Description

Your customer is an online store with a warehouse and a delivery service. Solution components are the following:

* There is an MS SQL DATABASE on the server.
* Desktop application for the warehouse
* Web version for managers
* Web version for customers (buyers)
* Mobile version for couriers
* Mobile version for customers (buyers)
* There is also a third-party application for generating reports

All these applications are not connected to each other and work directly with database.

The architecture could be illustrated the following way:



Customer works in a single country but has plans to enter the markets in several other countries.

**Task 1**

Determine the architectural style of the solution. Provide an explanation for your choice.

***Resolution***

After analyzing potential approaches, it is obvious that **Monolithic** architectural style would be bad scenario due to reasons like:

* High coupling among different teams to maintain and evolve several apps which have potential different technology (Mobile Apps 2 + Online Shop Web + Back Office Web + Warehouse App Desktop).
* Scaling out some services independently would be impossible without scaling out the other services which might not having a high load.

I can see many benefits to having a **Microservices** architectural style where splitting the current services that support the different applications and the database to be able to scale and evolve in the time making teams which maintain this apps more agile.

For reporting purposes, the implementation can be a Message Oriented Middleware Architecture to mediate the communication between changes on the Database to be consumed by the reporting System whenever a new Order is in place, we can trigger a publishing / subscriber message in order to fill in the Datawarehouse that the 3rd party reporting system will be using. For that, Serverless functions could be useful to process this messages whenever new Orders, changes in the Stock, updates on Profile data of users/customers, and any relevant information regarding Deliveries are publishing information around it.

Therefore, my final solution will be a hybrid Architecture style based mostly in Microservices and Serverless functions.

**Task 2**

Draw the architectural diagram of the solution as if you were designing it from scratch. Provide pros and cons of the two solutions.

***Resolution***

Proposed solution as described on Task 1 having this architecture style of Microservices with hybrid mode implementing Serverless Functions.



Pros and Cons between original solution and proposed solution:

|  |  |  |
| --- | --- | --- |
| Topic | Original solution | Proposed Hybrid architecture |
| **Scalability** | Cons: Difficult to scale out specific parts of the system which are having a high load in comparison to other parts of the application. | Pro: Easy to scale out the app services from each of the backend micro services. |
| **Consistency** | Pro: It’s easier to have a DataAccess Layer and Business logic in one server interacting with the SQL database having more reliable transactions and therefore always satisfy the need of high consistency of the information if needed. | Cons and Pro: By definition, having to communicate between multiple microservices, could be that one of them goes down for a limited period of time and making eventual consistency instead of high available consistency. However, if implemented a SAGA pattern for cross services transaction this could be mitigated for more critical operations such as order processing, payments, and deliveries. |
| **Plans to rollout cross regions** | Cons: since not so easy to scale out the backend and database if the need of operate in multiple countries requires this, this approach could be a problem for future operations. | Pro: when hosting the multiple client applications for customers cross world as well as having the Backend as a Service on the APIs in the cloud it is much easier to place it available with Georedundancy in multi regions. So latency for customers in different parts of the world could experiment similar SLAs. |
| **Security** | This approach is exposing the database to the 3rd party reporting system which can be a risk in the future. | A bit more complex to implement but having a dedicated service to handle the authentication of the users and having to issue Tokens for API authorization so clients like the Desktop, Mobile Apps and Web Apps can interact with the other backend services, it makes it more robust. Also in this approach the main databases are not exposed to 3rd party vendors |