Sistema de Gestão de Stock

Trabalho realizado por:

João Figueiredo Martins Peixe dos Santos

Francisco Alves Andrade

Luís Filipe Cruz Sobral

Paulo Silva Sousa

A89520 João Santos

A89474 Luís Sobral

A89465 Paulo Sousa

A89513 Francisco Andrade

Grupo 14
Projeto DSS
2020/2021
Universidade do Minho

Índice

1 Objetivos

Partindo do trabalho efetuado nas partes 1 e 2, a parte 3 consiste na demonstração do processo de transporte de paletes. Sendo este transporte desde que o Código QR é lido até que a palete correspondente é colocada na prateleira. Para tal demonstração, é necessária a implementação de **cinco** Use Case:

- Comunicar código QR;
- Sistema comunica ordem de transporte;
- Notificar recolha de paletes;
- Notificar entrega de paletes;
- Consultar listagem de localizações;

De modo a conseguir uma implementação segura e versátil, seguimos uma implementação do sistema - arquitetura em três camadas (Data, Business e Presentation), sendo esta última camada implementada em modo texto.

Com a meta de alcançar os objetivos desta fase decidimos proceder à implementação dos Use Case, no entanto, numa fase inicial, essa implementação seria apenas em memória. A implementação dos Use Case é acompanhada por um esboço da camada Presentation, de modo a facilitar os testes e debug dos Use Case. Por último, implementamos a cada de dados, de forma a conseguir concluir os objetivos.

2 Alterações realizadas nos modelos anteriores

Tendo em conta o enunciado da 3ª e última fase, foi necessário realizar várias alterações aos modelos apresentados nas fases 1 e 2.

2.1 Diagrama de Classes

O diagrama de classes teve de ser remodelado, visto que algumas classes deixaram de ser necessárias e foram, por isso, eliminadas. Exemplos disso são as clases: listagem, notificações, servidor de produção e sua respetiva interface, requisições, entre outras.

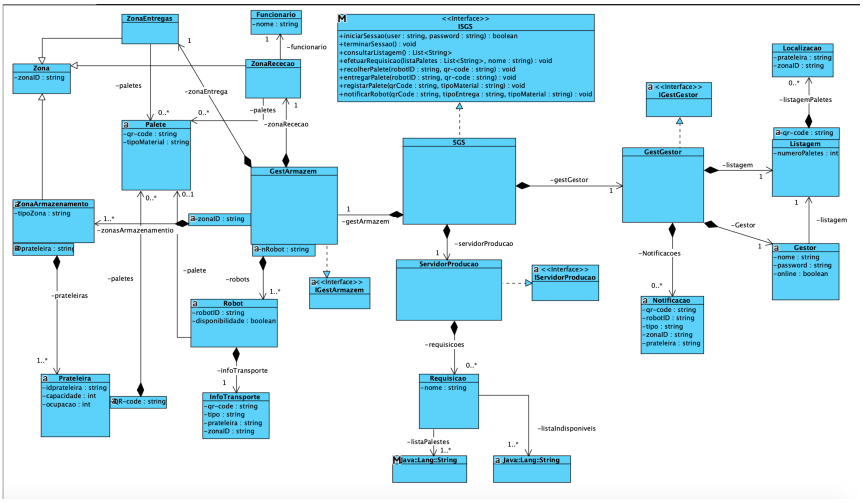

Figura 1 - Diagrama de Classes das fases 1 e 2

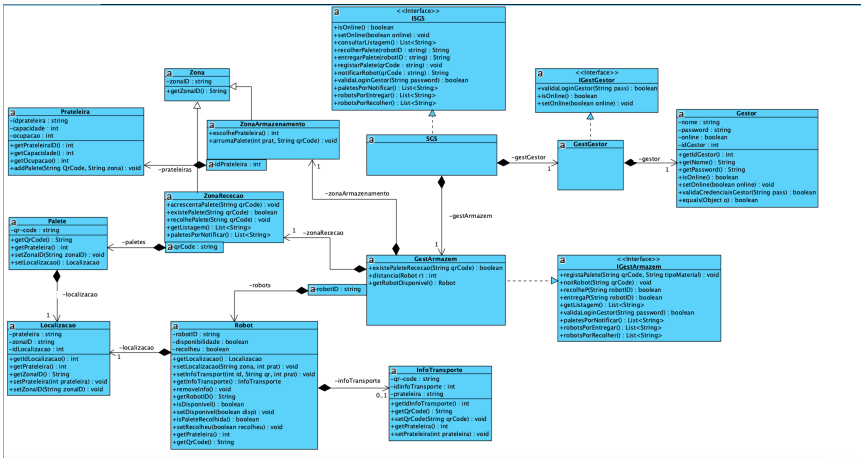

Figura 2 - Diagrama de Classes atual

Para além destas, também os Diagramas de Sequência sofreram alterações. Com os cinco use case apresentados para esta fase, foi necessária a realização de adaptações aos mesmos.

