Pierre-Émile Brassard

Repo : https://github.com/ThePhobosDeimosSuite/laboratoire-0_log430/tree/lab2

Projet is available in production here : log430@10.194.32.176

Run `docker compose run --rm shop` in projet folder which is located at `~/lab0`

A look back at lab0/lab1

After reading the requirement for lab2, i don't think a REST API is a good idea. First thing I'll do is to port everything to an MVC architecture with *terminal-kit* and then work on implementing the features for lab2. At least i won't have to change anything regarding the controller and database.

Introduction and Goals

Requirements Overview

A small business composed of 5 different shops, a supply center and a main shop needs an application to operate. As such, the app needs to keep track of sales and the inventory for each shop on top of a wide array of product. Employees will be able to create sales and a manager can access all of those sales in the shape of a sales report.

Quality Goals

Data needs to be saved and shared amongst all the different users. An interface in also required to allow users to read and input data. The service has to be hosted on a VM.

Stakeholders

Name	Expectations
Manger	Create sales report, view store performance on a dashboard, create and update product.
Supply center employee	Deliver new stock to a store, see current stocks.
Store employee	Search product, create sales, search sales, cancel sales, see stocks, order more stocks;

System Scope and Context

Business Context

The app has to execute 10 different use cases, each being initiated through the interface. Each use case can be found in a menu grouped by each type of stakeholder.

Technical Context

The app will communicate with a *postgresql* database. Users will use the command line to read and input data to the system.

