B2B Demo

#### Problem Statement

Duplo is planning to work with businesses by registering them and creating their accounts on

the Duplo platform.In turn, a business owner whose business is registered on the Duplo

platform, can go ahead and create user accounts for their businesses’ department heads(for

example soft drinks department, toiletries department, etc), who are the actual active users of

the platform.

In each business, the department heads are responsible for monitoring the status of the

inventory, and are required by the business owner to make orders using the Duplo platform

each time before the inventory in their department runs out.

\*\*Note that Duplo does not plan to sell the actual items, but is just a platform that is used to place

these orders

On a typical day from its many businesses, the Duplo platform sometimes gets up to 1,000

requests per minute from different department heads of multiple businesses, all making

orders through the platform. Duplo stores this order data in a Postgres database.

However, in its future product plan, Duplo plans to start offering a buy now pay later service

to its registered businesses, where businesses will be able to make orders for inventory, and can

pay for it later.

To achieve this, the Duplo platform has started logging transaction data into MongoDB.

Duplo logs the following data of a transaction/order: businessID, amount, date, status

The platform intends to later use this data to calculate the credit score of a business, to

determine how much credit a business can access, using the following formulae:

Total transactions or order amount / ( Number of transactions \* 100)

Also, the government tax authority has introduced a new law, which requires the Duplo

platform to make POST requests to an API it has provided for each order the Duplo platform

receives, so that it can log this data for tax purposes.The government tax authority STRICTLY

dictates that the details of each order must be passed to its API.

However, the API can be slow sometimes, and as such responses can take up to 15-35 seconds

to be returned by the government API.

The API URL is: <https://taxes.free.beeceptor.com/log-tax>

Sample expected JSON request payload is:

{

"order\_id": "dac3549d-aea2-4957-91dc-618f2e2c77f7",

"platform\_code": "022",

"order\_amount": 40000

}

#### Functional Requirement

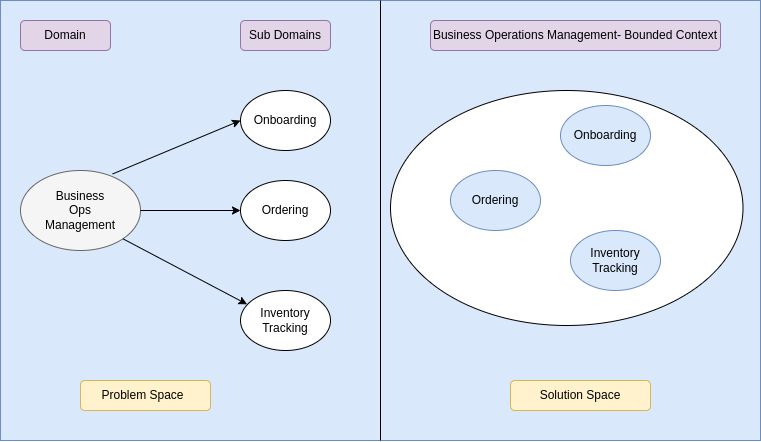
1. Curate orders from each businesses’ department lead
2. Process and store the order details as described above
3. Provide an API endpoint for a business to get their credit score
4. Provide an API endpoint for a business to fetch the order details (total number of orders, total amount of orders, total number of orders today, total amount of orders today)
5. Create a React App to display the orders of each business in a table. When one record is clicked a modal should pop up to show the order details
6. Deploy the backend solution as a docker image
7. Bonus points if you can setup a simple CI/CD pipeline to deploy the working frontend app on netlify and provide a preview link
8. Commit your code to github and add jask-vmi and kamaukelvin as collaborators

#### Analysis

The **Business Domain** is a B2B business operations management platform that provides business workflow automation, Onboarding, Order management, Inventory management services from businesses to businesses. From this understanding from the problem statement, the problem to solution space description will be described as a business Ops Manager.

