Description

*Please, discuss with your mentor the following:*

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:

Diagram

Description automatically generated

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’s been decided that **Monolithic** architectural style would be bad scenario. The negative sides are mainly:

* 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 the Customer Mobile App or Online Shop Web independently would be impossible without scaling out the other services which might not having a high load.
* Deployment in this monolithic approach could be not agile at all since QA is needed for all the parts of the application even just changing in the back-office web.
* Since there are future plans to rollout the applications to multi regions/locations this approach is much more difficult to maintain since the entire application will need to implement this requirement instead of the apps that might only needed (customer online shop web and customer mobile app).

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. However, there are some cons regarding the nature of Microservices like the describe in **BASE** (Basically Available, Soft State and Eventually consistent). Certain parts of the core system like processing orders in my opinion will need to support Durability and Consistency in order to commit with the compromises of delivery and order processing, as well as maintaining trustworthy levels of stock.

For example, benefiting from the API Gateway mechanism we can have a shared common Orders API for processing the orders and interact with the Database and then the client apps will communicate with this API.

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, with certain non-normalized databases and some normalized-strong consistency databases to process the key transactions of the system. Saga pattern to synchronize satellite services should be implemented to satisfy the main requirements on high consistency for Orders and Stock.

The real implementation will be described in the Task 2 (see graph and pros and cons), however a brief split of the microservices to be implemented in this architecture would be:

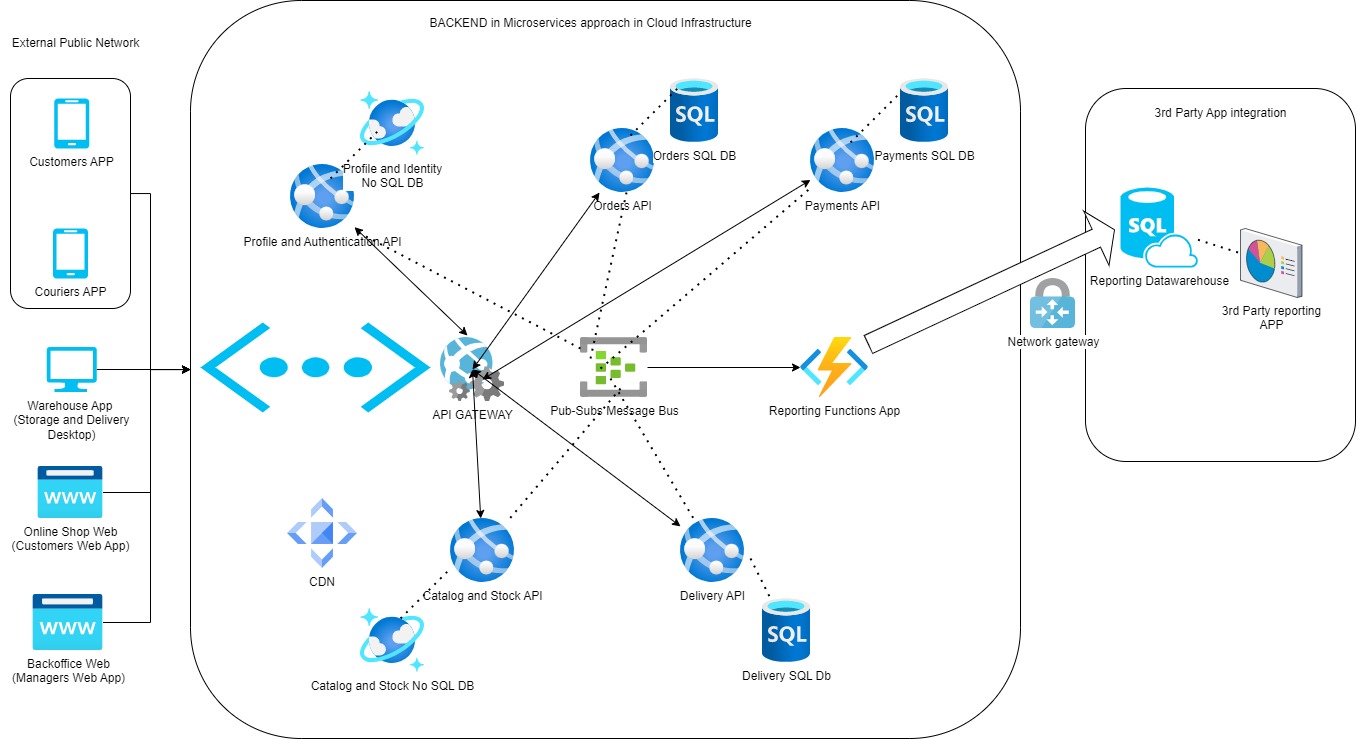
* **Profile and Authentication API:** backend service in charge of handling Profile data of users (managers and customers) as well as Identity data of these profiles to authenticate them when operating with the different apps (desktop, web, mobile, etc). The data could be persisted in a no-SQL fast response database since the schema for persisting profile and identity data is not very likely to change over the time.
* **Catalog and Stock API:** backend service to display the products and managing stock availability of the products describe from the catalog and being consistent to validate and process Orders offered by this e-commerce / online store that can be browse either from the Customer Mobile App, Online Web Shop and accessible by the Backoffice webapp. Datawarehouse desktop will also need to consume this API
* **Orders API:** backend service to process, create, modify, display orders of products consumed by clients like the Customer Mobile App, Online Web Shop, Backoffice webapp and **Catalog and Stock API** and **Delivery API**. ideally persisting data in a SQL RDBMS with high availability and strong consistency.
* **Delivery API:** consuming information coming from Orders and serve backend to the Couriers APP and Customers Mobile APP as well as display relevant information for the Online Shop if needed. The Warehouse desktop App can interact to verify that orders are getting process and send to delivery.
* **Payments API:** Available for orders to call this service to process payments
* **Mobile APP Service:** backend supporting both Curriers and Customers Mobile APPs interacting with Profile and Authentication API + Delivery API + Orders API + Catalog API + Stock API.
* **Reporting Serverless Functions:** FaaS that could process messages from a shared Service Bus across all APIs to synchronize in an async way the data processed by the different APIs and its databases and feed the Datawarehouse which the 3rd party reporting system is served to process its reports.
* **Client APPs:**
  + Warehouse APP
  + Backoffice Web
  + Online Shop Web
  + Couriers Mobile App
  + Customers Mobile App
  + 3rd party reporting App

**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 and API Web Services representing the Backend as a Service.  
The interaction with the Client Apps is done via API Gateway. To secure the backend access to the different clients, a mechanism of Token authorization can be implemented in order to secure our APIs to the external public world.



Pros and Cons between original solution and my 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. If needing to scale out, all parts of the backend and database have to be done simultaneously. | Pro: Easy to scale out the app services from each of the backend micro services. If a marketing campaign requires that the catalog and stock API will be more consumed than the Payments service, then individual microservices can be scaled out more easily. |
| **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 risk 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 |