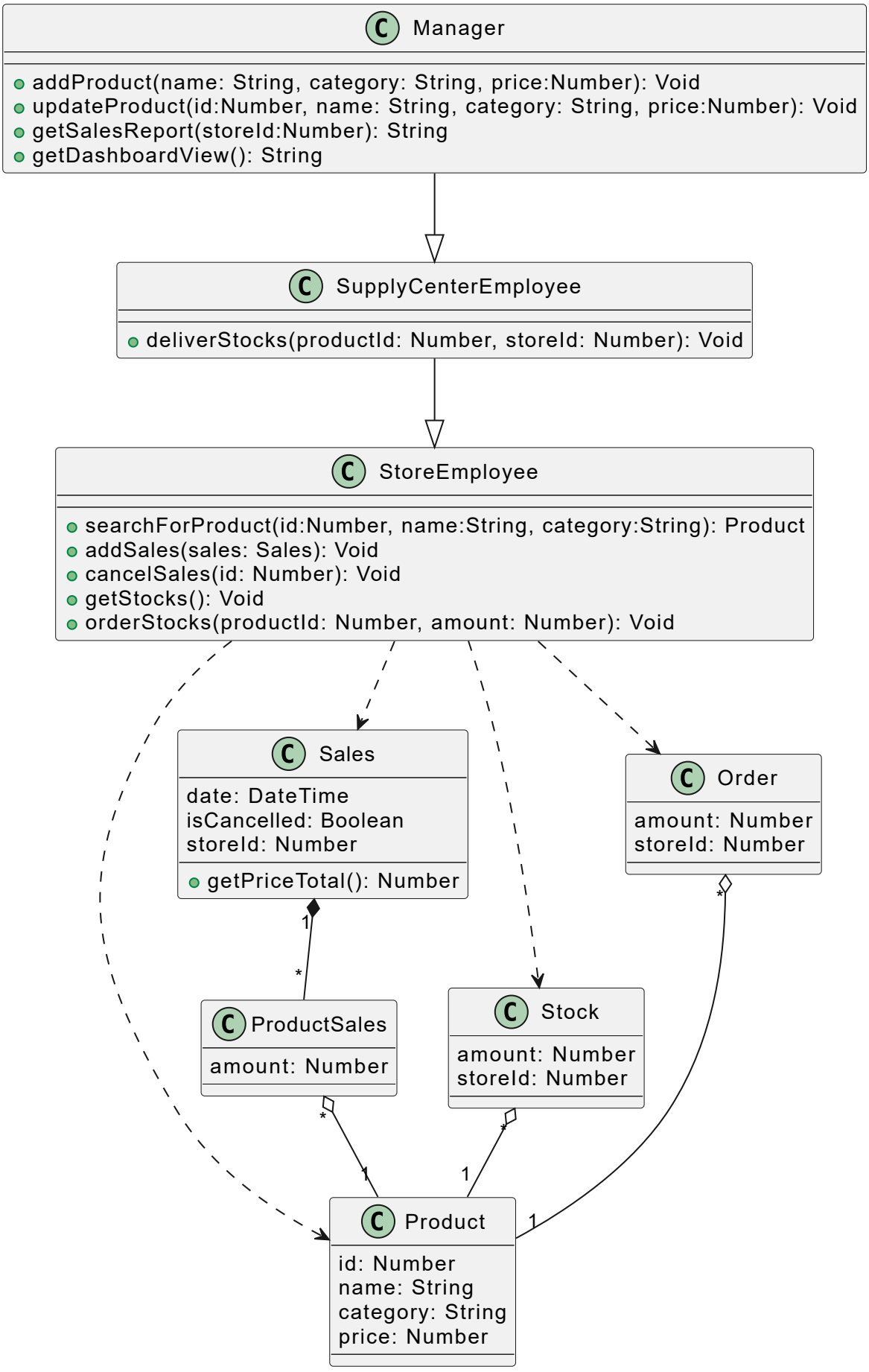
Solution Strategy

Persistance will be managed by a single database shared between all shops, supply center and main shop. Interface will consist of a simple command line interface with each user having their own menu tailored to their needs.

