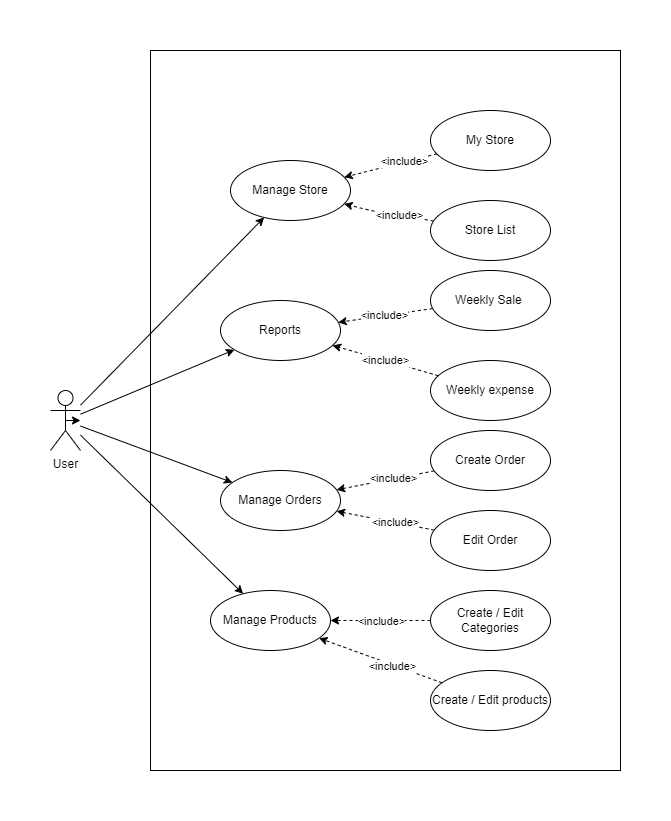
Invoice App Analysis and Microservices Architecture design

# Analysis

Invoice App from <https://github.com/rdg7739/Invoice> is a Windows Forms app with local database, with all business logic hardcoded in app. The app handles several use cases described below

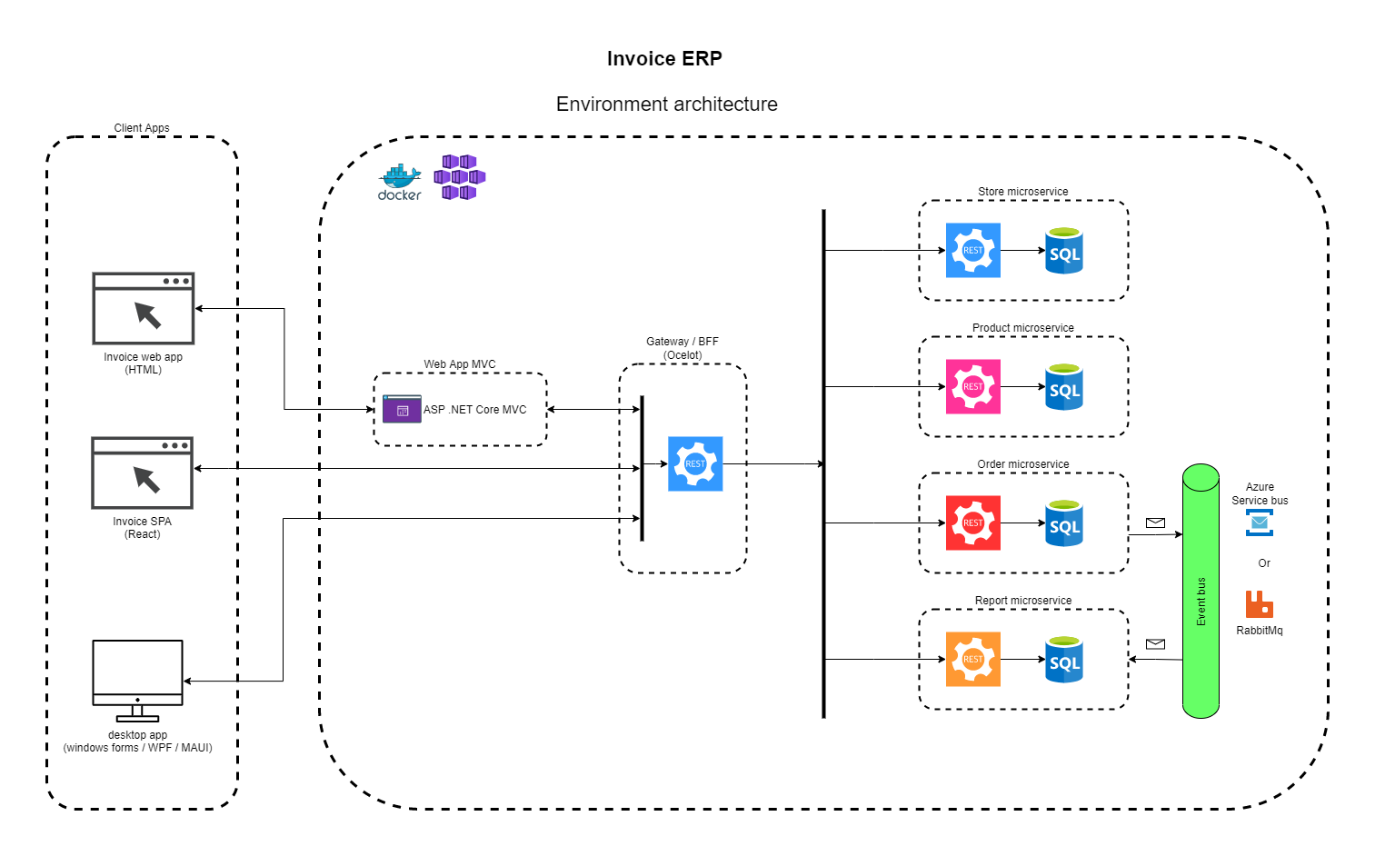
* “My Store”– used to manage contact data about our store.
* “Store List” – used to manage data about customer and suppliers.
* “Category List” – list of product categories
* “Product List” – list of products
* “Create Order” – used to create an order
* “Order List” – used to manage orders
* “Weekly Sale” – used to display weekly sale data for a selected week
* “Weekly Expense” – used to display weekly expenses data for a selected week