2.2 Diagramas de Sequência

2.2.1 Registar Palete

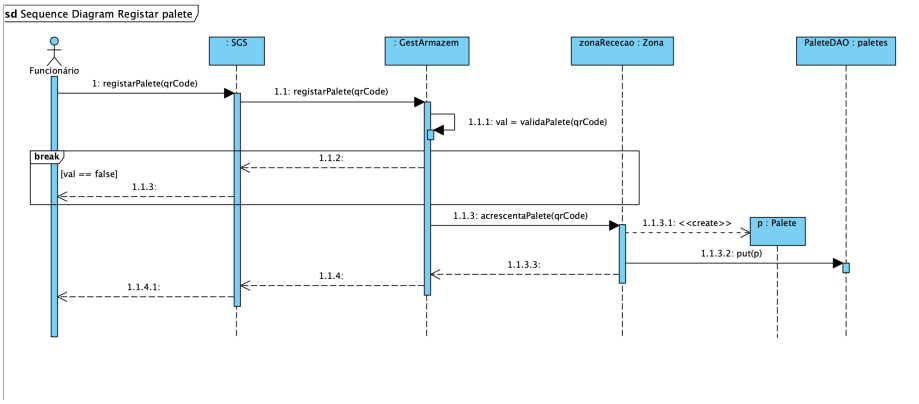

Figura 3 - Novo Diagrama de Sequencia de Registar Palete

2.2.2 Notificar Robot

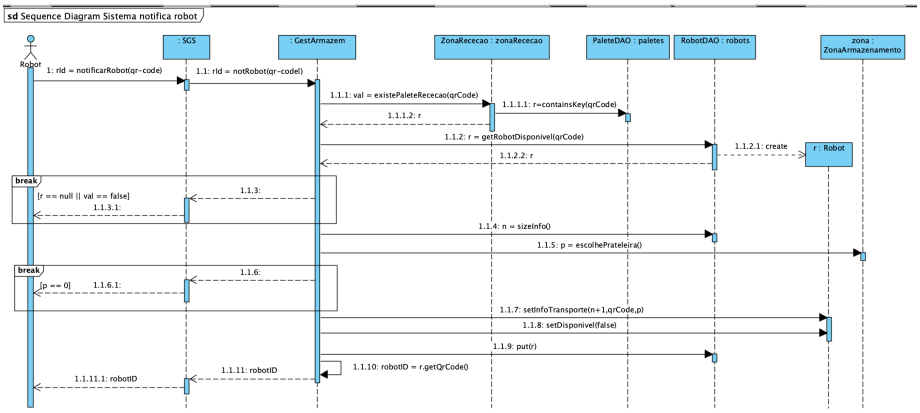

Figura 4 - Novo Diagrama de Sequencia de Notificar Robot

2.2.3 Recolher Palete

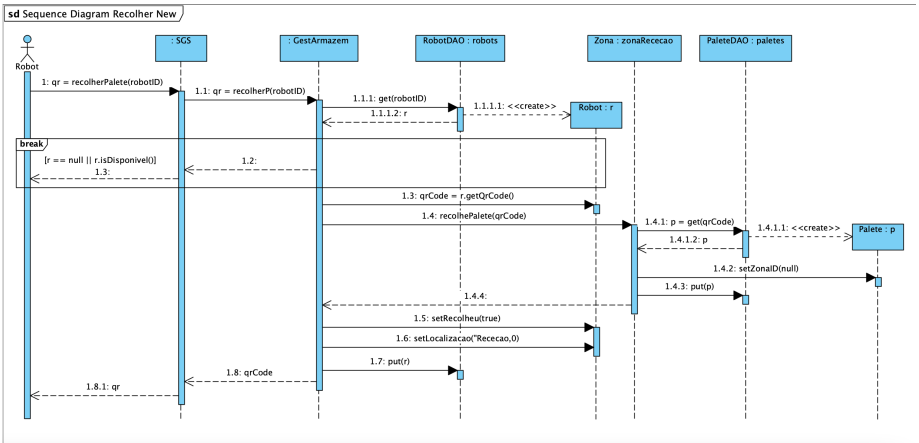

Figura 5 - Novo Diagrama de Sequencia de Recolher Palete

2.2.4 Entergar Palete

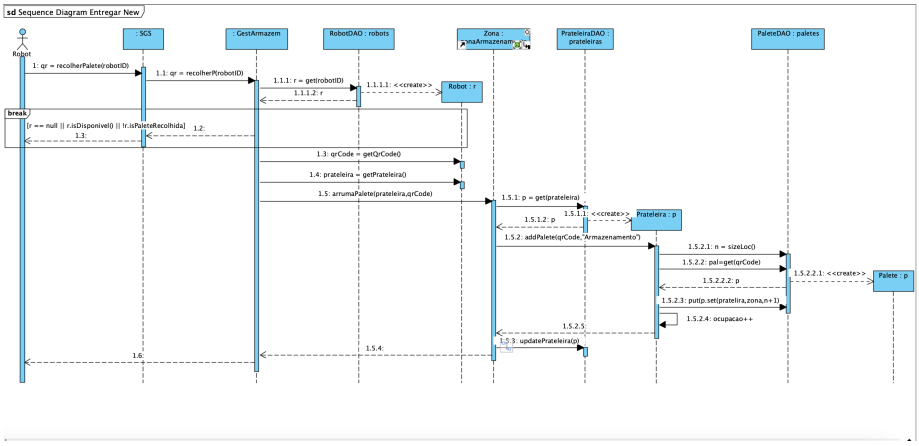

Figura 6 - Novo Diagrama de Sequencia de Entregar Palete

2.2.5 Solicitar Listagem

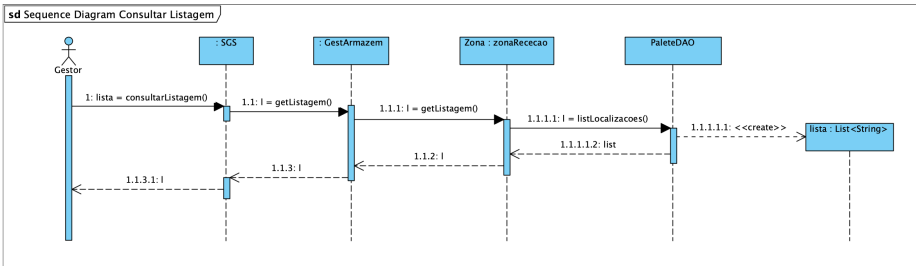

Figura 7 - Novo Diagrama de Sequencia de Solicitar Listagem