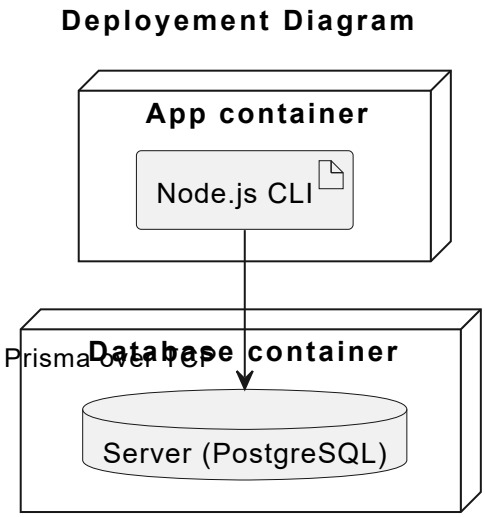
UML

Class Diagram

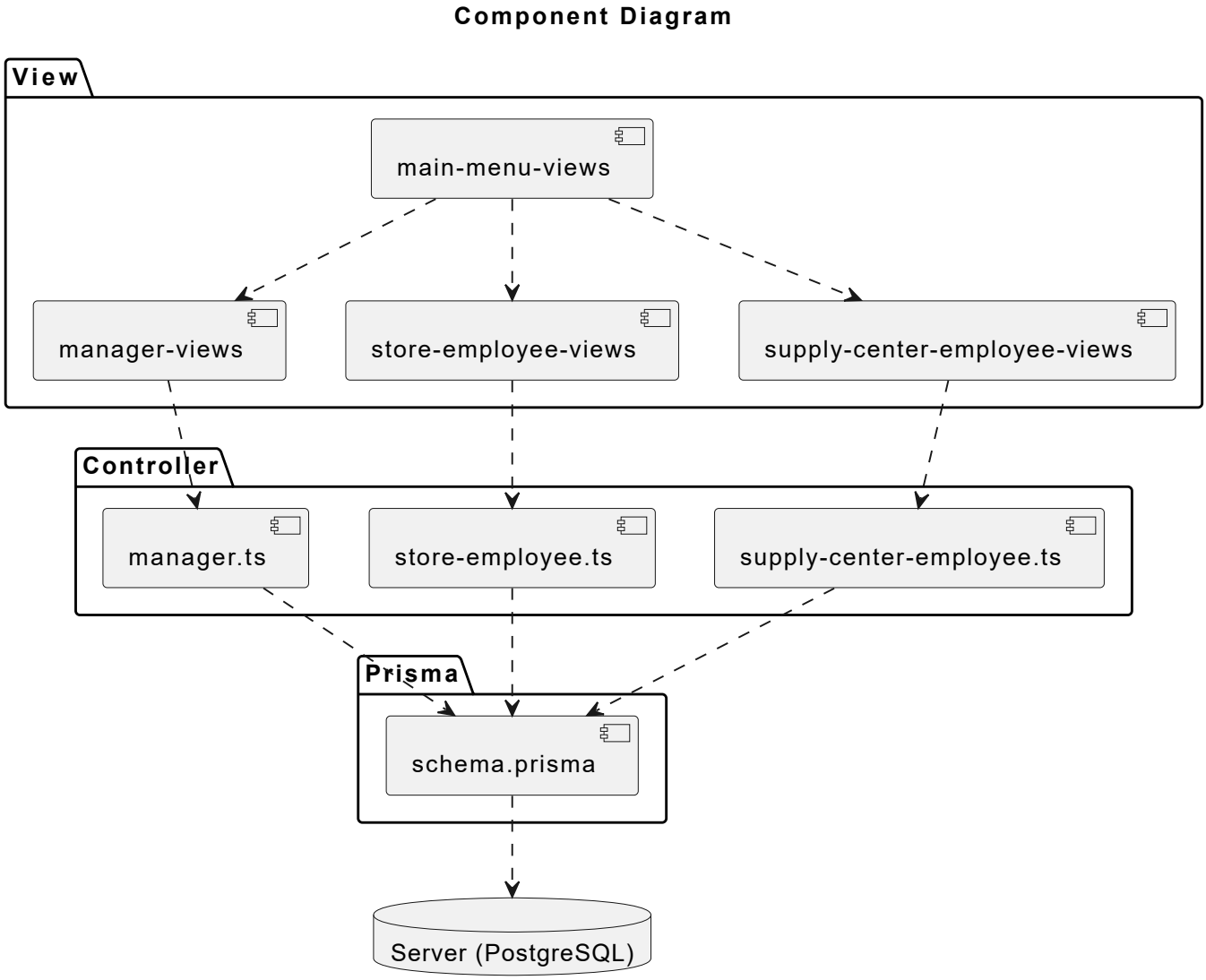
Class Diagram



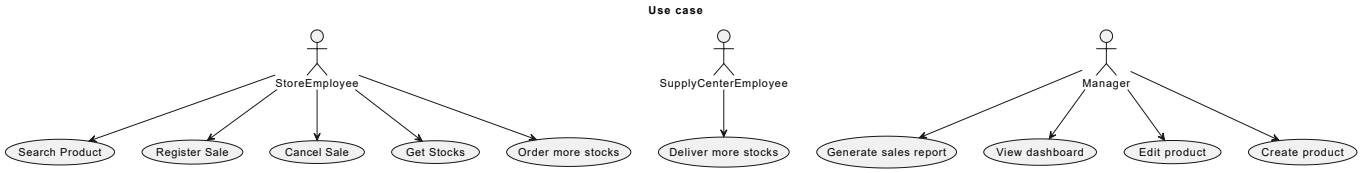
Deployement view



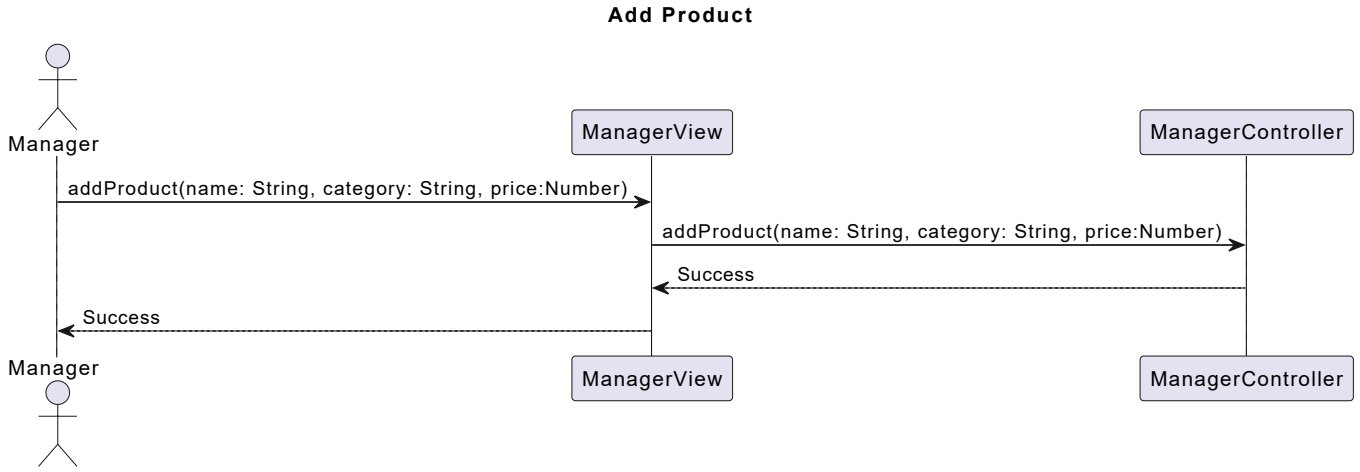
Component view



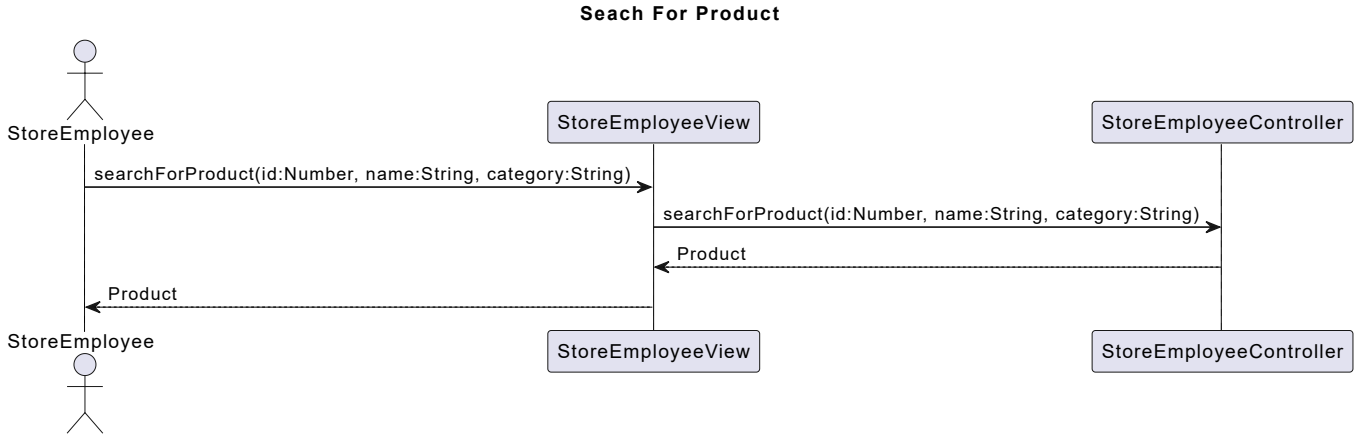
Use case



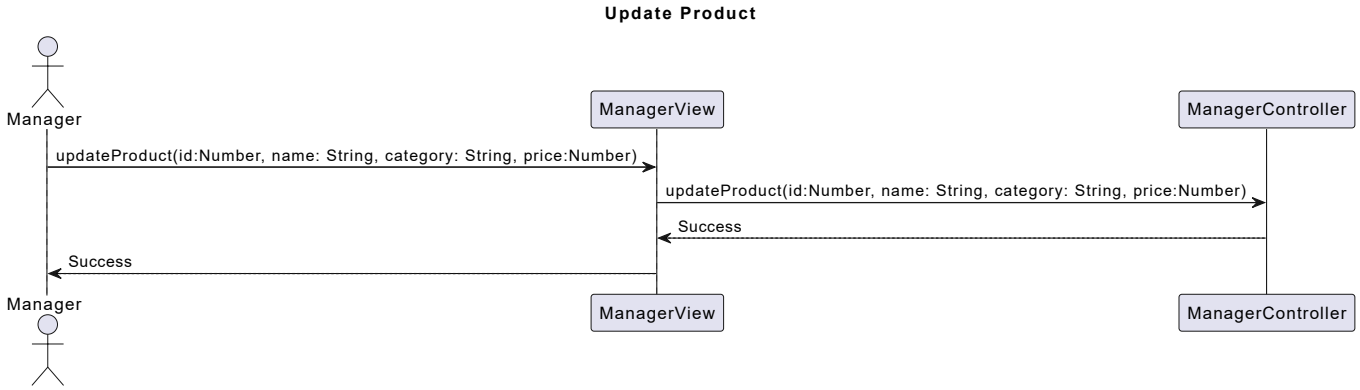
Add product



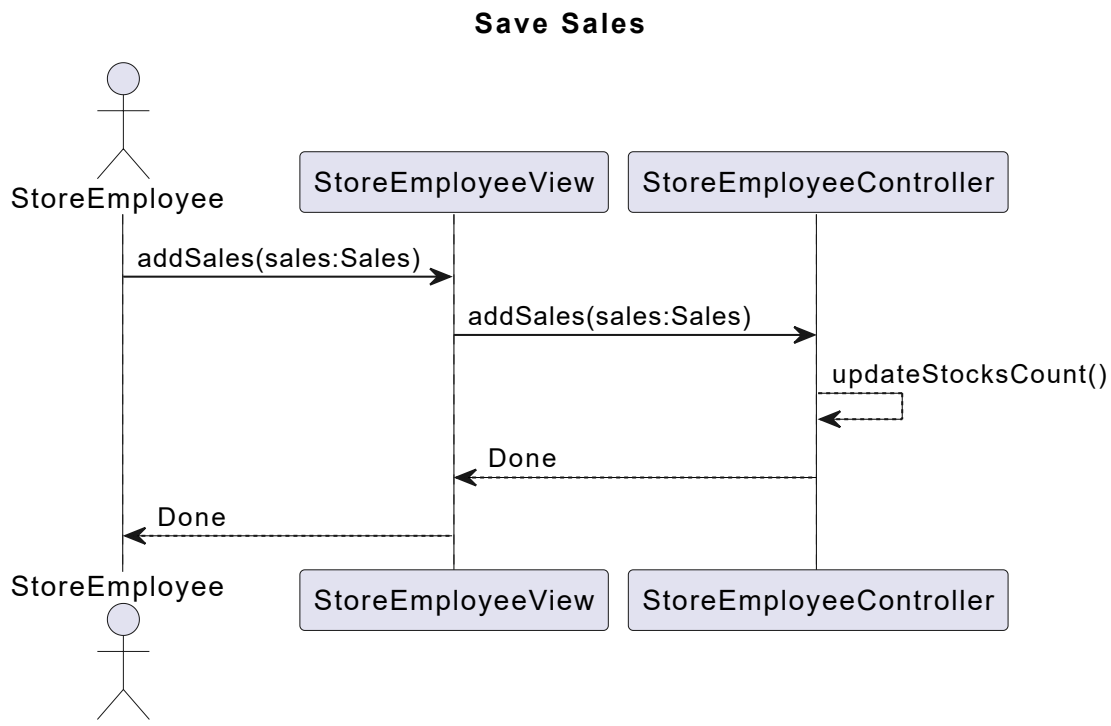
Search for product



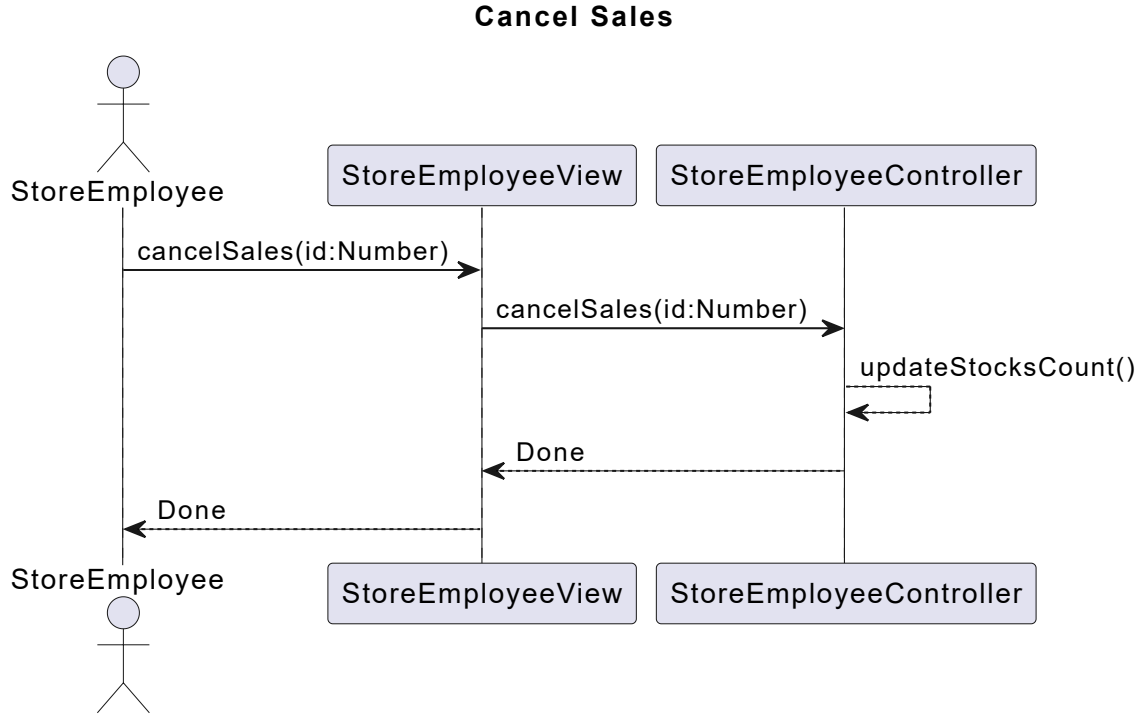
Update product



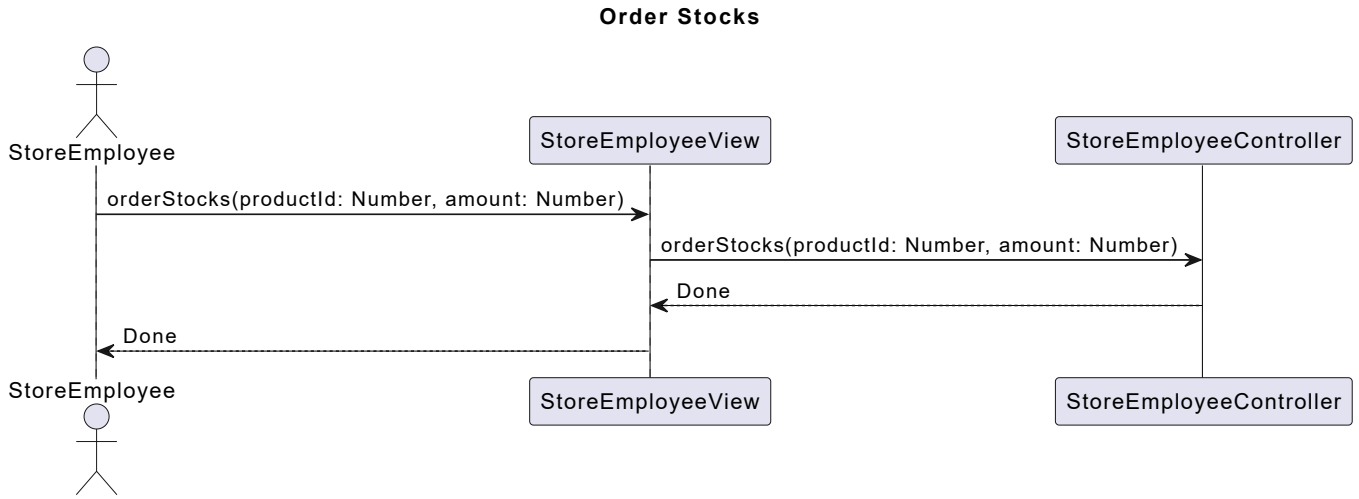
Save sales



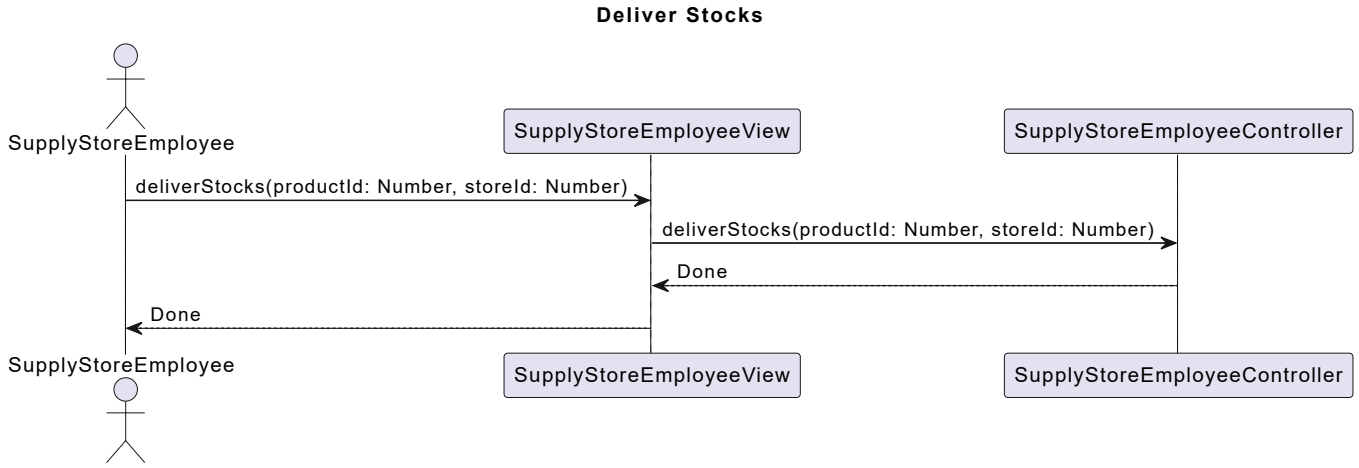
Cancel sales



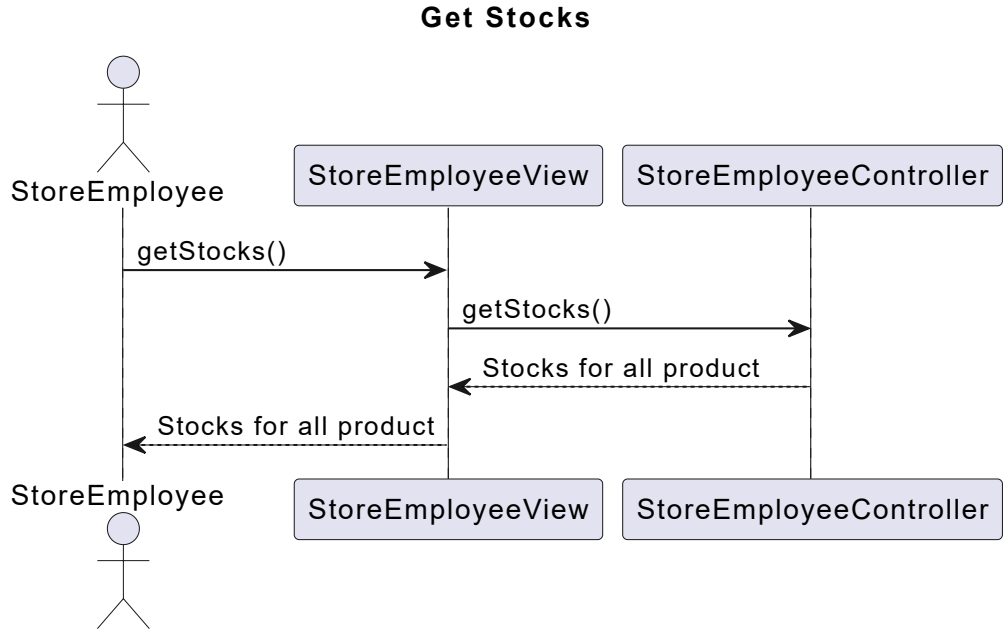
Order stocks



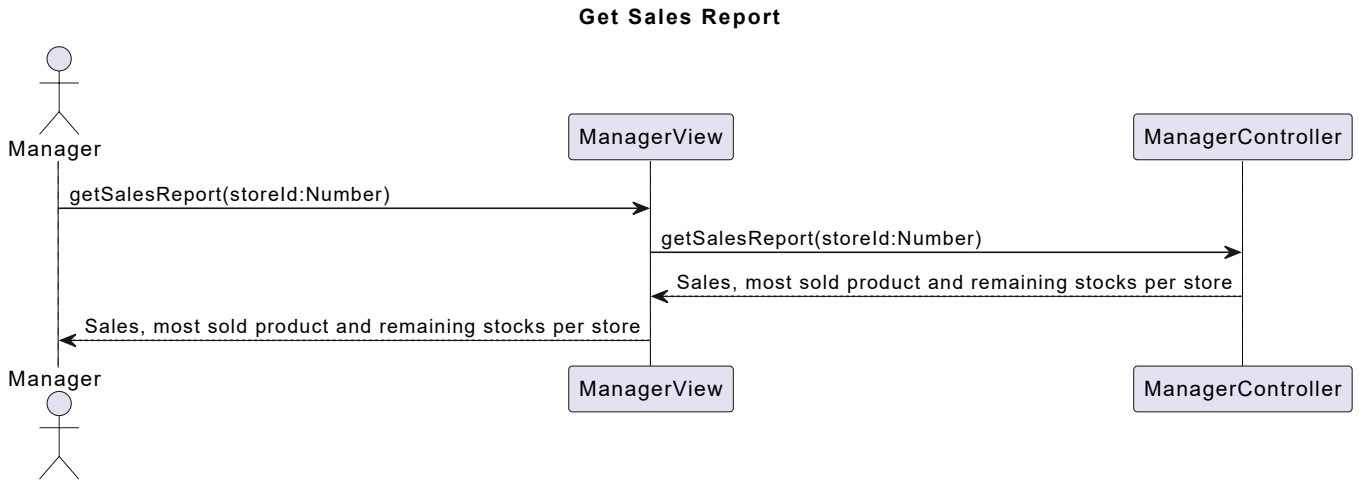
Deliver more stocks



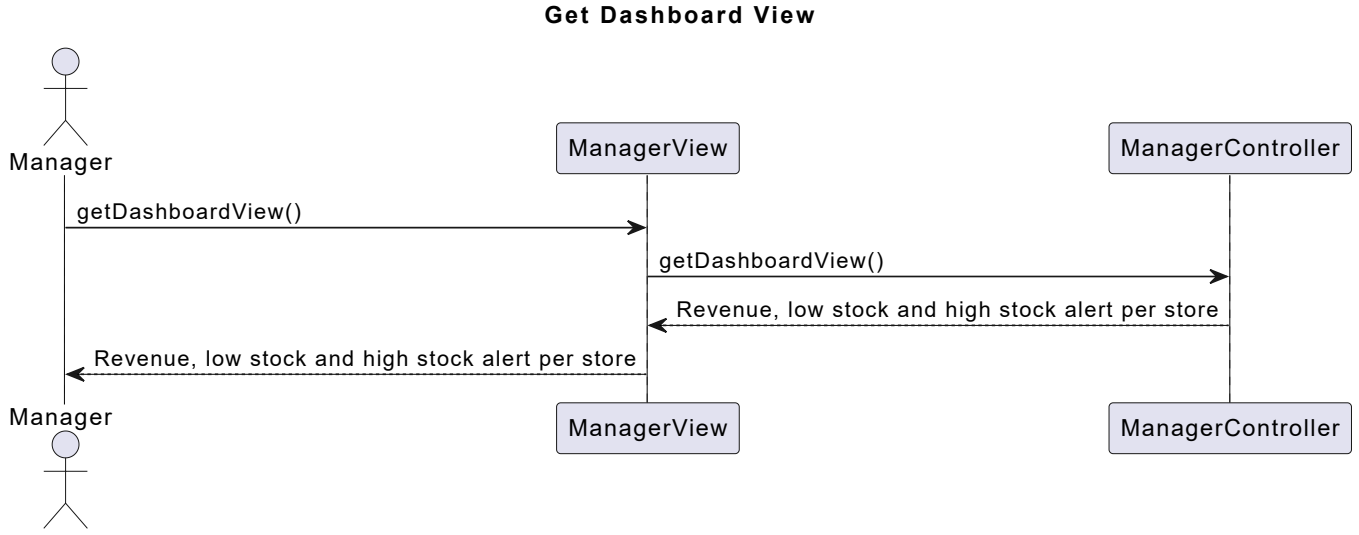
Get stocks



Get sales report



Get dashboard view



Design Decisions

MCV

Context

