Grill Fresh Online

Contents

1. Introduction

This document offers a high-level overview and explains the proposed architecture for "Grill Fresh Online", a pizza online ordering system for Grill Fresh Inc.

2. Context

Grill Fresh Inc wants to grow its business across the country in a gradual manner, in terms of customer base, volume of business and number of stores. It wants to achieve this goal through the following accelerators of business:

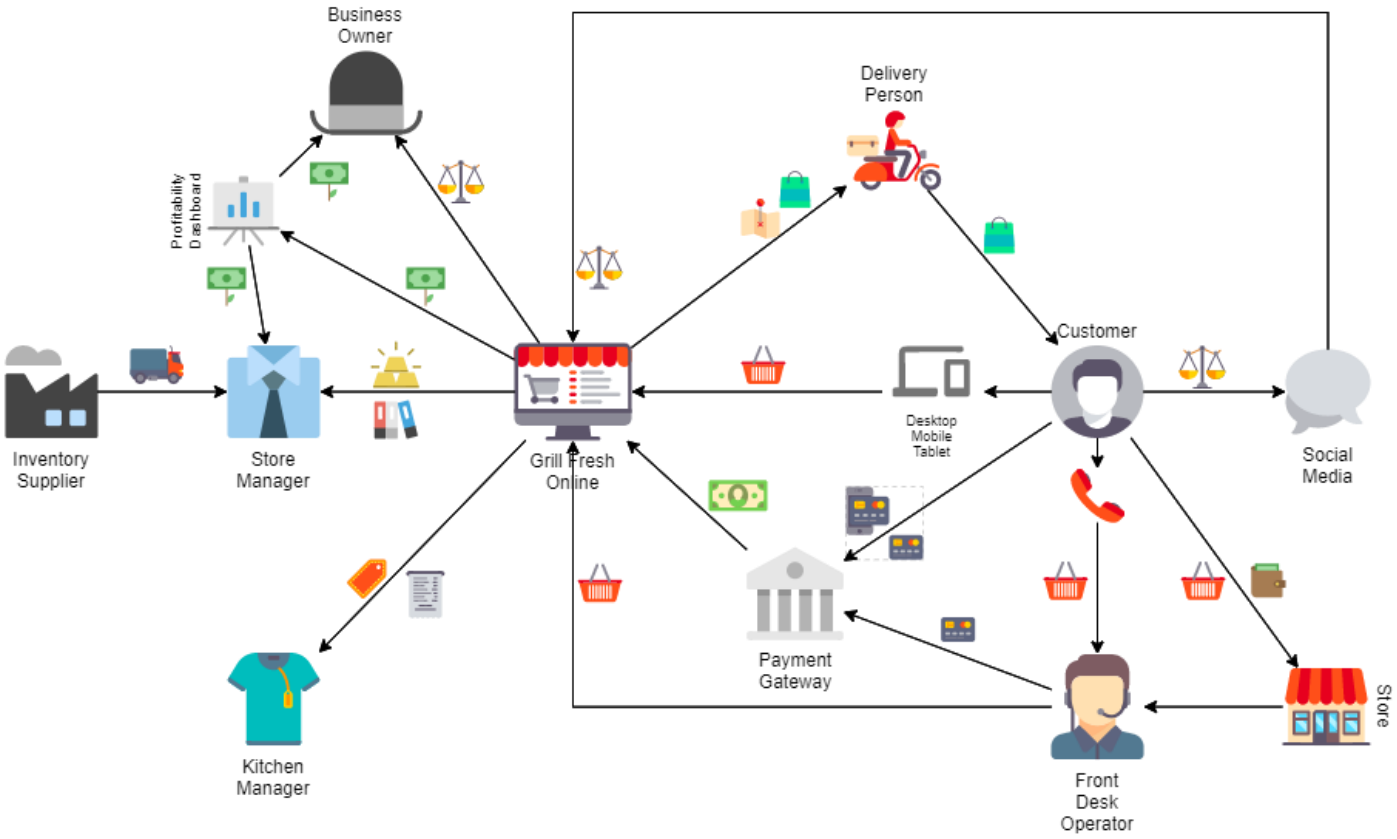
- A customer places orders for pizzas through online with any nearby store – instant and easy
- A customer receives delivery through additional modes like Take away and Home delivery, apart from dine-in option
- The business owner gathers a holistic view of profitability at business level or individual store level – to locate customer interests and expansion opportunities
- A store owner gathers a holistic view of profitability at store level – to understand business health and to act
- The business owner and store owners prefer to understand the sentiments of customers about business – to collate and correlate feedbacks (in near real-time, if possible)

Grill Fresh Inc is convinced that the potential enabler for this goal is to embrace the online platform opportunities.