2.3 Diagrama de Packages

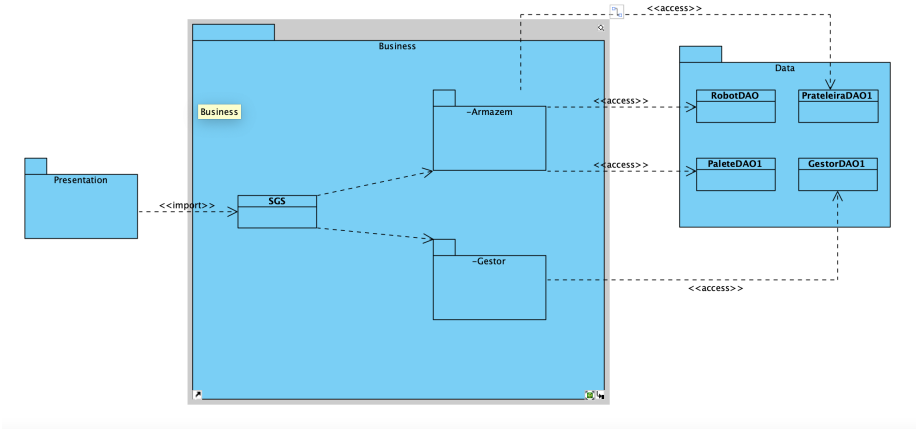

Figura 8 - Novo Diagrama de Packages

2.4 Diagrama de Componentes

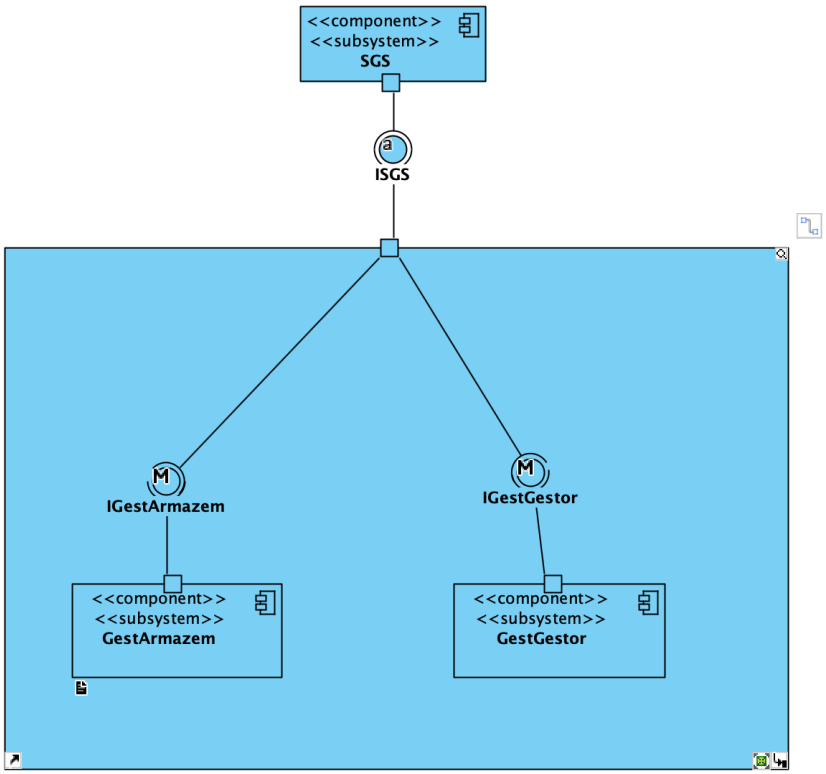

Figura 9 - Novo Diagrama de Componentes

3 Diagrama de ORM

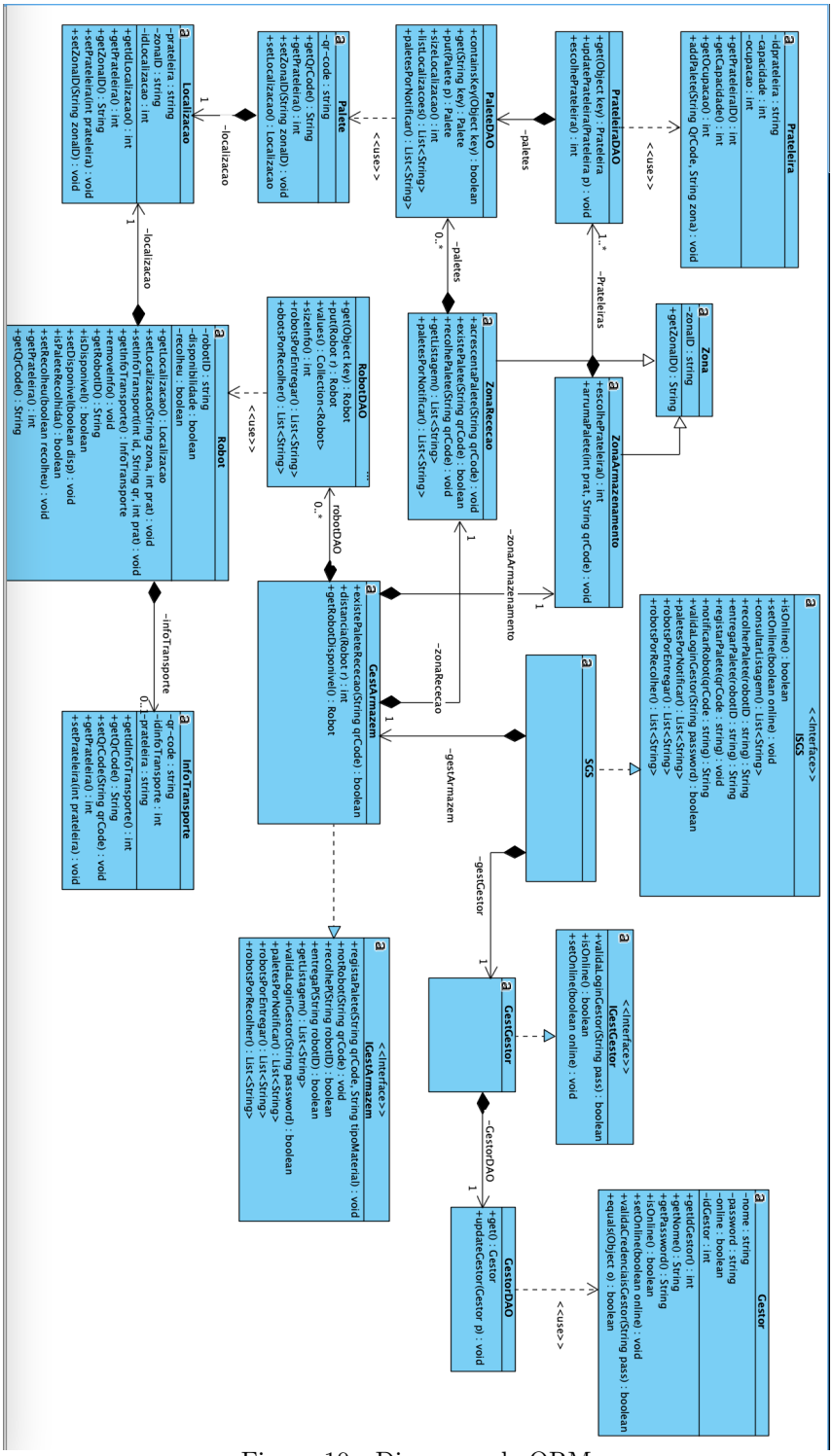

Figura 10 - Diagrama de ORM

4 Packages

4.1 Business

Dentro deste package, criamos mais dois subpackges, de modo a separar as classes relativas ao Armazem com as classes relativas ao Gestor.

4.1.1 ISGS

Interface do Sistema de Gestão de Stock

4.1.2 SGS

Nesta classe implementamos o Sistema da nossa aplicação. Esta classe tem como atributos a classe GestArmazem, responsável por todas as atividades do Armazém, e a classe GestGestor, responsável pelas atividades do Gestor.

4.1.3 IGestGestor

Interface do GestGestor.

4.1.4 GestGestor

Esta classe é responsável pela ponte entre o Gestor e o GestorDAO. Além disso, contém os métodos de validação das credenciais do Gestor.

4.1.5 Gestor

Nesta classe implementamos a entidade Gestor, que é a entidade responsável pelas credenciais da aplicação. Os atributos desta classe são o id e o Nome do Gestor, a palavra-pass da conta e o seu estado na aplicação (online / offline).

4.1.6 IGestArmazem

Interface do GestArmazem.

