### **About the Hackathon**

You must create a software solution for managing the ordering process in Amazon. To achieve so, you should create 3 logical boundaries either using the microservices architecture, a modular monolith **or your preferred architecture**.

* Sales context
* Billing context
* Shipping context



#### **Goals**:

* Increase your understanding about Service Oriented Architecture and Event-Driven architecture

#### **Notes**

* The words “service”, “context” and “module”, for sake of simplicity, are used interchangeably.
* The keywords "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in RFC 2119.

### 

### [**Technical specifications**](https://drive.google.com/drive/folders/1DIV-T37aZk7KrR1RH8AjmosFSGmZqUX4?usp=sharing)

#### **Event models**

* [Happy path, payment failure and order refunding scenarios](https://drive.google.com/file/d/1xbz2fCJUNt11_HxFnn_3PujBBWvEP7Zs/view?usp=drive_link)
* [Order status transitions in Sales context](https://drive.google.com/file/d/1yFMN_nAO1w3IJeCCQQLJFz88vSI4LHAl/view?usp=drive_link)

#### **Domain models**

* [Overview of domain models](https://drive.google.com/file/d/1KU6pZatf9U_do8q157HVbfuOb6Znm9ij/view?usp=drive_link)
* You **MAY** implement other models, for example a Payment model in the Billing service to keep track of how much we charged our customer for a specific order in a given point in time, so that we can use this information in case of some Refund operation.

#### **Services Architecture**

* [High level architecture](https://drive.google.com/file/d/15cBhuJHITu75bOaMHS-DmM9KKpSCPLCw/view?usp=drive_link)
* [Resilient message handling architecture](https://drive.google.com/file/d/1cfG3k5o3XgLJeMusLlVqr6O46AMgNgVD/view?usp=drive_link)

#### **OpenAPI contracts**

* [Services (modules) API contract](https://drive.google.com/file/d/1QGjI4TsO-tUC0-qMWundmaHSWOee7e1b/view?usp=drive_link)
* [API Gateway API contract](https://drive.google.com/file/d/1Bp7IQpHcKl21JXEtsq_yOz5S_MLY_ulw/view?usp=drive_link)

#### **AsyncAPI contract**

* [AsyncAPI for all Services](https://drive.google.com/file/d/1ZjEAZGfgPvMx6rtR1UylZ1MpqHuOE0gn/view?usp=drive_link)

#### **Sample Seed JSON files**

* [Products for Catalog module](https://drive.google.com/file/d/16_QnRlqUJDpB4DvWA2hMZAzJxmk2DFUG/view?usp=drive_link)
* [Products for Sale module](https://drive.google.com/file/d/1DlO3oQg9h-v6LTRSW6WmuILx_VphSTiy/view?usp=drive_link)
* [Products for Shipping module](https://drive.google.com/file/d/18NoDNxYG1QiwYHvHSp9-mD2DJ5ZTJ298/view?usp=drive_link)
* [Billing Accounts for Billing module](https://drive.google.com/file/d/115j8b8m73IINfmh9fLMwuJ5cQSLqCyHq/view?usp=drive_link)

### **Business rules**

#### **Billing service**

* You **MUST NOT** bill an order if we don’t have enough money in the billing account.
* When charging a customer, the balance **MUST** be decreased.
  + The available balance **MUST** be greater or equal to zero always. **Be aware of concurrency**.
* When refunding an order, the balance **MUST** be increased according to the Payment amount.

#### **Shipping service**

* You **MUST** only create a Shipping Label if both Order Placed and Order Billed events have been handled by shipping service.
* When creating a Shipping Label, if there’s enough inventory, you **MUST** decrease the products quantity on hand by the quantity the customer selected.
* You **MUST NOT** oversell more than the configured stock. If there’s at least one product in the order products list for which you don’t have enough stock, then you **MUST NOT** create the shipping label for the order.

### **Requirements**

* You **SHOULD** version your projects inside your team leader's personal workspace in GitLab.
* You **MUST** implement the endpoints that are defined in the OpenAPI contracts.
* You **MUST** implement the events that are defined in the AsyncAPI contract.
* You **MUST** use docker.
* You **MUST** use a PostgreSQL version 16.
* You **MAY** use NestJS for building a Modular Monolith.
* You **MUST** implement the Resilient Message Handling Architecture (RMHA) and its patterns (Transactional Outbox, Inbox, Retries, Error Queue) using RabbitMQ as the message queue.
* You **MAY** implement the Failed Messages Monitoring service.
* You **MUST** consume events concurrently by configuring a Prefetch Count of 25.
* You **MAY** implement a frontend app like the described in the example image.
* You **MAY** implement a login in the frontend app using firebase.
* You **MUST** use the provided json files to seed initial information.
* You **MUST** store the seed files in some kind of folder within the project (*e.g. infrastructure/static*) which won’t be tracked by git. For this you have to use some kind of file system driver so that we can use either local disk in development or cloud disk in production.
* You **MUST** use a single database server with either multiple databases or a single database with multiple schemas for each module/service/context.
* You **MUST** provide some functionality for refreshing the database (truncate all tables).
* You **MUST** deploy all your services (services, databases, rabbitmq, buckets) in the assigned Google Cloud Project.
* You **MAY** implement CI/CD in your projects, if so you **SHOULD** design your pipelines based on a git workflow (git-flow, trunk development, etc.).
* You **MUST** follow the deployment requirements.