2.1 Grill Fresh Online (GFO)

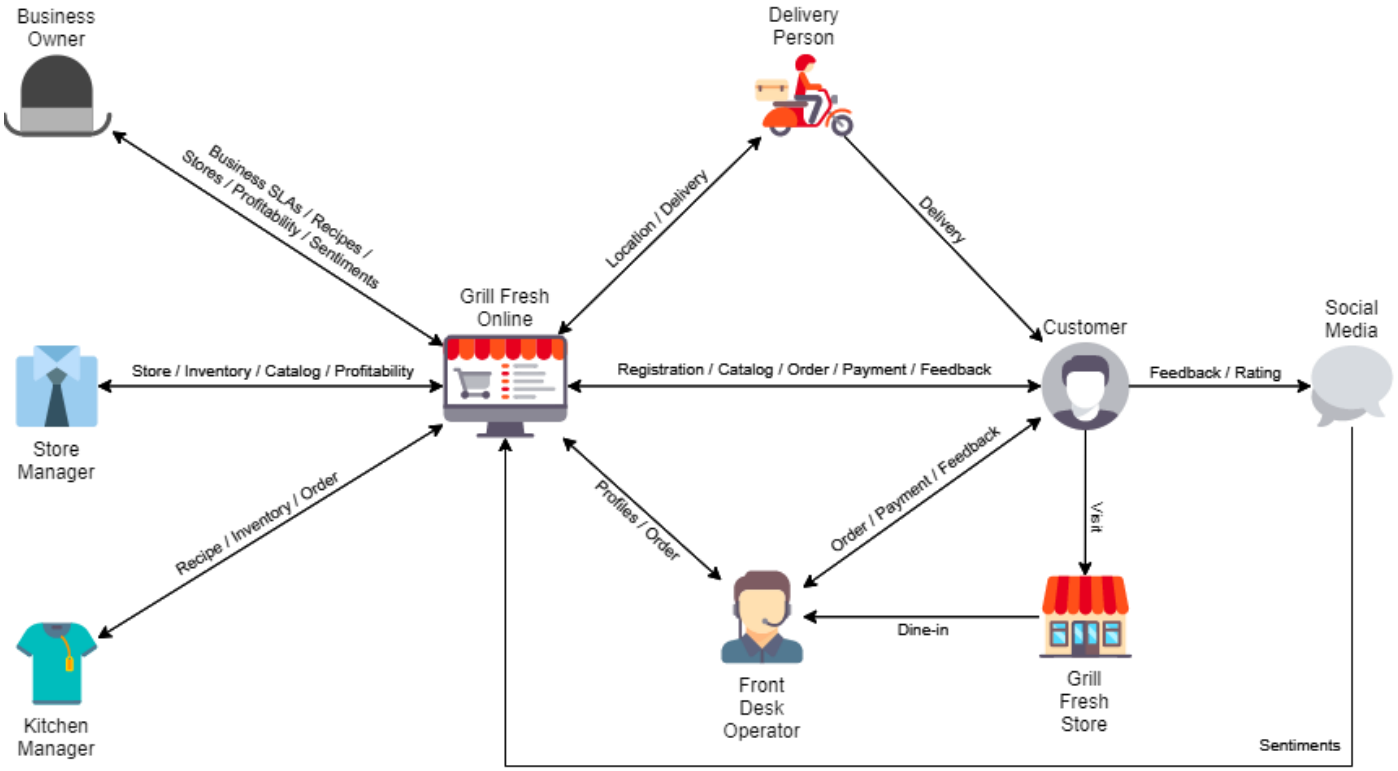
GFO system is a unified platform for all stakeholders of business – customers, business owner, store manager, front desk operator, kitchen manager and delivery person – catering to their varied interests. It is accessible through various devices – desktops, laptops, tablets, smart phones and Point-of-Sale.