4.1.7 GestArmazem

Nesta classe implementamos a entidade GestArmazem. Esta funciona, de certa forma, como um gestor do armazém, ou seja, é responsável por todas as atividades deste. Tem atributos como a Zona de Receção, a Zona de Armazenamento e Robot, de modo a conseguir gerir e controlar as atividades do armazém.

Alguns dos métodos mais importantes nesta classe, tendo em conta a nossa aplicação, são os métodos *Robot getRobotDisponivel ();* e *int distancia(Robot r);*.

Este primeiro método, de modo a escolher um robot para realizar uma recolha, itera sobre todos os robots e verifica a sua disponibilidade e a sua distancia à zona de receção, escolhendo o Robot disponível mais perto desta.

Para calcular a distância usamos o segundo método referido em cima, seguindo as medidas da figura 4. Por exemplo, se um Robot estiver na Zona de Receção, a distância será 0 metros, mas se se encontrar na prateleira 8, a distância já será de 17 metros (5+5+7).

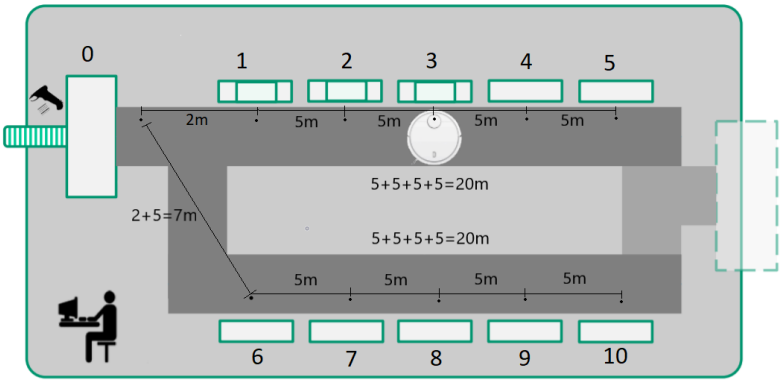

Figura 11 - Mapa do Armazém

4.1.8 Zona

Nesta classe implementamos a entidade Zona que nos permite a obtenção da sua localização através do seu ID (ZonaID). É também de realçar que a Zona é uma superclasse que engloba as duas subclasses ZonaArmazenamento e ZonaRececao.

4.1.9 ZonaArmazenamento

Nesta classe implementamos a entidade ZonaArmazenamento e todos os métodos associados à mesma, tais como o fornecimento da prateleira disponível para uma determinada palete e é reponsável pela devida arrumação na mesma (utilizando para isso um robot). É também de realçar que a ZonaArmazenamento é uma subclasse da Zona.

4.1.10 ZonaRececao

Nesta classe implementamos a entidade ZonaRececao e todos os métodos associados à mesma,tais como, a verificação da validade ou existência de uma palete. É responsável também pelo recolhimento de uma palete (utilizando para isso um robot), e por fornecer a listagem de todas as paletes. É também de realçar que a ZonaRececao é uma subclasse da Zona.

4.1.11 Prateleira

Nesta classe implementamos a entidade Prateleira e todos os métodos associados à mesma,tais como, a sua ocupação, capacidade e, consequentemente, o seu espaço livre. Também é possível a obtenção da sua identificação através do ID.

4.1.12 Palete

Nesta classe implementamos a entidade Palete e todos os métodos associados à mesma,tais como o fornecimento e alteração da sua localização, fornecimento da sua identificação (através do seu qr code), da sua localização, do tipo de material nela contido e repetiva prateleira, a zona onde esta se encontra, entre outros.

4.1.13 Robot

Nesta classe implementamos a entidade Robot e todos os métodos associados à mesma, tais como o fornecimento ou estabelecimento da sua localização e zona, fornecimento da informação de cada transporte por ele efetuado, a verificação da sua disponibilidade, entre outros.

4.1.14 InfoTransporte

Nesta classe implementamos a entidade Robot e todos os métodos associados à mesma, tais como o fornecimento ou alteração do ID de uma determinada informação de transporte, fornecimento do qr code da palete a ser transportada e da prateleira destino dessa palete.

4.1.15 Localizacao

Nesta classe implementamos a entidade Localização e todos os métodos associados à mesma, tais como o fornecimento do ID da localização e sua respetiva alteração. É também responsável por indicar a prateleira e o ID de uma determinada Zona.

4.2 Data

Neste package temos os nossos DAO's. Estes permitem nos fazer a ponte de ligação entre a base de dados e a nossa aplicação.

Inicialmente, iamos por os DAO's a implementar a interface Map, de modo a facilitar a sua implementação. Porém, olhando para o contexto da nossa aplicação, percebemos que seria mais eficiente se criássemos nós os metodos que nos fossem mais convenientes nos DAO's, logo, não utilizamos essa implementação.

4.2.1 DAOconfig

Nesta classe, temos as configurações necessárias para realizar o acesso à Base de Dados: username, password, nome da base de dados e o driver.

4.2.2 GestorDAO

Nesta classe implementamos os seguintes métodos:

- *Gestor get();*
- *void updateGestor(Gestor p);*

Deste modo, podemos ir buscar e atualizar as informações do Gestor da Aplicação à Base de Dados.