This app needs a basic console interface allowing the user to send and retrieve data from a **postgreSQL** database.

Decision

The **MVC** architecture is flexible yet simple, the code is easy to understand and everything can be put in place fairly quickly. There's still the ability to scale up the app in the future in case new features are needed. It's also a very easy architecture to test because the view doesn't really need testing, thus we can focus on testing the controller and model. The projet isn't going to grow very much in the future so we don't need an architecture that could support stuff like networking.

Status

Accepted

Consequence

The view will use **terminal-kit** to show and retrieve data from the user. Models will be automatically generated by Prisma and each of the 3 user types will have a dedicated controller to manage data between the view and the model. Everything will be organized into two folders:

- `/src/view`
- `/src/controller`

Inheritence

Context

There are 3 different types of users:

- an employee working at one of the store location
- an employee working at the supply center
- a manager who oversees all activities.

The manager has more power than the supply center employee and the supply center employee has more power than the store employee. A manager has to be able to retrieve stocks to generate a sales report, just like the store employee needs to retrieve stocks to order more stocks.

Decision

The manager will inherit from the supply center employee and the supply center employee will inherit from the store employee. This means the manager and supply store employee have the power to do everything as the regular store employees.

Status

Accepted

Consequence

Pictured in the **class diagram** is how each type of users will inherit form one another. This will be useful when implementing more complex features like the *sales report*. Each view will use a specific controller depending on which user is logged in. This will also allow for each users to be tested upon using Jest.

Risks and Technical Debts

Type

Some sections of code don't have types and use `any`, which can increase the chance of having data that's a different type which we intended.

View

Some views are too big and chaotic. For instance `sales-report-view.ts` falls into callback hell where we wait for a user input to trigger a callback and keep on with the execution. The code if very convoluted because everything is in callbacks. The way user input is treated isn't very scallable which could cause issues in the future if we ever need a complex view asking for a lot of input from the user.

Technology chosen

Node.js

