National College of Ireland

SMART WAREHOUSE APPLICATION

Distributed Systems
2023

Mohammad Asem Soufi
Student nr: x18191665

Contents

1 Introduction

The purpose of the application is to create a smart warehouse that can track stock and automate orders.
The system will consist of smart services that inter-communicate with each other via gRPC.
The warehouse will be equipped with services that monitor the stock levels and that can automate the ordering process.
The overall goal is to reduce the workload of the warehouse staff and to minimize errors in stock management.

In addition, the application keeps records of existing Employees and offers the possibility to add new ones.

2 Service 1: Stock Management Service (stockService)

This service will be responsible for managing the stock levels of all products in the warehouse.

2.1 Methods

2.1.1 RPC Method 1 – updateQty (Bidirectional Stream)

This method will allow the client to update the stock level of multiple items at the same time, for example when placing an order. The client will send a stream of UpdateQtyRequest messages, each containing the item code and the amount by which it needs to be updated with, which could be either Positive or Negative. The server will respond with a stream of UpdateQtyResponse messages with 'bool' response that indicates success/failure, and a relevant 'string' message to provide further information.

```
rpc updateQty(stream UpdateQtyRequest) returns (stream
UpdateQtyResponse){}

  message UpdateQtyRequest {
    int32 stockNumber = 1;
    sint32 qty = 2; // could be a negative number if
  decreasing quantity
  }

  message UpdateQtyResponse {
    bool success = 1;
    string message = 2;
  }
```

2.1.2 RPC Method 2 - getProduct (Server Stream)

This method will allow the client to retrieve the current stock level of a particular or All products. The request ProductRequest will include the product code for a specific product, or an empty code to get a stream of all products in stock. The response ProductResponse will include all product/s details.

```
rpc getProduct(ProductRequest) returns (stream
ProductResponse) {}
```

```
message ProductRequest {
  int32 stockNumber = 1;
}

message ProductResponse {
  int32 stockNumber = 1;
  string description = 2;
  float price = 3;
  int32 qty = 4;
}
```

2.1.3 RPC Method 3 - addProduct (Unary)

This method will allow the client to add a new product to stock. The client will send a AddProductRequest message containing all product details needed to create a new product, the server will respond with a AddProductResponse message with 'bool' response that indicates success/failure, and a relevant 'string' message to provide further information.

```
rpc addProduct(AddProductRequest) returns
(AddProductResponse) {}
```

```
message AddProductRequest {
  int32 stockNumber = 1;
  string description = 2;
  float price = 3;
  int32 qty = 4;
}

message AddProductResponse {
  bool success = 1;
  string message = 2;
}
```

3 Service 2: Order Management Service (orderService)

This service will be responsible for managing the ordering process in the warehouse.

3.1 Methods

3.1.1 RPC Method 1 – placeOrder (Client Stream)

This method will allow the client to create a new order. The request PlaceOrderRequest will include the list of products to be add to the order by providing a stream of products details and the required quantity of each product, the Order Server will build and add the new order, and the response getOrderResponse will include the full details of the new order (as a string).

```
rpc placeOrder(stream PlaceOrderRequest) returns
(GetOrderResponse) {}
```

```
message PlaceOrderRequest {
  int32 stockNumber = 1;
  string prodDescription = 2;
  float prodPrice = 3;
  int32 prodQty = 4;
  int32 orderedQty = 5;
}

message GetOrderResponse {
  string orderDetails = 1;
}
```

3.1.2 RPC Method 2 – cancelOrder (Unary)

This method will allow the client to cancel an existing order. The request CancelOrderRequest will include the order number that we're trying to cancel, and the response CancelOrderResponse will be a 'bool' response.

```
rpc cancelOrder(CancelOrderRequest) returns
(CancelOrderResponse) {}
```

```
message CancelOrderRequest {
  int32 orderNumber = 1;
}

message CancelOrderResponse {
  bool success = 1;
}
```

3.1.3 RPC Method 3 – getOrderDetails (Server Stream)

This method will allow the client to request the details of a single order by making a request GetOrderRequest that has the order number, or an empty request to get a full list of all orders in the system, and the server will respond GetOrderResponse with a single reply of that specific order, or a stream of all orders.

```
rpc getOrderDetails(GetOrderRequest) returns (stream
GetOrderResponse) {}

 message GetOrderRequest {
   int32 orderNumber = 1;
 }

 message GetOrderResponse {
   string orderDetails = 1;
 }
```

3.1.4 RPC Method 4 – getTotalSales (Unary)

This method will allow the client to request (TotalSalesRequest) the total value of sales of all orders in the system, no variables needed.