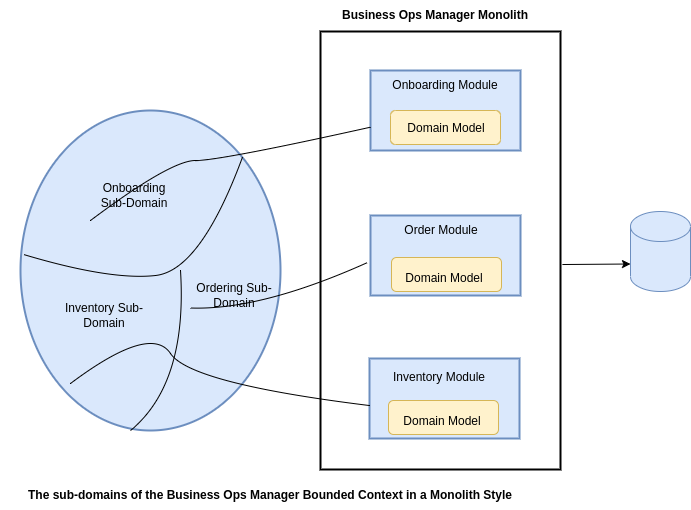
The **Sub Domains** are identified to be the following; Onboarding, Purchase Ordering, Inventory handling.

The **Bounded Context** is the carving out of the solution space for the business domain and its subdomains representing the capture of the core business problem and the direction of what their solution looks like. We have separate domain spaces for each of our subdomains and so our bounded context will be mapped to a specific bounded context.



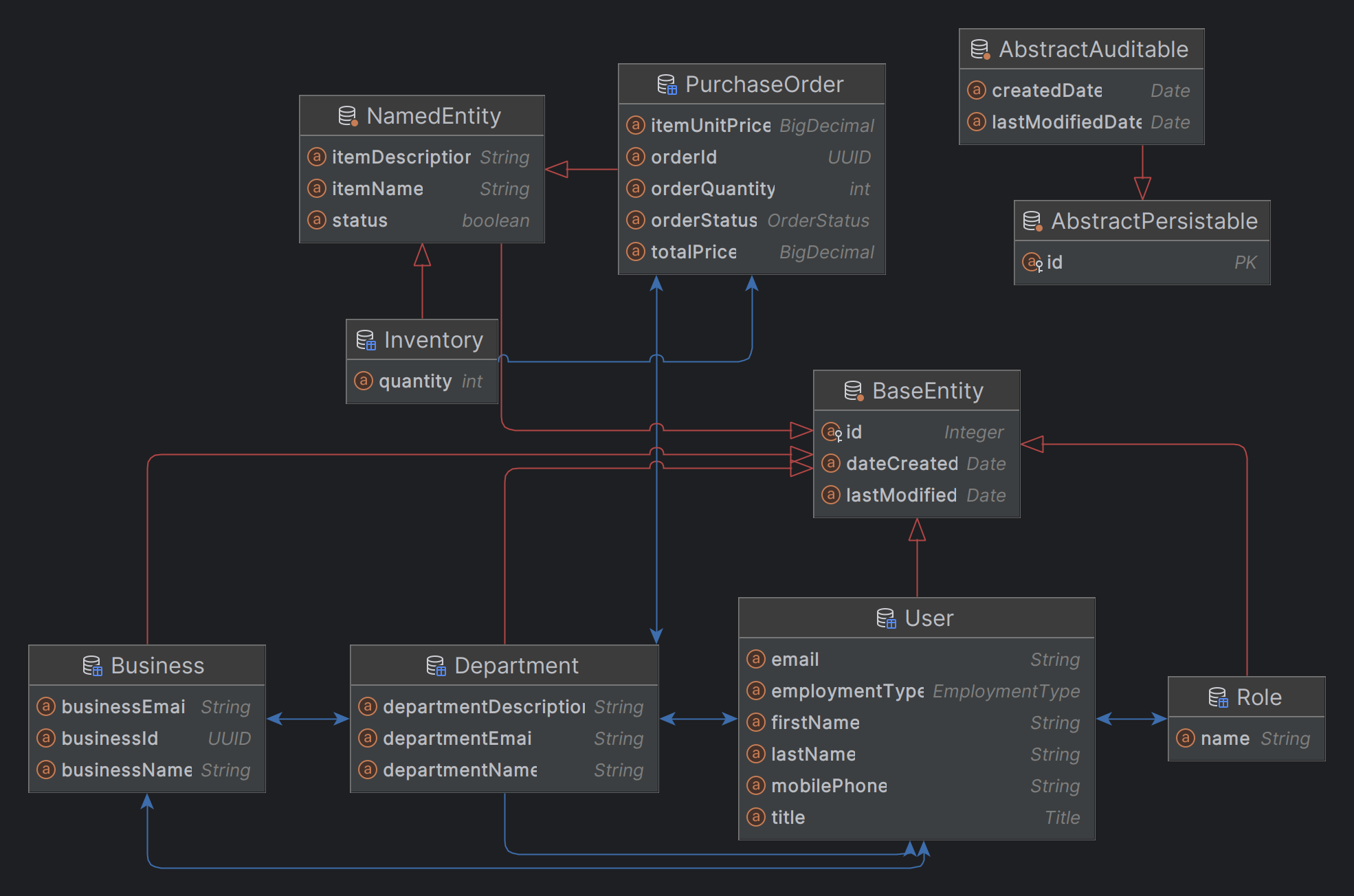
Within the scope of our Sub-Domain/Bounded Contexts, we have identified the following business areas:

