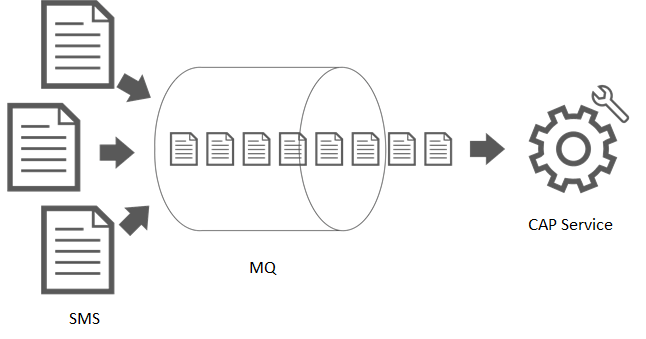
**Mitto’s Task**

Purpose of this document is to describe the implementation of required task. Project was written in .NET Core 3.1 in Visual Studio 2019, database MS SQL 2019 and Kafka as MQ were used.

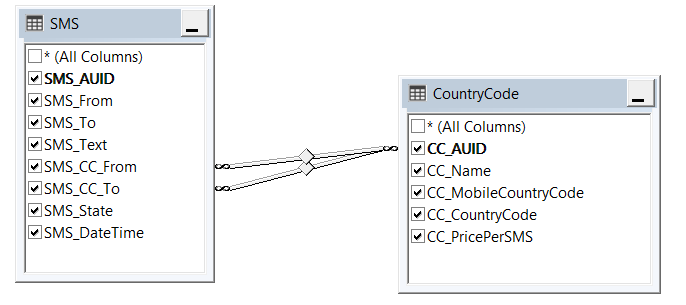
Architecture of solution:



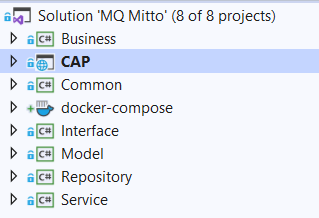
For the demo purpose publishers and subscribers are part of the same service (CAP service).

Published messages are being read and, based on service configuration, processed later on.

Database designer:



Solution:



Microservice architecture with repository pattern was used in this solution.

Controller – First layer, used to receive request and forward it to Business layer.

Service – used to resolve different services for handling messages, calls business afterwards.

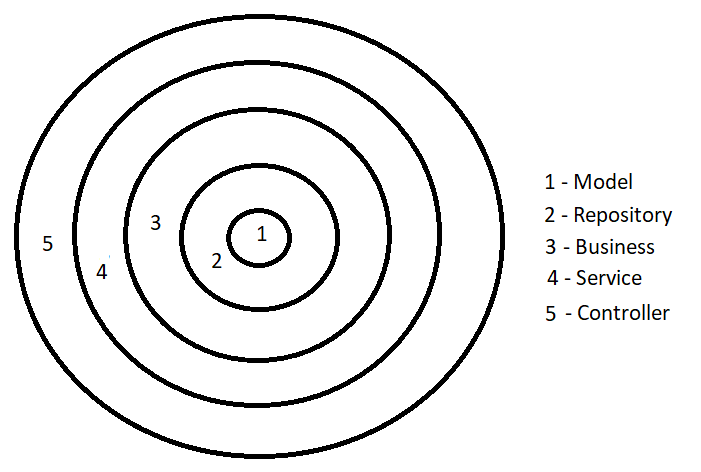
Business – Business Layer, all business logic is written here.

Repository – Repository Layer, all data from database are being fetched in this layer. Entity framework core was used to communicate with SQL.

Interfaces – project that contains only interfaces. Whole solution is covered with interfaces because of dependency injection.

Common – all enums and constants are placed here

Model – every entity, model, context is being stored here.



This microservice architecture is onion architecture.

**Challenges:**

* Integrating MQ in the solution
* Creating and running Kafka and zookeeper in docker
* Planning architectural concept of application

**Optimizations:**

* Unit tests
* Add authorization
* To develop and utilize service layer more efficiently
* Bind models, foreign keys and use entity framework more efficiently
* Better exception handling
* Add wrapper in repository pattern
* Add group to the consumer (not to use default one)
* Use different xml serializer (current one does not allow interfaces or anonymous classes)
* Etc.

Note: This is my first time playing with docker and MQ in real example, that is why this was challenging and inspiring for me. I haven’t got time to investigate how to docker-ise everything, so I don’t have that part with docker finished.