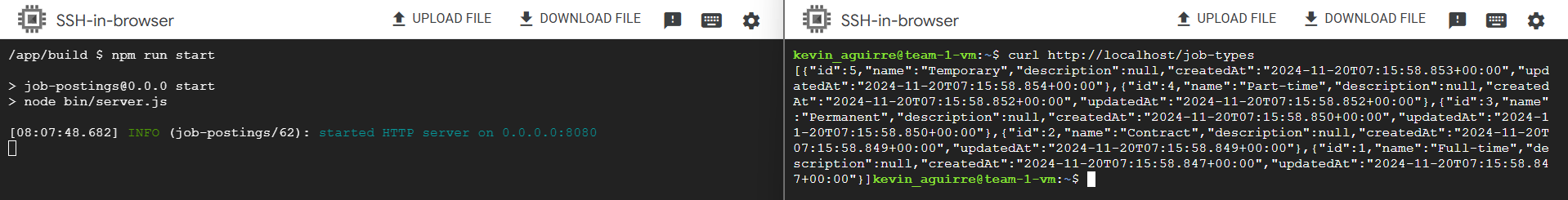
### **Finishing the hackathon**

* To finish the hackathon, each team **MUST**:
  + Create a **1.0.0** tag
  + Submit the URLs of the tagged repository in GitLab. This tag will mark the **stable** and **deployed version** of the project.
  + Submit the production url of the API Gateway and all of the other services. For example: *http://34.46.79.199/api/v1/*
* The submitting **MUST** be done by the team leader in the hackathon Slack channel r**ight before the hackathon finishes (08:00 pm)**. If you don’t send the tag by this time, the evaluator won’t review your work.
* Once the tag is submitted no other modifications will be allowed. **DOING THIS WILL DISQUALIFY THE TEAM.**

### **Deployment requirements**

* Access via SSH to the available VM instance for your team in the assigned GCP project -> Compute Engine.
* The machine has git and docker already installed.
* Clone your repository and configure it to get it up and running in production mode by using docker compose.
* You **MUST NOT** start the applications in Development mode.
* You **MUST** use port 80 as the exposed docker compose port for the API Gateway.
* You **SHOULDN’T** expose any other service API.

### 



### **Evaluation Process**

The evaluation will take place **the day after the hackathon**. Each team will explain briefly how the development was to begin with.

When the hackathon is finished, an orchestration script will be executed. Using a controlled input, we will spam your production API Gateway POST endpoint submitting all the desired requests of orders.

Then with all the orders stored and when the team lead lets us know that all the orders are processed, we will hit the GET endpoint, obtain the final status of the orders.

Finally, the evaluator contrasts the results of your orders service against the expected ones that are only known for the evaluator.

Keep in mind that [the shared json files](https://drive.google.com/drive/folders/1BJ_rUFjIZdNTVFUE9At1BhqVI06-7W0a?usp=sharing) are just a guide to create the seeders file so you can know the key names in advance during the seeds execution. The evaluator will add different sets of data to test different scenarios.

#### **Evaluations points**

In order to apply for evaluation and win base points, you **MUST** meet all the defined requirements.

Expected results **MUST BE** an **exact match** between the expected results against the obtained from your services: (**100 base points**).

* Scenario 1: **20** points
* Scenario 2: **30** points
* Scenario 3: **50** points

#### **Extra points**

* Evidence of queues implementation requirements (+20)
* Frontend app: (+20)
* Frontend login: (+10)
* Failed Messages Monitoring service (+20)
* Fully Dockerize development environment with required services (+10)
* CI/CD: (+10)
* Testing (+10)

In order to apply +points, one scenario **MUST BE** complete. All teams will be tested with the same scenarios (controlled inputs).

### **Recommendations**

* Keep in mind that the frontend **app is optional**, but **you will get extra points if you build the app**, therefore building the **Products Catalog** service/module/context **is also optional**.
* Take a moment to plan and divide your solution into MVPs (deliverables).
* A feature-based architecture may help, but choose your preferred architectural styles and patterns.
* **Don’t try to build a perfect and clean solution**. Instead, focus on delivering value to your stakeholders (KISS).
* If you're implementing a modular monolith, each database seeder MUST be executed for a specific database module connection (e.g. different schemas).
* Creating the **Failed Messages Monitoring service is optional**, poisson messages may be checked directly in the error queue by the evaluator.
* If you’re going to implement the frontend app, you may want to include the API Gateway as part of the same project (*for example a NextJS app with an API*)
* Add the RMH architecture (message bus) as part of a **shared module** in the NestJS app.
* Try not to use too many Cloud Services, **you can just create a single Compute Engine instance** and use docker-compose.

**Resources**

* [Modular Monolith NestJS](https://github.com/kevinnaguirre9/nestjs-microservices)
* [Sagas: Event Choreography & Orchestration (NServiceBus)](https://youtu.be/rO9BXsl4AMQ?si=1snug2IFHrZf_Vat)
* [Putting your events on a diet](https://particular.net/blog/putting-your-events-on-a-diet)
* [SAGAs](https://docs.particular.net/tutorials/nservicebus-sagas/1-saga-basics/)
* [Filesystem Local Driver and GCP Driver example implementation](https://gitlab.fbr.group/sales/scholarship-application-requests/scholarship-application-requests/-/tree/main/src/infrastructure/storage?ref_type=heads)