At a high level, this system along with its neighborhood systems:



3. Functional Overview

Various major workflows fulfilled by the GFO system are as described below:



- Customer:
  - Register and manage profile – contact details and preferences
  - Browse through catalog of available items – pizzas, toppings, sides, desserts, beverages
  - Place orders for take away and home delivery
  - Pay for orders using varied secure payment options
  - Track status updates for orders placed
  - Provide sentiments about a store, item or overall business by direct feedback or over social media
- Store Front Desk Operator:
  - Browse through catalog and place orders on behalf of a customer – for dine-in, take away and home delivery
  - Approve and move orders into queue for Kitchen Manager
  - Track status updates for all orders placed in store
- Kitchen Manager:
  - Browse through recipes and ingredients for preparing items
  - Check inventory of ingredients and notify purchase needs
  - View and receive notification about incoming orders in queue
  - Process and notify status updates for orders
- Delivery Person:
  - Pick-up out-for-delivery orders from delivery queue
  - View order and delivery information for customer
  - Confirm after delivering orders
- Store Manager:
  - Manage store – address, serving radius, front desk operators, kitchen managers and delivery persons
  - Manage catalog of items serving in store
  - Manage inventory and purchases for ingredients
  - Find profitability of store
  - Find sentiments from customers about store

- Business Owner:
    - Set business SLAs – order fulfillment time, max inventory levels, delivery charges, etc.
    - Manage stores – stores and store managers
    - Manage catalog of items serving in business and prices
    - Manage recipes and ingredients for preparing items
    - Find profitability of stores
    - Find sentiments from customers about business and stores

4. Quality Attributes

GFO needs to fulfill following quality expectations of stakeholders to offer a flawless, convenient and delightful experience:

- Availability
    - System should be available 24x7 to fulfill important workflows:
        - Receive orders from customers
        - Fulfill business activities inside stores – recipe, inventory, order and payment management
    - A downtime of "less than 3 minutes per day" can be tolerated.
    - System shall continue being available even during deployment, if possible.
- Security
    - System should be available only to authorized users reaching through internet cloud.
    - System should support role-based authorization.
    - System should support different authentication mechanisms for easy and intuitive onboarding experience:
        - Internal identity provider based on stored credentials
        - External authentication providers: Google, Facebook and Twitter
    - System should be deployed within the country for statutory data compliance requirements.
- Scalability
    - System should initially support workflow traffic happening across 2 existing outlets.
    - The workflow traffic is expected to increase gradually with an estimated growth plan as described below:

Year	# of cities (3 outlets per city)
1st	5
2nd	10
3rd	20
4th	40
5th	80

    - The number of customers is expected to grow to 80 lakhs in 5 years.
    - There should not be any impact in system response time due to such planned increase in workflow traffic.
- Reliability
    - System should 100% reliably capture shopping cart details even on client-session expirations.
    - System should 100% reliably capture orders once the cart is checked-out and payment is made.
    - System should 100% reliably track and notify order status.
    - System should 100% reliably alert on lower inventory levels to raise purchase orders on time.
- Accuracy
    - System should 100% accurately calculate prices, taxes and other charges for every order.
    - System should 100% accurately pass approved orders to kitchen queue in same sequence as received.
    - System should 100% accurately calculate inventory levels based on orders and wastages.
    - System should 100% accurately calculate profitability at store-level and at business-level.
- Latency
    - System should notify about order status updates within 2 seconds.

    - System should pass approved orders to kitchen queue with zero-time delay.
    - System should fetch customer profile, wish-list items and respective availability-in-store details within 1 second for front desk operator.
    - System should fetch catalog details and respective availability-in-store details within 2 seconds for customer.
- Observability
    - System should provide continuous updates about and ability to monitor system health.
    - System should automatically restart on any failures.
    - System should provide system-wide activity logs.

Any other attributes can be evaluated periodically and decided based on more understanding of requirements.

5. Constraints

- IT infrastructure – stakeholder wants to invest in IT infrastructure incrementally in concurrence to business expansion.
- Point of entry – customers shall be able to place orders from variety of devices – laptops, tablets and smart phones
- Point of service – front desk operators shall be able to place orders from point-of-sale devices and smart phones, kitchen managers shall be able to update orders from point-of-service devices and business owner and store managers shall be able to access system from laptops.
- Technology stack – modern trending development stack to be used for system to portray as technology innovator or pioneer in domain and to attract new tech savvy generation.

