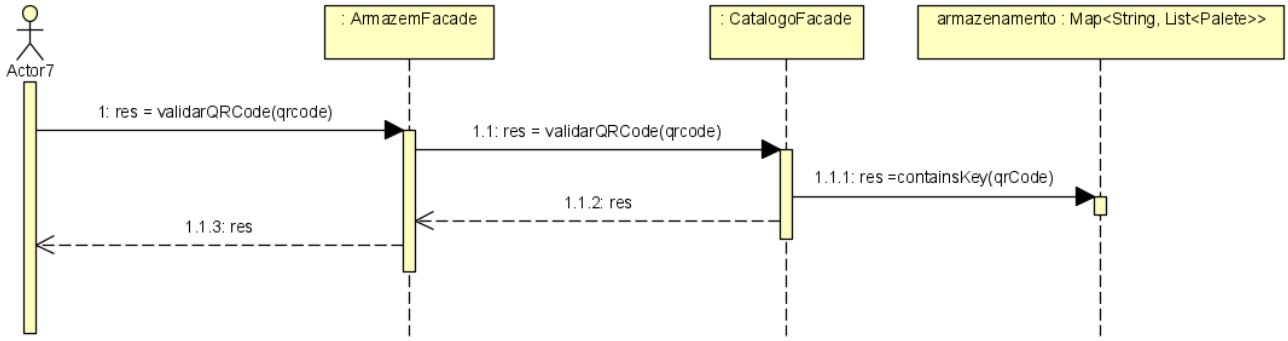


Figura 22: Diagrama de sequência do método validarQRCode

Atualizar Estado de uma Palete

O método **atualizarEstadoPalete** trata de alterar o estado de uma palete, podendo mudá-la de receção/requisição para aguardar transporte ou de aguardar transporte para robot.

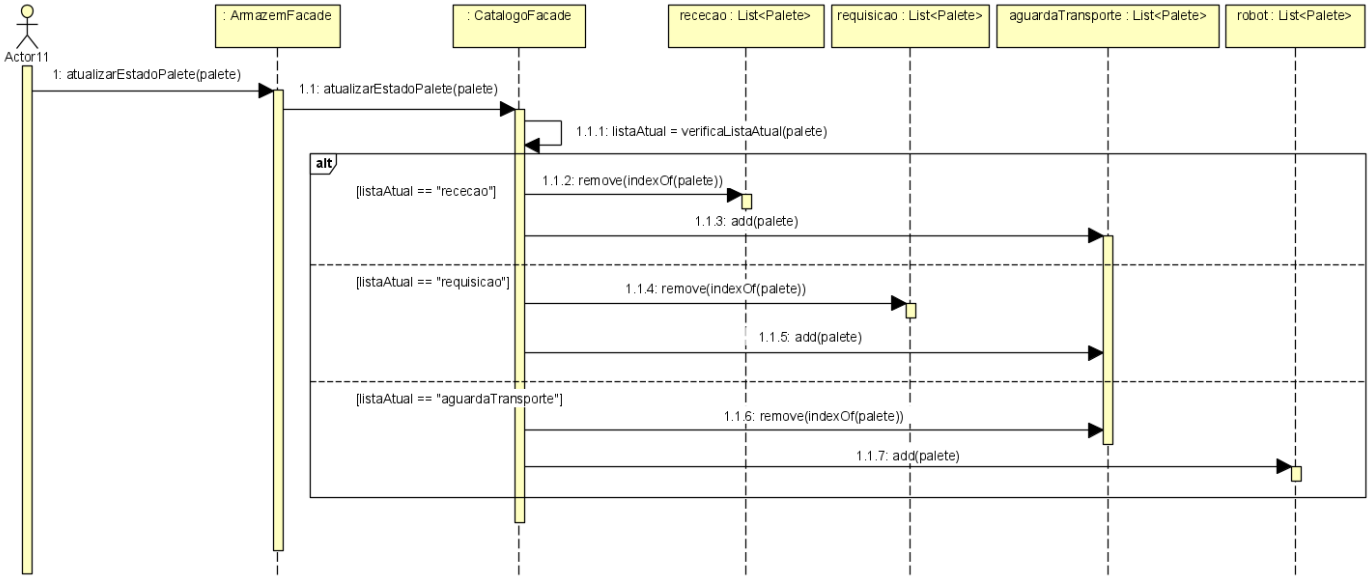


Figura 23: Diagrama de sequência do método atualizarEstadoPalete

Atualizar Estado de uma Palete do Robot

O método **atualizarEstadoPalete** atualiza o estado de uma palete quando chega ao seu destino, quer seja quando é entregue ou quando é armazenada. No caso em que a palete é armazenada, é atualizada a sua localização com o novo lugar de armazenamento. Neste método foi necessário utilizar o método getZona que fornece a zona em que a palete se encontra.

