Part 3

In this assignment, we will create a set of APIs allowing other e-commerce web applications to retrieve the list of our shop products and advertise/sell it in their web site. Conversely, we will also provide an API that allows them to advertise/sell their products in our shop.

Introduction

Finally, it's the moment to talk business. *thin air LTD* plans to invest a lot of money in expanding their platform, and they have something quite complex in mind. To mimic their biggest competitor (notably e-bay) and try to attract more customers, they want to introduce their own idea of a marketplace. Long story short, on `https://thin-air.appsec.saarland` it will be possible to both buy and sell products. Registered users can buy products sold either from other users, or from *thin air LTD* themselves, or from partner business (hereinafter "partners"). This means that your e-commerce application now needs a second part in charge of managing the shop (products, orders, payments, and so on).

This is quite a lot to do. You are struggling with planning the implementation of the new features, and your colleagues give you the following tip: they heard that *thin air LTD* is struggling with partner businesses, who are actually also the investors of this project, and thus advise you to prioritize the "business to business" part of the application. This way, your next deliverable could help the company representative keeping the stakeholders at bay, and you could also be praised for your work. You accept, and now have to implement yet another part of the application...

In a first step, `https://thin-air.appsec.saarland` will issue an API key token to a requesting partner (a strictly confidential information!). This token will then authorize partners invoking APIs to either insert products from their shop, or delete products that they previously advertised and are now not available anymore. By using the token, partners can also retrieve the list of products registered in the marketplace, and the associated meta-information (e.g., if the price is currently discounted due to a promotional campaign); part of this information can then be used by Partners to advertise our products in their websites. These operations are security-sensitive and reveal confidential data, and thus only the HTTP requests that present a valid API key are authorized. There is no user interface required, as these services are not intended for physical users browsing the web site.

Since you managed to implement the SSO and password reset by yourself, your colleagues now entrust you with the implementation of this API key/access token to authorize requests from partners. Again, you must consider different possibilities, and decide which one of these best suits your setting. You need to rise the ranks the hard way...

Task 3.1 Database Models

Schema Description. Our shop will have a set of `Product`s for selling. These products come either from our own shop, from a customer of our shop, or from another shopping web site modeled by the `Partner` entity.

For this assignment, consider that the APIs are going to be used by `Partner`s only, and not by `User`s. This is also why you do not need to implement any user interface (HTML page), but only provide JSON data as a response when necessary.

Create the following entities in your database schema:

1. Partner
    - pk: primary key.
    - name: the name of the partner (e.g., e-bay).
    - web_site: the URL of the web site of the partner.
    - token: the API key for authorization.

2. Product
    - pk: primary key.
    - name.
    - description.
    - slug: a *slug* is a **unique** string auto-generated from the product name (by replacing whitespaces with hyphens) and a random suffix. It acts as a descriptive product identifier that you can use in the URLs of our application.
    - price: in Euros.
    - special_price: the minimum price that our shop can offer for selling this product (e.g., in special offers, etc). This price is confidential and is only shown to partners.
    - count: the amount available in stock.
    - image: the URL of the product's image.
    - seller: an identifier designating if this is a own shop product (set to 0, by default), or the seller is another customer (set to `u-<user-id>`), or another shop (set to `p-<partner-id>`, otherwise).

Please consider using the above-specified names as a binding naming convention (◀).

Task 3.2 Shop API

Your shop should provide the following APIs. Note that the data provided by the API (i.e., the HTTP response) contains confidential information that must not be leaked.

1. GET `/api/products?page=X&pagination=Y`◀: returns a JSON list of products. The results can be paginated. The query parameter `pagination` specifies the number of items in a page, and the query parameter `page` specifies the page number. Note that `X` and `Y` are parameters (i.e., an integer).
2. GET `/api/products/<product-id>`◀: fetch the details of a product.
3. DELETE `/api/products/<product-id>`◀: delete the specified product (if belongs to the requesting partner).
4. POST `/api/products/create`◀: creates a product. Product details must be specified in the request body using the corresponding key-value pairs, e.g., `name:<product-name>`, `description:<description>`, etc.

API Key. You should authorize API accesses with bearer tokens. A Bearer authorization grants access to the bearer of the token. A partner shop demanding API access will simply need to send the bearer token (that is previously provided by the API owner) with the `Authorization: Bearer <token>` request header. Try to find and pick the best approach to generate and validate the API key. For this assignment and for the purpose of testing your developed functionality, it is enough to write a function that generates an API key, creates a partner entry in the database, and associates the key to the partner. You should execute the function once to fill your database for testing purposes.

Note. Make sure to implement a proper access control with the API keys. API endpoints should respond with an error if the corresponding operation can not be done (e.g., deleting a product that does not exist). Make sure you use the correct set of HTTP headers on the returned response.