The server will respond (TotalSalesResponse) with a float number of Total Sales.

```
rpc getTotalSales(TotalSalesRequest) returns
(TotalSalesResponse) {}

 message TotalSalesRequest{}

 message TotalSalesResponse{
   float salesTotal = 1;
 }
```

4 Service 3: Employee Management Service (employeeService)

This service will be responsible for managing employees in the warehouse.

4.1 Methods

4.1.1 RPC Method 1 – addEmployee (Unary)

This method will allow the client to assign an employee to the warehouse.

```
rpc addEmployee (AddEmployeeRequest) returns
(AddEmployeeResponse) {}
```

```
message AddEmployeeRequest {
  int32 employeeNumber = 1;
  string employeeName = 2;
  string position = 3;
  int32 salary = 4;
}

message AddEmployeeResponse {
  bool success = 1;
  string message = 2;
}
```

4.1.2 RPC Method 2 – getEmployee (Unary)

This method will allow the client to get (GetEmployeesRequest) the details of a specific employee based on his number, and the server replies with a GetEmployeesResponse with that employee's details (string).

```
rpc getEmployee (GetEmployeeRequest) returns
(GetEmployeeResponse) {}

message GetEmployeeRequest {
  int32 employeeNumber = 1;
}

message GetEmployeeResponse {
  string employeeDetails = 1;
}
```

4.1.3 RPC Method 3 – getAllEmployees (Server Stream)

Allows client to request (GetAllEmployeesRequest) all employees details in the system. The server replies (GetEmployeesResponse) with a stream of (string) details of each emplyee.

```
rpc getAllEmployees (GetAllEmployeesRequest) returns
(stream GetEmployeeResponse) {}
```

```
message GetAllEmployeesRequest {}
```

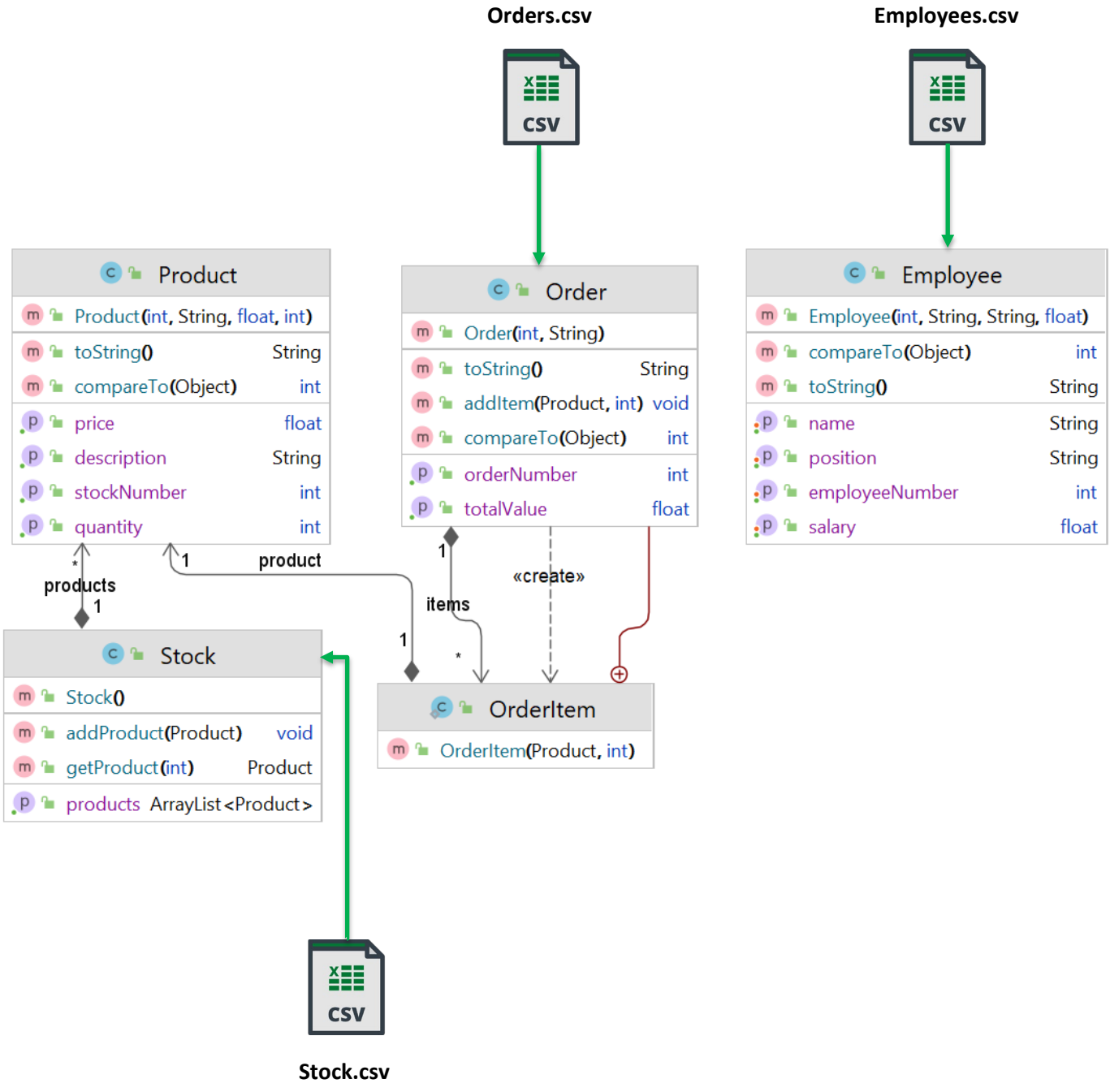
```
message AddEmployeeResponse {
  bool success = 1;
  string message = 2;
}
```