* Onboarding:
  + Open Business Account
  + Users Setup
  + Department Setup
* Ordering: To maintain service delivery to its customers, a business orders goods and services from suppliers.
  + Raise purchase request
  + Process purchase request approval
  + Compute credit score
  + Generate order report
* Inventory Tracking: As stocks are getting sold, the system tracks threshold and triggers signals when breached.
  + Trigger signal when quantity is less than threshold



#### Design

**Domain Model**



**Data Access | Repository**

Repository pattern is used to implement data access for the entities.

| **s/n** | **Entity** | **Repository** |
| --- | --- | --- |
| 1 | Business | BusinessRepository |
| 2 | User | UserRepository |
| 3 | Department | DepartmentRepository |
| 4 | Purchase Order | PurchaseOrderRepository |
| 5 | Inventory | InventoryRepository |

**Service Layer**

Service (facade pattern) interface is used to expose all operations that are to be implemented in service implementation class.

| **s/n** | **B2BService** | **B2BService Implementation** |
| --- | --- | --- |
| 1 | Business Operations | 1. GenericResponse<BusinessRecord> openBusinessAccount(BusinessRecord businessRecord); 2. GenericResponse<UserRecord> addStaffToABusiness(Integer id, UserRecord userRecord); 3. GenericResponse<DepartmentRecord> createDepartmentForABusiness(Integer id, DepartmentRecord departmentRecord); 4. Optional<Business> getByEntityId(Integer id); 5. Optional<Business> getByBusinessId(String businessId); |
| 2 | Department Operations | 1. GenericResponse<DepartmentRecord> addStaffToABusinessDepartment(Integer departmentId, Integer userId); 2. GenericResponse<String> calculateCreditScore(Integer businessId); 3. GenericResponse<DepartmentRecord> addStaffToABusinessDepartment(Integer departmentId, UserRecord userRecord); |
| 3 | Order Operations | 1. GenericResponse<PurchaseOrderRecord> createPurchaseOrder(Integer departmentId, PurchaseOrderRecord purchaseOrderRecord); 2. GenericResponse<PurchaseOrderSummary> getDepartmentPurchaseOrderReport(Integer departmentId); |

**Web API**

The web layer is implemented in a RESTful style architectural pattern relieving it of any UI responsibility. The api documentation is a postman collection and will be available on project github repository at [b2bplatform](https://github.com/bolarge/b2bplatform).

In addition, its swagger documentation can be accessed after its deployment at {{base\_url}}/b2b/swagger-ui/index.html#/

#### Testing

The increment contains no test, this will come in subsequent increments

#### Deployment

1. Ensure your machine host has both docker and docker-compose running
2. mkdir b2bplatform
3. cd b2bplatform
4. Download compose.yml file from [b2bplatform repository](https://github.com/bolarge/b2bplatform/blob/master/compose.yaml)
5. To run, issue the command ‘docker compose up’
6. Access swagger doc or postman collection and try out something
7. To stop, issue the command ‘docker compose down’