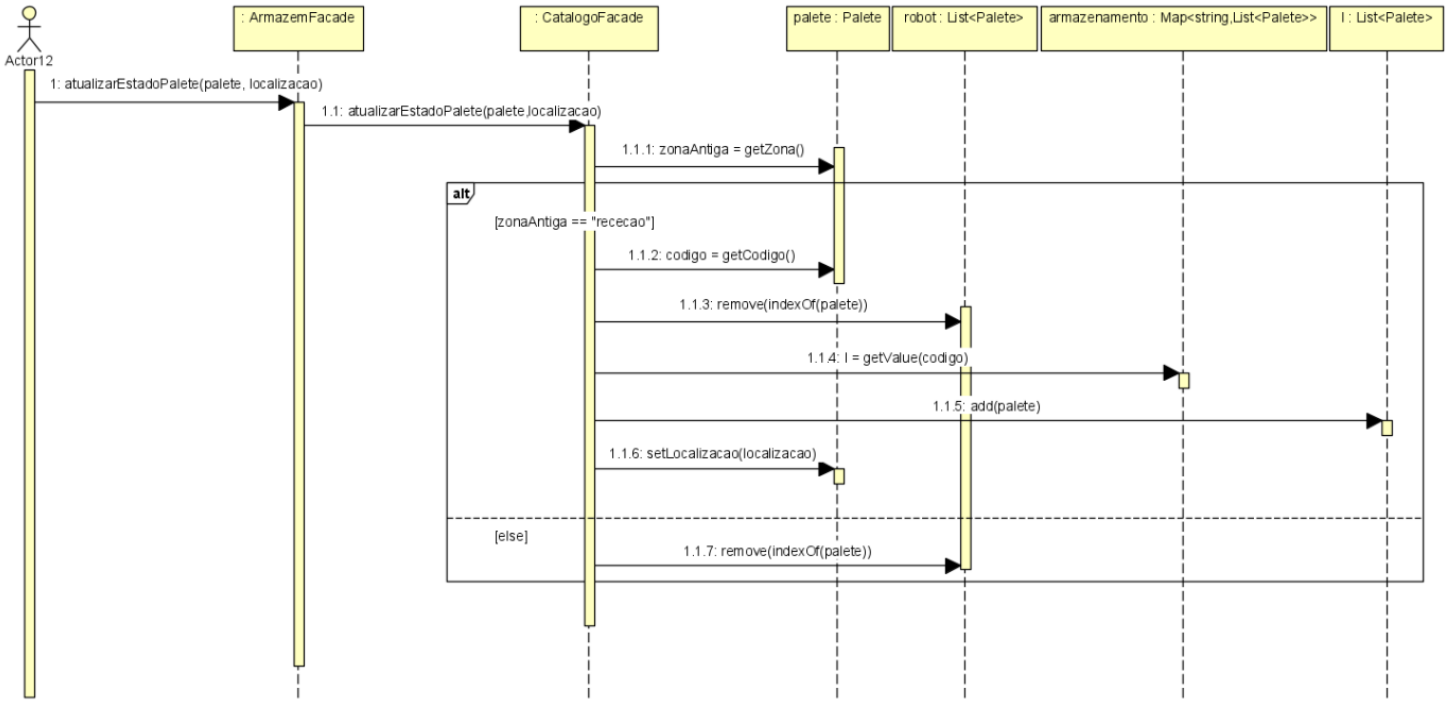


Figura 24: Diagrama de sequência do método atualizarEstadoPalete

Verificar Stock

O método **verificaStock** verifica se existe stock suficiente para concretizar uma dada requisição. Caso não exista, devolve uma lista com as paletes que não estão em stock.