6. Architecture Principles

6.1 Microservices Architecture

The architecture proposes a microservices architecture implementation with multiple autonomous microservices in order to ensure:

- Services aligned to business capabilities – ownership of business capabilities can be with small focused teams
- Loosely coupled – clear microservices boundaries and interaction across boundaries through well-defined APIs
- Highly cohesive and isolated – each microservice owns its data and database
- Independently scalable – scale-out only certain required business capabilities on high demand
- Highly resilient and non-disruptive deployment

Each microservice is developed using .Net Core framework following MVC pattern. Polly is used to support resiliency measures using patterns like retries and circuit breakers.

Each microservice runs inside a separate container. Each database runs inside a separate container for isolation and resilience needs.

6.2 API Gateway / BFFs

API gateway

There are few concerns in exposing APIs from individual microservices directly to client:

- Typically, business capabilities to support a single functionality are spread across multiple focused microservices. For e.g. a functionality for placing an order involves the services namely “Catalog”, “Cart”, “Payment” and “Order”. APIs are very fine-grained in nature, while the client needs are much larger. Consequently, the client code that offers this functionality needs to fetch information by calling APIs from multiple services. This results in multiple roundtrips to server, thus increasing latency.

- When API endpoints change, client dependencies get disrupted and consequently need to be synchronized.

In order to address these concerns, the architecture places the APIs behind an API Gateway that takes care of fanning out a client request to multiple required microservices appropriately.

Since the API gateway acts on behalf of server by consolidating all APIs, it encapsulates all boilerplate functionalities like load balancing, authentication, authorization and more in a single place.

Backends for Frontends (BFFs)

There are few significant differences in supporting disparate devices like laptops and mobile devices:

- Data types – due to smaller size of screens, mobile devices tend to show a subset of data than laptops.
- Data formats – since only a subset of data needed, mobile devices can use efficient techniques like GraphQL than REST APIs.
- Network speeds – due to slower mobile network speeds, mobile devices prefer lesser requests and roundtrips.

In order to serve for these differences efficiently, the architecture defines a separate API gateway for each kind of client, in other words, a separate backend for each frontend – laptops and mobile devices. Each BFF provides a unique endpoint for its clients and then forwards the call to specific microservice or custom aggregator.

There are 4 BFFs – one for customer-facing shopping functionalities and another for internal business operation functionalities (like business and sentiments insights), one variant each for different clients.

The architecture proposes an implementation using Envoy to employ its advanced capabilities towards load balancing, health-check observability, dynamic routing, JWT and external authentications, role-based access controls.

Each BFF runs inside a separate container.

6.3 Communication

Angular Push Notifications

The architecture proposes to use Angular Push Notifications to notify about status updates and other business events.

Event-driven workflows

This system exchanges messages (or events) between microservices to establish an orderly, reliable and efficient communication mechanism. It uses publish/subscribe and topic-based communication through messaging channels of RabbitMQ to exchange these messages.

RabbitMQ runs inside a separate container.

Data interchange format

The architecture proposes JSON as the preferred data interchange format for major communications between microservices and between client/server.

6.4 Diagnostics and Observability

Logging

The architecture proposes a system-wide, centralized and structured logging using Serilog such that it becomes easy and robust to diagnose failures faster.