4.2.3 PaleteDAO

Nesta classe implementamos os seguintes métodos:

- *Palete containskey(Obejct key);*
- *Palete get(String key);*
- *Palete put(Palete p);*
- *int sizeLocalizacao();*
- *List<String> listLocalizacoes();*
- *List<String> paletesPorNotificar();*

O método *List<String> listLocalizacoes();* permite-nos obter a listagem das localizações das paletes.

Já o método *List<String> paletesPorNotificar();* permite-nos obter as paletes que estão na zona de receção à espera para serem transportadas, para que o utilizador da aplicação consiga saber quais são essas paletes.

4.2.4 PrateleiraDAO

Nesta classe implementamos os seguintes métodos:

- *Prateleira get(Object key);*
- *void updatePrateleira(Prateleira p);*
- *int escolhePrateleira();*

O método *int escolhePrateleira();* retira da base de dados a prateleira não cheia cuja ocupação é menor. Deste modo, as paletes serão sempre igualmente distribuidas pelas prateleiras.

4.2.5 RobotDAO

- *Robot get(Object key);*
- *Robot put(Robot r);*
- *Collection<Robot> values();*
- *int sizeInfo();*
- *List<String> robotsPorRecolher();*
- *List<String> robotsPorEntregar();*

O método *List<String> robotsPorRecolher();* permite-nos criar uma lista com os robots que estão prontos a fazer uma recolha, de modo a apresentá-los ao utilizador da aplicação. Já o método *List<String> robotsPorEntregar();* faz a mesma coisa, mas para os robots que estão prontos a fazer uma entrega.

4.3 Presentation

Este package contém o controllador e a view da nossa aplicação.

4.3.1 TextUI

Esta classe interliga o SGS com o vista do programa.

4.3.2 Menu

Nesta classe temos todas as funções que recebem input e devolvem output para o utilizador.

```
----------------------------
 Sistema de Gestao de Stock
----------------------------
 1 | Registar Palete
 2 | Notificar Robot
 3 | Recolher Palete
 4 | Entregar Palete
 5 | Consultar Listagem
 0 | Sair
----------------------------
Opção:
```

Figura 12 - Menu Inicial

4.4 Exceptions

Neste package temos todas as exceptions desenvolvidas para a nossa aplicação. As exceptions que criamos foram:

- *ArmazemCheioException*
- *ListaPaletesPorNotificarException*
- *ListaRobotsPorEntregarException*
- *ListaRobotsPorRecolherException*
- *ListagemVaziaException*
- *PaleteInvalidaException*
- *PaleteNaoExisteException*
- *PaleteNaoRecolhidaException*
- *RobotNaoDisponivelException*

5 Base de Dados

5.1 Modelo Lógico

No nosso projeto decidimos utilizar o MySql como sistema de Base de Dados, pois é o sistema que o grupo utilizou durante as aulas.

Assim, utilizamos o MySqlWorkbench para criar o modelo lógico da nossa base de dados.

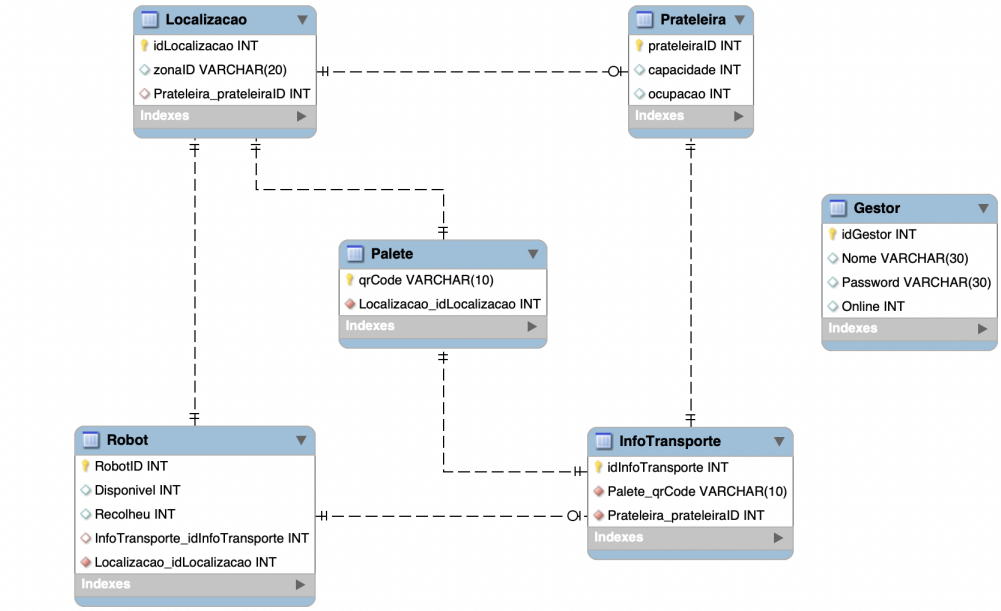

Figura 13 - Modelo Lógico

5.2 Modelo Físico

Tendo por base o modelo lógico construido, decidimos utilizar a funcionalidade "Forward Engineer" da plataforma MySqlWorkbench, de modo a gerar o modelo físico.