5 Smart Warehouse Components

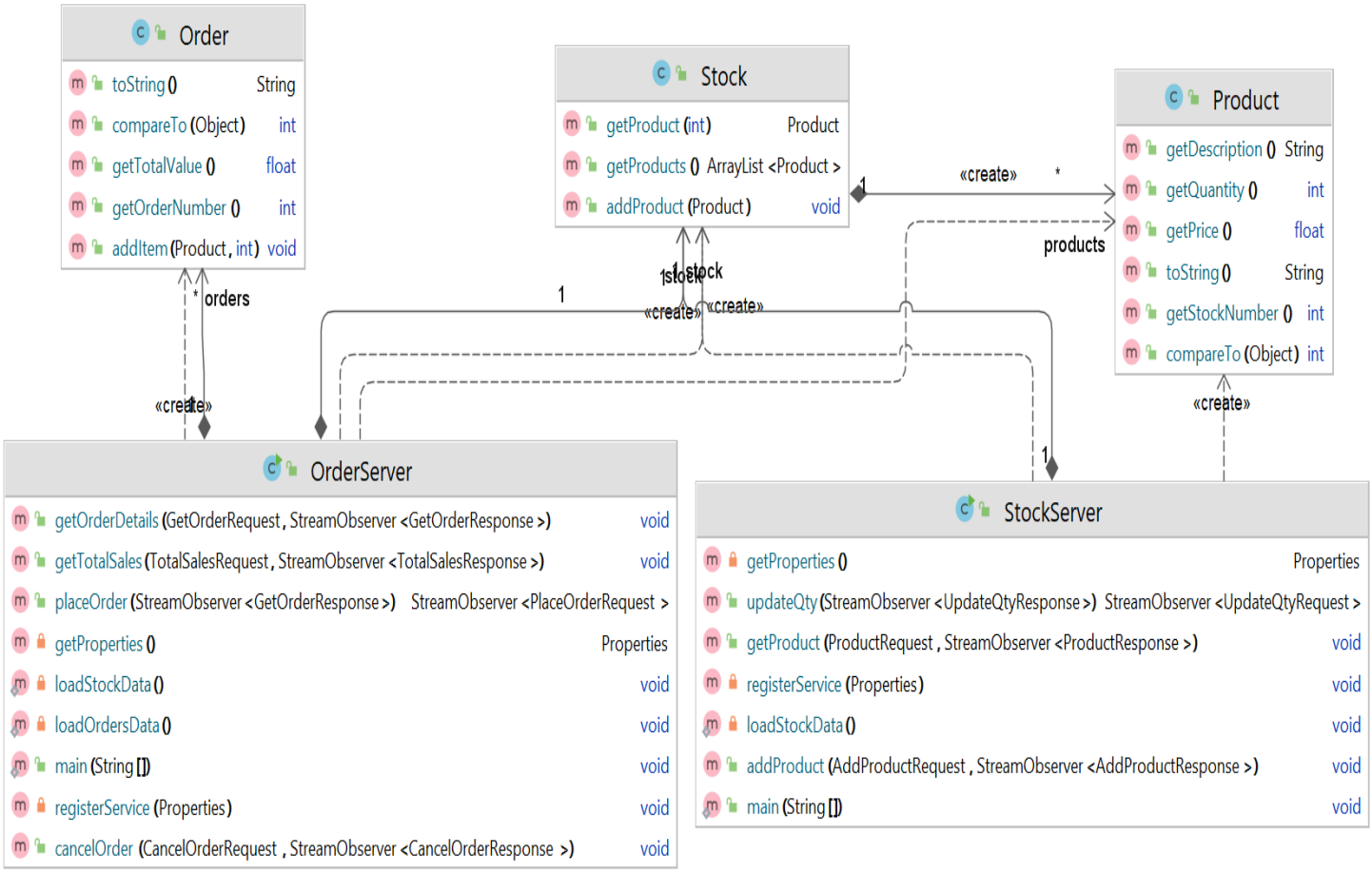
The warehouse itself is designed with the a set of entities that serve as the base for all data objects and sub-information needed to provided services. Mainly:

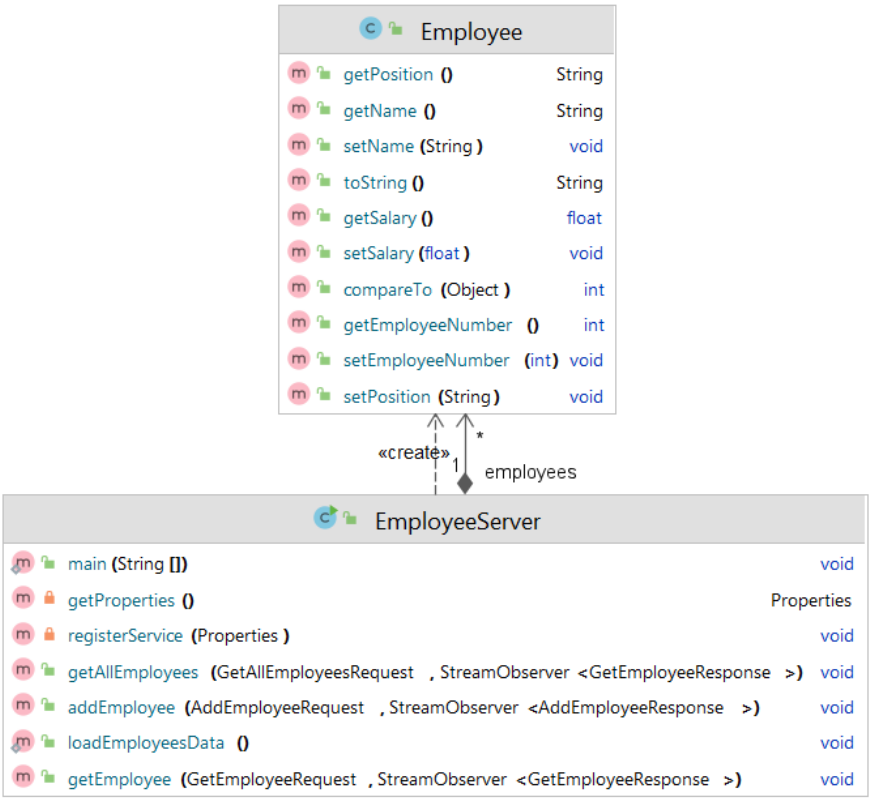
- Employee
- Product
- Stock
- Order

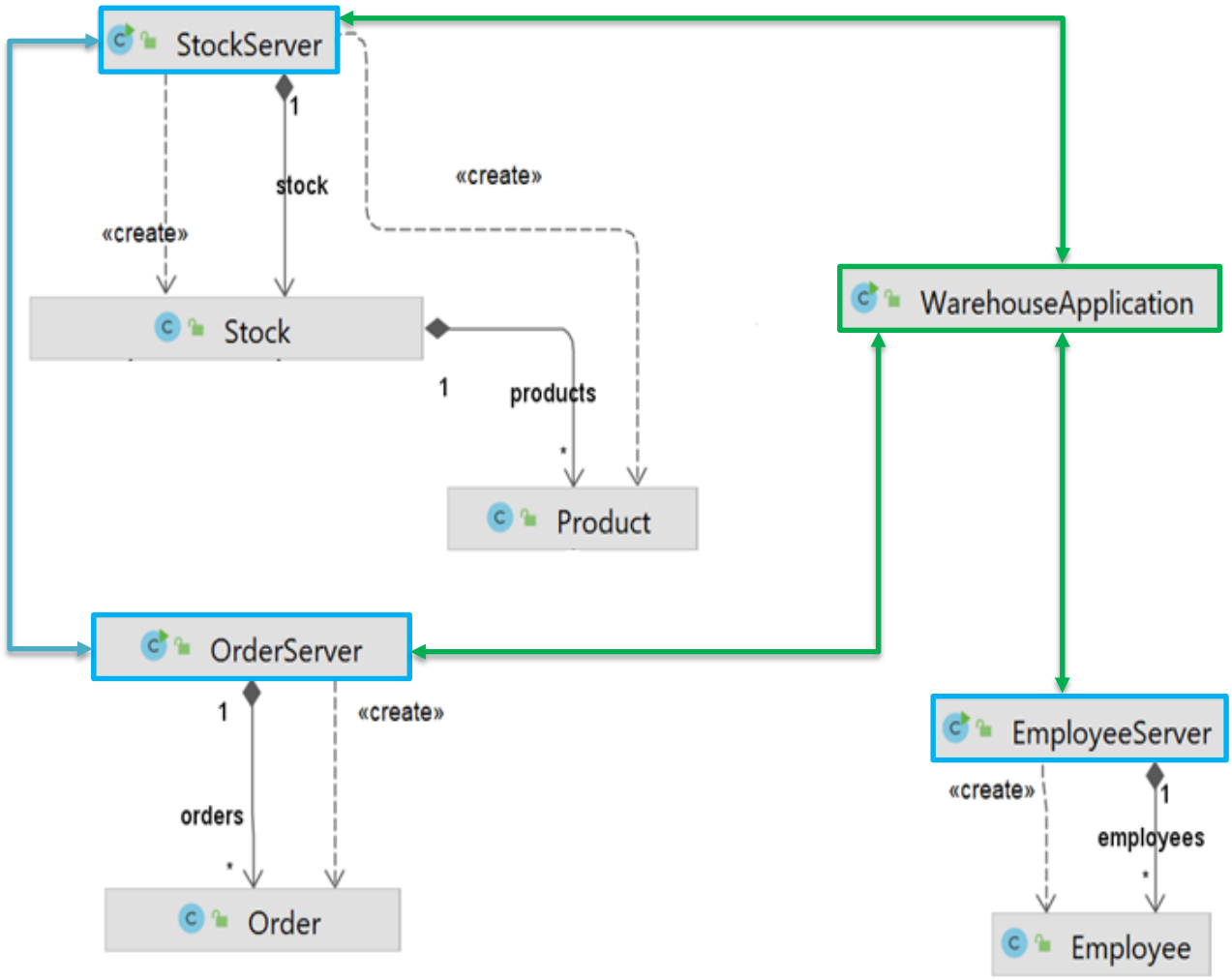
Below is a high level class diagram ro illustrate this:



6 Messages communications







7 GitHub

https://github.com/asemsoufi/NCIDistSysProject

8 Recording

Recording-20230506_210546.webm