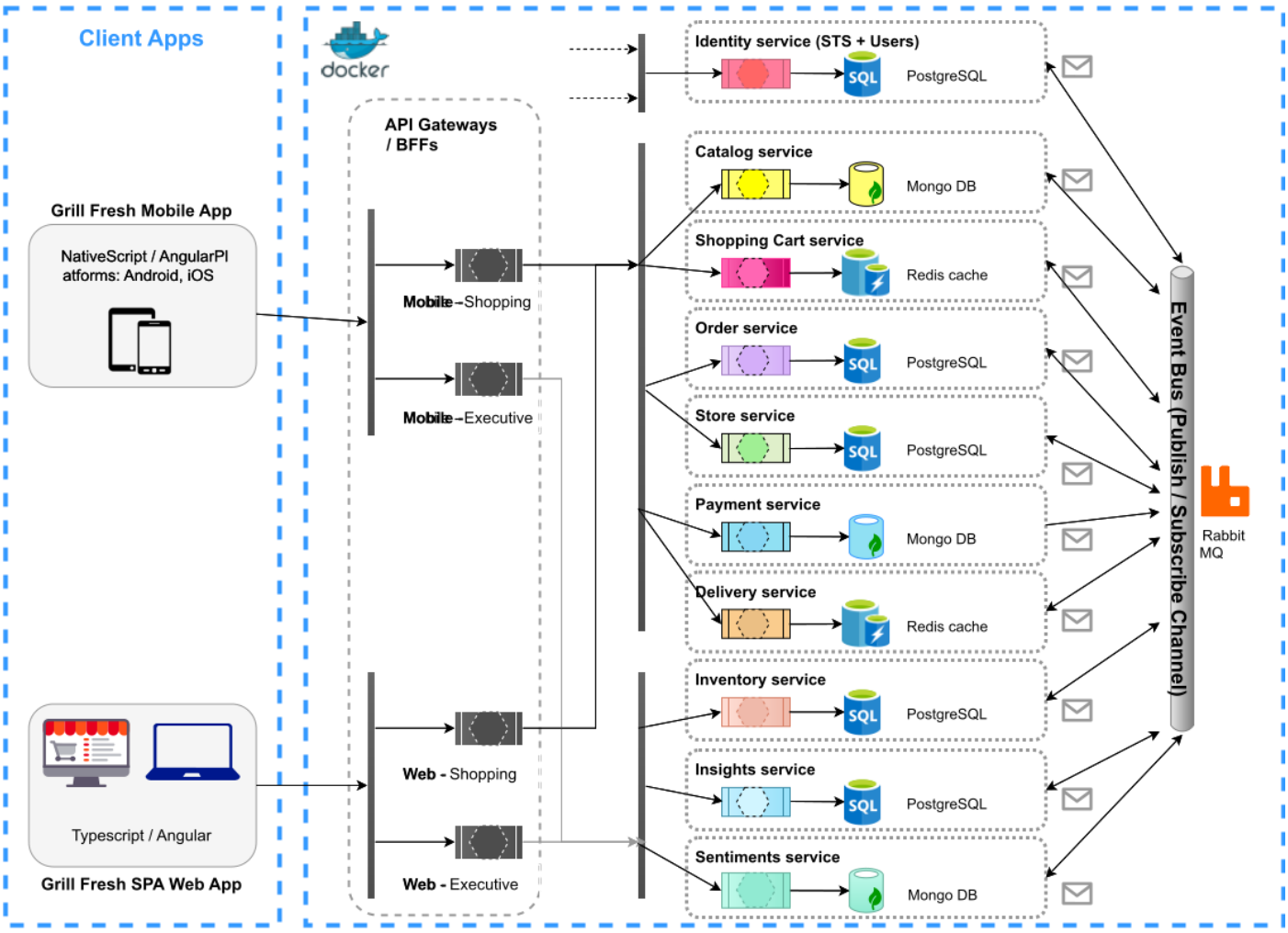
Serilog and logging service run together inside a separate container.

Observability

All applications and microservices expose endpoints that check the application and all its dependencies.

Health check APIs run on a separate container.

7. Architecture



7.1 Applications

The architecture proposes two variants of the application that can be respectively accessed from mobile devices and laptops.

SPA (Single Page Application) Web App

The architecture proposes to offer the laptop client app as a SPA style Web App. It is developed using Typescript in Angular framework.

Native Mobile App

The architecture proposes to offer the mobile client app for both Android and iOS platforms. It is developed using NativeScript in Angular framework.

Swagger UI (for Open API)

All the microservices allow anyone (with right credentials) to access the APIs. Swagger UI for a microservice can be accessed by navigating to the Swagger UI page for the respective microservice.

Each client application runs from a different container.

7.2 Shopping Gateway / BFFs

The shopping gateway organizes all the APIs required to fulfill customer-facing shopping functionalities. This aggregator serves into 2 BFFs, one variant each respective for 2 different client apps (mobile and laptop devices). The microservices under this gateway are described in this section below.

7.2.1. Identity Service

The architecture proposes to have a separate Identity Management Service. Identity STS (Security Token Service) generates a JWT per every successful login session. Auto logoff is activated on session expiry (set to 20 minutes presently).

The architecture proposes to implement JWT and external authentications and role-based access controls (claims-based) inside STS service. Also, it proposes to use Envoy at Shopping Gateway level such that it is straightforward to decide whether to forward the requests further based on the identity and role. At present, external authentications are supported for the customers based on Google, Facebook and Twitter authentications. Only security-token based authentication mechanisms are supported for internal employees.

The user profiles and roles are stored inside PostgreSQL database. The user credentials are encrypted before storing inside database.

