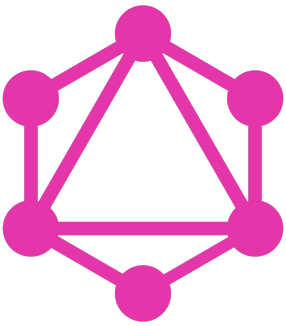
UNIVERSITÀ
DI TRENTO

Department of Information Engineering and Computer Science

Service Design and Engineering

Graph Market

Online e-commerce API in GraphQL



Students
Carlo Corradini
223811
Andrea Stedile
220930

Academic year 2020/2021

1 Introduction

Graph Market is an online platform that offers (some) E-commerce services. The name of this project is due the underlying GraphQL API, which we later describe.

In Graph Market, we identify three different actors:

- **Administrators**: They can create new products, also providing images and descriptions.
- **Sellers**: They advertise a product's availability.
- **(Unregistered) Users**: They browse for products (or search for them) and can create a new account that will be validated.
- **Registered users**: Users who have registered to Graph Market that are able to buy products and leave reviews.

Graph Market services are both offered via Graph Market's own website and exposed by its GraphQL API.

Using Graph Market means interacting with six different services:

- **Authentications**: This service handles user registration and the sign in and sign out procedures. When a user registers to Graph Market, they receive a pair of keys via Email and SMS that they need to provide to complete the registration.
- **Users**: Handles user creation, update, and deletion. A user is identified by their roles (i.e., they can be either a Seller or a normal User). Users can upload images of themselves as avatars.
- **Products**: Handles product creation, update of a product's description or photo, deletion.
- **Inventories**: Much like Amazon, each product has one or more corresponding inventories, allowing different sellers much like Amazon does. A product in an inventory can be in various statuses: brand new, renewed, used.
- **Purchases**: Handles purchases of products. Upon purchasing a product, a user receives a confirmation email. Given the scope of this course project and practical constraints, this is a fake service, which merely decreases the availability of a bought product by one.
- **Reviews**: Registered users can leave reviews to products. If the review is left in correspondence to a product they actually bought, the review has a verified badge.

Graph Market uses some external APIs to fulfil its own services. These services are available through a dedicated set of adapters that must be initialized during the bootstrap phase.

- **Twilio**: Allows to send SMS messages. It is used by the Authentications service in the registration process.
- **SendGrid**: Allows to send emails. It is used by the Authentications service in the registration process, and by the Purchases service to send purchases confirmations.
- **Cloudinary**: Allows to upload, store and deliver media assets. It is used by the Users service to store user avatars, and by the Products service to store product images.

2 Architecture

Graph Market has a fine-grained Service-Oriented Architecture. The specific architecture that Graph Market employs is called Apollo Federation. From Apollo Federation's website:

An Apollo Federation architecture consists of:

- A collection of **implementing services** that each define a distinct GraphQL schema;
- A **gateway** that composes the distinct schemas into a federated data graph and executes queries across the services in the graph.

Apollo Federation encourages a design principle called separation of concerns. This enables different teams to work on different products and features within a single data graph, without interfering with each other.