Desta forma, criamos um script com o modelo físico de modo a conseguirmos formular as tabelas da base de dados.

```
CREATE TABLE IF NOT EXISTS `DSS_Project`.`Localizacao` (
  `idLocalizacao` INT NOT NULL,
  `zonaID` VARCHAR(20) NULL,
  `Prateleira_prateleiraID` INT NULL,
  PRIMARY KEY (`idLocalizacao`),
  INDEX `fk_Localizacao_Prateleira1_idx` (`Prateleira_prateleiraID` ASC) VISIBLE,
  CONSTRAINT `fk_Localizacao_Prateleira1`
    FOREIGN KEY (`Prateleira_prateleiraID`)
    REFERENCES `DSS_Project`.`Prateleira` (`prateleiraID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Figura 14 - Implementação Física da Tabela Localizacao

```
CREATE TABLE IF NOT EXISTS `DSS_Project`.`Prateleira` (
  `prateleiraID` INT NOT NULL,
  `capacidade` INT NULL,
  `ocupacao` INT NULL,
  PRIMARY KEY (`prateleiraID`))
ENGINE = InnoDB;
```

Figura 15 - Implementação Física da Tabela Prateleira

```
CREATE TABLE IF NOT EXISTS `DSS_Project`.`Palete` (
  `qrCode` VARCHAR(10) NOT NULL,
  `Localizacao_idLocalizacao` INT NOT NULL,
  PRIMARY KEY (`qrCode`),
  INDEX `fk_Palete_Localizacao1_idx` (`Localizacao_idLocalizacao` ASC) VISIBLE,
  CONSTRAINT `fk_Palete_Localizacao1`
    FOREIGN KEY (`Localizacao_idLocalizacao`)
    REFERENCES `DSS_Project`.`Localizacao` (`idLocalizacao`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Figura 16 - Implementação Física da Tabela Palete

```
CREATE TABLE IF NOT EXISTS `DSS_Project`.`InfoTransporte` (
  `idInfoTransporte` INT NOT NULL,
  `Palete_qrCode` VARCHAR(10) NOT NULL,
  `Prateleira_prateleiraID` INT NOT NULL,
  PRIMARY KEY (`idInfoTransporte`),
  INDEX `fk_InfoTransporte_Palete1_idx` (`Palete_qrCode` ASC) VISIBLE,
  INDEX `fk_InfoTransporte_Prateleira1_idx` (`Prateleira_prateleiraID` ASC) VISIBLE,
  CONSTRAINT `fk_InfoTransporte_Palete1`
    FOREIGN KEY (`Palete_qrCode`)
    REFERENCES `DSS_Project`.`Palete` (`qrCode`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_InfoTransporte_Prateleira1`
    FOREIGN KEY (`Prateleira_prateleiraID`)
    REFERENCES `DSS_Project`.`Prateleira` (`prateleiraID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Figura 17 - Implementação Física da Tabela InfoTransporte

```
CREATE TABLE IF NOT EXISTS `DSS_Project`.`Robot` (
  `RobotID` INT NOT NULL,
  `Disponivel` INT NULL,
  `Recolheu` INT NULL,
  `InfoTransporte_idInfoTransporte` INT NULL,
  `Localizacao_idLocalizacao` INT NOT NULL,
  PRIMARY KEY (`RobotID`),
  INDEX `fk_Robot_InfoTransporte1_idx` (`InfoTransporte_idInfoTransporte` ASC) VISIBLE,
  INDEX `fk_Robot_Localizacao1_idx` (`Localizacao_idLocalizacao` ASC) VISIBLE,
  CONSTRAINT `fk_Robot_InfoTransporte1`
    FOREIGN KEY (`InfoTransporte_idInfoTransporte`)
    REFERENCES `DSS_Project`.`InfoTransporte` (`idInfoTransporte`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Robot_Localizacao1`
    FOREIGN KEY (`Localizacao_idLocalizacao`)
    REFERENCES `DSS_Project`.`Localizacao` (`idLocalizacao`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Figura 18 - Implementação Física da Tabela Robot

```
CREATE TABLE IF NOT EXISTS `DSS_Project`.`Gestor` (
  `idGestor` INT NOT NULL,
  `Nome` VARCHAR(30) NULL,
  `Password` VARCHAR(30) NULL,
  `Online` INT NULL,
  PRIMARY KEY (`idGestor`))
ENGINE = InnoDB;
```