7.2.2. Catalog Service

The catalog service serves two purposes:

- Managing the catalog of items – some items though available at business-level may not be available at individual store-level.
    - The business owner sets the catalog of items at the holistic business level.
    - The store manager may decide at store level to offer only a subset of items catalog specific to region.
- Managing the catalog of recipes for preparing those items.
    - The business owner sets the standardized recipes including ingredients for preparing items.
    - A kitchen manager browses through the catalog of recipes as standardized.

Mongo DB is used to store both catalogs of items and recipes.

7.2.3. Shopping Cart Service

This service manages all the user activities and business logic with respect to managing the shopping cart. This includes all activities just until placing an order. There are all sorts of combinations of:

- Pizza sizes (small, medium, large), pizza kinds (veg, non-veg), pizza range (value, luxury, exotic), pizza crust type (thin, hand-tossed, cheese-burst), pizza base (Maida, Wheat), pizza toppings (leaves, olives, jalapenos, etc.)
- Sides, desserts and beverages
- Items prices, taxes and delivery charges

Since, this service deals with transient data, it uses Redis cache. Persistence is turned on to restore the state in case of service crashes or failures.

7.2.4. Order Service

Order service comprises of two major functionalities:

- Order management
  - Persist order details, retrieve them when the customer asks about past orders (customer-to-order mapping), when the delivery person asks about delivery location (order-to-customer mapping).
  - The order details are stored inside a Mongo DB database, with sharding enabled for easier scaling purposes – the state identifier of the store location is used as the sharding key.
- Order fulfillment
  - Facilitate approval of order (by store front desk operator) and place approved orders in kitchen queue.
  - Maintain and notify all status updates for placed orders – order-placed, order-confirmed, in-preparation, ready, out-for-delivery, delivered
  - The order details along with status updates are stored inside Redis cache with persistence enabled.

7.2.5. Payment Service

The payment service supports various payment modes: Credit/Debit cards, UPI, Digital wallets, one-time payment URLs. At present, these payment modes are supported using one payment gateway provider. In future, it may engage services from more than one payment gateway provider.

This service persists details about payment transactions and respective states in Mongo DB database.

7.2.6. Store Service

This service manages all the store related functionalities:

- Managing serving radius and deliverable locations
- Adding, updating or deleting stores
- Managing store operational details (address, contact, business hours)
- Managing employees belonging to stores

The store related details are stored inside PostgreSQL database. The employee related details are stored inside separate PostgreSQL database.

7.3 Executive Gateway / BFFs

The executive gateway organizes all the APIs required to fulfill internal employees' / executives' needs related business operation functionalities – like inventory management, business insights and sentiments insights.

7.3.1. Inventory Service

This service attempts to manage the inventory levels of all ingredients at individual store level for the next "n" days, as defined by the store manager. It uses the combination of ingredients required for preparing various items and the expected amount of sales for the items in order to calculate the inventory levels for next "n" days.

Thus, it helps on a daily basis:

- The store manager to place purchase orders for the required ingredients at the right time.
- The store manager can add details about the inventory that went waste.

The inventory related details are stored inside a PostgreSQL database.

7.3.2. Insights Service

This service offers various analytics to aid and improve the growth of business. Various metrics of analytics are calculated:

- Profitability – to start with, it is initially calculated as: Sales – Inventory – Wastages – Operating expenses (like salary, etc.)
- Trends of Sales, Inventory and Wastages – simple trend curves and heatmaps across days in a month

The calculated details are stored inside a PostgreSQL database.

7.3.3. Sentiments Service

This service works on 2 kinds of events – direct customer feedback and social media ratings:

- Places a watch over social media looking for sentiments as expressed by various people, for example, likes, dislikes, comments – Facebook and twitter at present.
- Gathers the incoming sentiments events stream and looks for interested patterns and transforms them into ratings.
- Correlates the ratings with respect to past customer orders or other customer engagements

To start with, this service initially employs a simple brute-force pattern matching mechanism and thus it may not reflect accurate customer sentiments. In future, this service can employ any sophisticated AI based (like ML) correlation and pattern recognition algorithm for sensing the moods precisely.

The sentiments from event streams and ratings calculated from those events are stored inside a MongoDB database with sharding based on the store location enabled.

8. Sequence Diagrams

This section describes the major functional workflows happening in the system:

8.1 Sales workflow

8.2 Insights Analytics workflow

9. Design Diagrams

9.1 User Authentication and Authorization

9.2 Order Capture

9.3 Order Fulfillment

9.4 Inventory Management

9.5 Insights and Analytics

10. Deployment