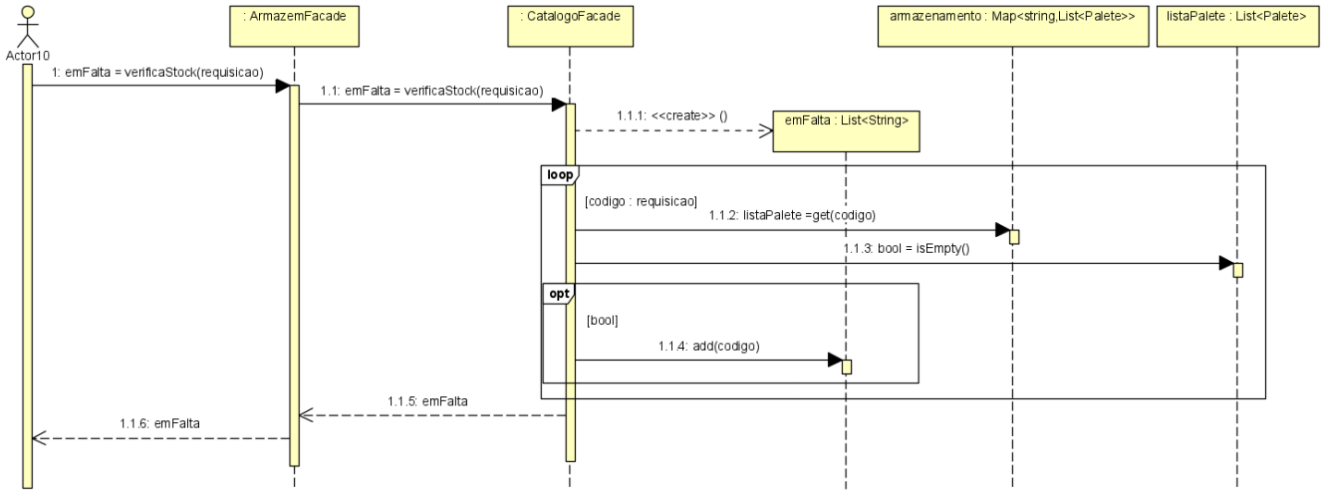


Figura 25: Diagrama de sequência do método verificaStock

Registar Requisição

O método **registarRequisicao** permite registar uma requisição na respetiva lista.

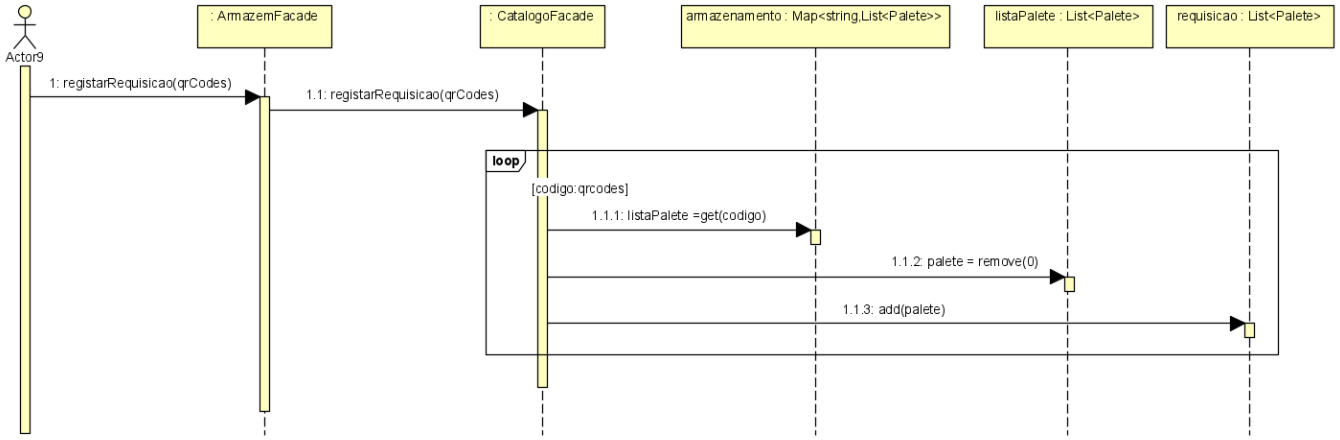


Figura 26: Diagrama de sequência do método registarRequisicao

CONCLUSÃO

Dada por concluída a segunda fase do projeto, consideramos favorável elaborar uma análise crítica do trabalho realizado.

No caso da API da lógica de negócio, consideramos que os métodos obtidos são suficientes para suportar as funcionalidades do sistema. Além disso, esses métodos foram divididos em diferentes subsistemas de forma lógica, originando os diagramas de componentes e de packages.

Através do diagrama de classes implementamos as classes que consideramos necessárias para o sistema. Juntamente com os diagramas de sequência, conferimos uma noção temporal e sequencial dos métodos, permitindo assim um fundamento para a fase III do projeto, que se irá traduzir na implementação do sistema.

Em particular, sentimos dificuldade em implementar o algoritmo de cálculo de percurso do robot. No entanto, consideramos que conseguimos contornar esse problema, de acordo com o raciocínio mencionado anteriormente.

Deste modo, apesar das dificuldades sentidas, consideramos que conseguimos superá-las através do apoio dos docentes.