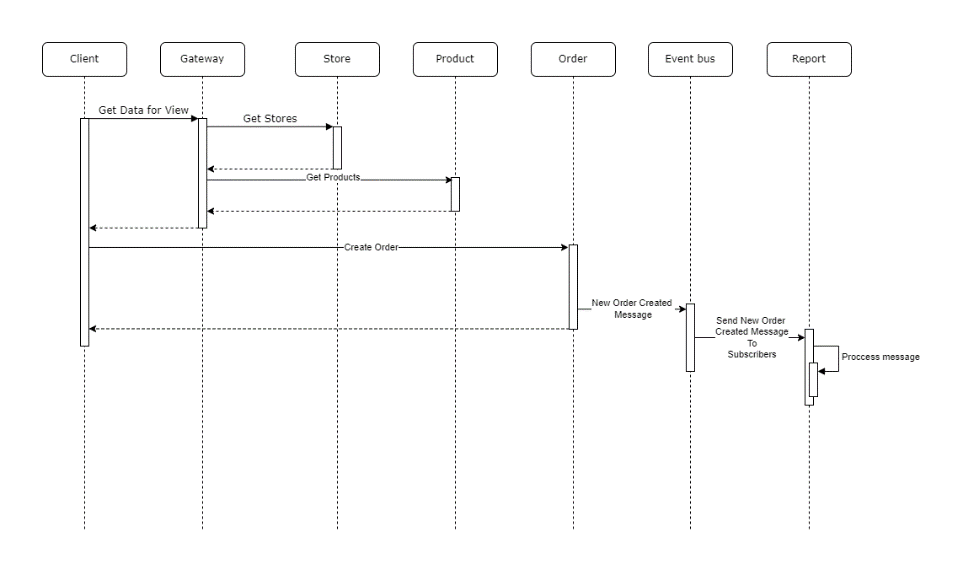
# Architecture overview

Based on these use cases I would divide this to following microservices:

* Store microservice – a service to manage data about “My Store” and the “Customer” / “Supplier”
* Product microservice – a service to manage product catalog and categories
* Order microservice – a service to manage and create orders
* Reporting microservice – a service to manage / provide data for reporting
* Gateway – a service used to orchestrate services



In this solution services are loosely coupled, and they are working with only their domain. Orchestration of the request is done by Gateway – Ocelot which will either route requests to specific microservice or orchestrate data for requests / responses. To ensure communication between services I would use an Event Bus (which could be RabbitMq or Azure Service Bus). In this case it involves communication between Order microservice and Report microservice, where Order service would publish new event after creating an order, which would be then processed by Report service and included in report data (described by Create order activity diagram below).



# Example implementation description

Source:

In the example implementation of the services I’ve created separate projects for Order, Product and Store microservices, they are build on .net 5 with Entity Framework (Code First) for data persistence, each service has its own database, using Clean Architecture approach with MediatR for CQRS.

In a production environment each service / client app would have it’s own repository, that would be used for CI/CD.

Each service consists of 4 projects

* API project – web api project, contains controllers, swagger interface, fluent validation for request
* Domain project – class library, that contains domain entities
* Application project – class library that contains messaging implementation for CQRS, again for simplicity each project has quite a lot of base classes that would be in real case extracted to a separate project and then included as nuget packages (e.g. Messaging base classes – CommandBase, QueryBase, HandlerBase, HandleWithResponseBase). Interfaces for command and query repositories are defined here.
* Infrastructure project – class library that contains implementation of repositories, api clients, persistence – EF db context, again for simplicity domain classes are configure in OnModelCreating method using Fluent API, which in real app would by done by configuration classes and their use in OnModelCreating method.

In this example I’ve omitted the gateway and the WinForms client app has the logic that should be done by the Gateway / BFF implemented in Invoice.Application and Invoice.Infrastructure project. In real world these projects would contain implementation of communication with the Gateway.

For the UI I’ve used the Windows Forms App from the solution that was analyzed and it was updated to .net5. The client app has **only some of the logic implemented**:

* Create Order (without save)
* Product list – (add, update, delete product)
* Store list (add, update, delete store)
* My Store (my store update)
* Category list (add, update, delete category)