Javascript is untyped so it's flexible with how it processes data. I think it's useful when creating a small app where you don't really want to pay too much attention to what type a certain field is and instead focus on making the app work.

Typescript

I still think having types is handy sometimes, of instance it can be useful when dealing with data coming from the ORM.

Terminal-kit

I've never used this library before, but from what I've seen on the readme page it seems to be a very versatile tool. Simple tasks like asking the user for input can also be done very easily.

Prisma

I've never worked with this library before, but there seems to be a huge amount of documentation online. I'm also somewhat unfamiliar with ORMs so i just asked *ChatGPT* for the best library for this particular project.

Jest

This is what I've always used. It was my first time setting up the environment for Jest and I had a hard time making it work with *ESM* and *Typescript*.

File structure

`/view` is split into the 3 different types of users found in the app (*store-employee, supply-center-employee, manager*). Each subfolder contains individual files for each interface. For instance there's a file which will print out the interface to search for a product. There's also a menu screen for each user giving access to the previously mentioned interfaces.

`/controller` is also split into the 3 different types of users. These controllers contain the methods used to send and retrieve data from the database. `manager.ts` inherits from `supply-center-employee.ts` which inhertits from `store-employee.ts`.

`/utils` has various methods used all around the app, like a custom method to output *json* or a custom method to ask the user to enter a *string*.