Figura 19 - Implementação Física da Tabela Gestor

5.3 Povoamento da Base de Dados

Para povoar, inicialmente, a Base de Dados, optamos por criar um script de povoamento. Através deste, preenchemos todas as tabelas com alguma informação, de modo ao programa estar pronto a utilizar desde início.

```
USE DSS_Project;

INSERT INTO Prateleira VALUES
    (1, 5, 1), (2, 5, 0), (3, 5, 0),
    (4, 5, 2), (5, 5, 1), (6, 5, 0),
    (7, 5, 1), (8, 5, 0), (9, 5, 0),
    (10, 5, 0);

INSERT INTO Localizacao VALUES
    (1, "Armazenamento", 1), (2, "Armazenamento", 4),
    (3, "Armazenamento", 4), (4, "Armazenamento", 1),
    (5, "Rececao", null), (6, "Armazenamento", 4),
    (7, "Armazenamento", 5), (8, "Armazenamento", 6);

INSERT INTO Palete VALUES
    ('a1', 1), ('a2', 2), ('a3', 3), ('a4', 4),
    ('a5', 5), ('a6', 6), ('a7', 7);

INSERT INTO InfoTransporte VALUES (1, 'a6', 2);

INSERT INTO Robot VALUES
    (1, 0, 0, 1, 6), (2, 1, 0, null, 7),
    (3, 1, 0, null, 8);

INSERT INTO Gestor VALUES ('1', 'Paulo Sousa', 'root', 0);
```

Figura 20 - Povoamento da Base de Dados

6 Conclusão e Análise de Resultados

A realização deste trabalho teve várias particularidades que se revelaram desafios interessantes e extremamente didáticos, apesar de não estarmos habituados a trabalhar desta maneira, demonstrou-se extremamente eficiente na maioria dos casos. Desta forma, podemos afirmar com segurança, que as fases 1 e 2 nos ajudaram bastante não só planeamento e organização do projeto, mas também na implementação do mesmo.

Foi, também, desafiante o facto de a terceira fase do projeto apenas englobar uma parte do sistema inicialmente imaginado e planificado por nós, no sentido em que apesar de querermos implementar algumas funcionalidades, estas já iam para além do processo requerido.

6.1 Evolução de um Use Case

O Use Case que escolhemos para demonstrar a evolução ao longo de todo o projeto foi o Use Case **Registar Paletes** (Comunicar Código QR, de acordo com o enunciado).

Na fase inicial do projeto (Fase 1), levantamos os requisitos necessários para a eleboração deste Use Case, ou seja, do que seria necessário este Use Case realizar. Desta forma, criamos uma tabela na qual aplicamos estes requisitos sequencialmente, esta tabela ainda suporta os vários fluxos possíveis para o Use Case.

Use Case:	Registar paletes	
Ator:	Funcionário	
Pré condição:	True	
Pós condição:	Sistema regista as paletes	
Fluxo normal:	Input do ator:	Resposta do sistema:
	1. Lê QR-code	
		2. Valida o QR-code
		3. Regista a palete
Fluxo de exceção [QR-code inválido] (passo 2):	Input do ator:	Resposta do sistema:
		2.1 Rejeita as paletes

Figura 21 - Usecase Registar Palete

Na segunda fase do projeto, avançamos na preparação para a implementação do Use Case **Registar Paletes** através da eleboração de um diagrama de sequência para este Use Case. Este novo diagrama permite uma maior aproximação à implementação real, através da idealização de métodos por exemplo, facilitando a realização da fase 3.

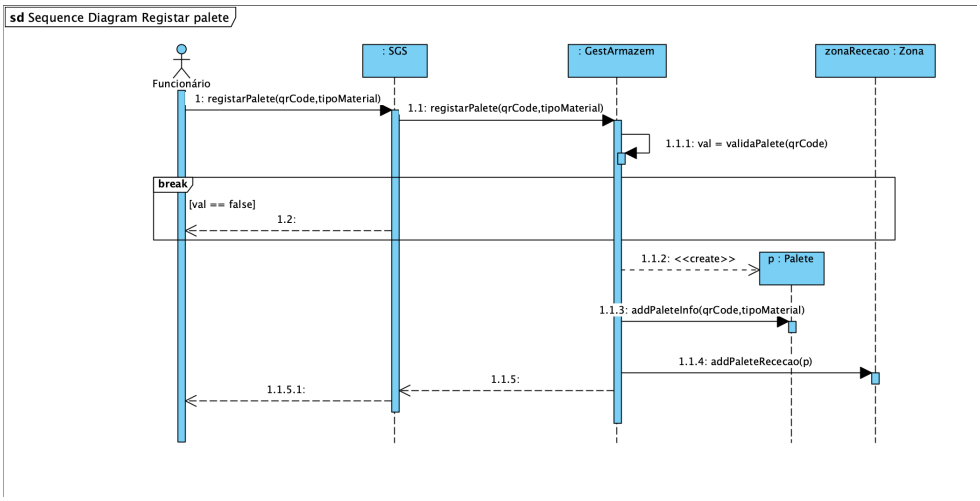

Figura 22 - Diagrama de Sequência Registar Palete

Por último, na Fase 3, implementamos fisicamente o Use Case, apoiando-nos nas bases criadas pelos diagramas elaborados anteriormente.

Concluindo, a realização de todos os diagramas (Classes, Use Case, Sequência, Packages e Componentes), planeamento estruturado e avanço progressivo, facilita a implementação física do projeto, poupando tempo e esforço de planeamento e até mesmo de correção de erros.