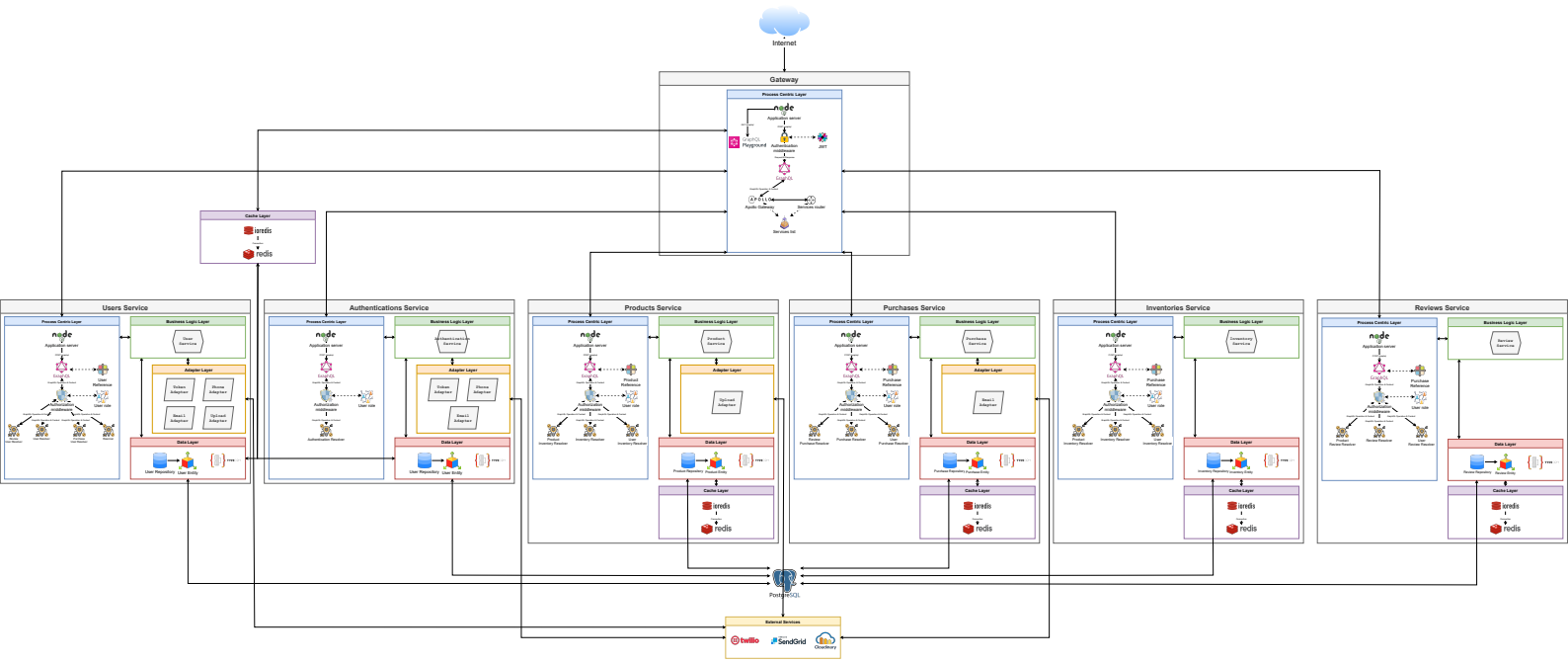
Each Graph Market service thus corresponds to its own Apollo Server. To visualize Graph Market's architecture with full resolution, we refer to the project's Github repository. The image depicts the various services, and clearly demarcates each service's layers (**Process-Centric Layer**, **Business Logic Layer**, **Data Layer**, **Adapter Layer**).

Each service has specific responsibilities. For example, the Products service is responsible for fulfilling requests about products; whereas the Purchases service fulfills requests about purchases, and so on.

Some services functionalities are available to authenticated (and authorized) users only. For example, obtaining the list of available products does not require any proper privilege (i.e., role), while obtaining the list of purchases of a particular user requires having a User role. This mechanism is implemented by JSON Web Token (JWT in short). The flow is as follows:

1. The Gateway accepts a new incoming request. If the request contains a JWT, it verifies the JWT using a secret key. If the JWT is valid and not expired, it forwards the request to the required services, setting a particular HTTP "user" header with the decoded JWT. The decoded JWT is, in practice, an array of strings.
2. If the request does not contain a JWT, the Gateway forwards the request anyways.
3. The involved service receives the request. If access to the requested resource does not need any particular authorization, they satisfy the request. Otherwise:
4. The service obtains the decoded JWT from the HTTP "user" header. If it does not find any JWT, then it rejects the request. Otherwise:
5. It extrapolates the user's roles from the JWT. If such a role is authorized to access the resource, it satisfies the request. Otherwise it rejects the request with an Authorization error.

In REST, securing specific routes or endpoints is possible. However, this is not the case for GraphQL, as the API is available through a single and unique endpoint `/graphql`.


Internet
Gateway
Process Centric Layer
Application server
PostgreSQL Playground
Authentication middleware
Jest
PostgreSQL
Apollo Gateway
Services router
Services list
Cache Layer
redis
Users Service
Authentications Service
Products Service
Purchases Service
Inventories Service
Reviews Service
Business Logic Layer
Adapter Layer
Data Layer
Cache Layer
PostgreSQL
External Services
twilio
SendGrid
Cloudinary