Part 4

In this assignment, we will add the functionality of purchasing a product from our e-commerce web application.

Context and introduction

You now have to implement the user-dedicated part of the e-commerce application. This means that additionally to the routines that provide the services, you need to take care of HTML pages as well as database bindings. After meeting with your client and your colleagues, you were able to draft a specification as follows.

The shopping area of `https://thin-air.appsec.saarland` is accessible to all users (logged in and non-logged-in users). The marketplace has a homepage which is accessible under `https://thin-air.appsec.saarland /shop/products/list`: here, all products are listed, from any seller. To make the user interface more compact and clean, not all the product details are shown, although all details are sent from the server to the client: the user can see them by clicking on the chosen product, which will open up a `HTML DIV` panel over the list; this will be handled in the client-side, without reloading the page. As asked by *thin air LTD* , each product must have a unique URL with all the details that people can share. Your seniors have asked you to reuse the same product list page and the `HTML DIV` panel for showing the product details, by storing the data that you need to show in the product panel in the URL hash fragment part (i.e., `/shop/products/list#<product-details>`).

Users can add or remove products from their cart by using specific buttons, each couple paired with its product in the list web page. Each product in the list has an "Add to Basket" button. When this button is clicked by a non-logged user, they are redirected to the login page, and after a successful login, the selected product will be added to their basket subject to its availability. For logged user, the selected product will be added to the basket upon clicking on the"Add to Basket" button depending on its availability. The total amount due for the shopping is shown in the top left corner of the page, and it automatically updates when items are added or removed; cart details (items, amount, price, ...) are shown at `https://thin-air.appsec.saarland /shop/products/basket/`.

Users with an ongoing order can proceed to the checkout page, by clicking on the checkout button at anytime. Here, they are asked to provide an address (at the moment, it is not relevant whether shipping and billing address are separated) and to choose a method of payment between the supported ones. If the data they provided is valid and correct, the address and payment information are finalized in the database, the cart is emptied and the order marked as placed, and – of course – money is automatically withdrawn from the provided payment endpoint. You noted down a comment from your client: he once had a startup (before it went bankrupt) where users managed to buy too many items of the same product, while actually it was unavailable! Somehow, they were seeing inconsistent views... and the refund process was a terrible headache.

The good news is that you can reuse part of the model that you already implemented (for partners and products); the bad news, well...

Task 4.1 Database Models

Schema Description. You can reuse the `Product` and `Partner` models. Additionally, you need to define `Payment`, `Address`, `Order` and `CartItem`.

When a user accesses the marketplace for the first time, and adds a product to his/her shopping basket/cart, they are automatically assigned an `Order`. The order must have the `placed` field marked as `False` by default, and no address, placement date, and payment details are associated with the order. This represents an empty shopping basket. When the user accesses the marketplace again, he/she can still see and modify the same `Order`, as long as it has not been placed.

When a user selects a `Product` from the list and adds it to the basket/cart, by clicking the specific button, a `CartItem` with `quantity=1` is added to the order. The user can add as many items as available, one at a time; items can also be removed one at a time, until there is no more of a certain product in the basket.

By clicking the checkout button, `placed` becomes `True`, and a placement date is associated with this entry (as well as payment and address information). When the user visits the marketplace again, and adds a new product to his/her shopping basket, similarly to the initial case, a new non-placed `Order` is created for him/her.

Thus, create the following entities in your database schema.

1. CartItem
   - pk: primary key.
   - product_id: the id of the product added to the shopping basket.
   - quantity: the amount/count added to the shopping basket.
   - order_id: specifies which order/shopping basket this `CartItem` belongs to.
2. Payment
   - pk: primary key.
   - amount: the amount that has been paid.
   - method: the payment method used (e.g., credit card, PayPal, etc).
3. Address
   - pk: primary key.
   - user: the `User` that this address belongs to.
   - street.
   - city.
   - zip_code.
   - country.
   - additional_info: additional address information such as unit number or C/O.
4. Order
   - pk: primary key.
   - customer_id: specifies the `User` that owns this basket.
   - placed: boolean specifying if the shopping basket of the order is purchased already.
   - date_placed: the date on which the shopping basket of the order is purchased.
   - shipping_address: the delivery `Address` for a purchased basket.
   - payment: the `Payment` information of the purchased basket.

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

Task 4.2 Purchasing a Product

Create the following web pages:

1. A list of shop products to purchase, accessible at: `/shop/products/list`◀. When the user clicks on an item, a pop-up should appear with the product details.
   - The URL `/shop/products/list#<product-details>`◀must open a pop-up window, read the product details specified in the URL hash fragment and show it in the pop-up `HTML DIV` panel. The parameter `<product-details>` is a `JSON` string containing the product slug, name, price, and available count, which will be shown in the pop-up window. This `JSON` string is constructed when the user clicks on an item in the list of products by the JavaScript code, and is stored in the URL fragment part.

2. A web page for viewing the user's current basket (i.e., non-finalized order) or finalized orders, accessible at: `/shop/basket/<order-id>`◀. Users must be able to only view their own orders. For the non-finalized orders, a checkout button must exist in the page to initiate the checkout process.

3. A checkout page to purchase the basket (i.e., finalizing the order), accessible at: `/shop/checkout/<order-id>`◀.

    - The checkout page must contain input fields for the shipping address and payment info (e.g., credit card number, etc), which should be set and written to the Document Object Model (DOM) by the client-side JavaScript code using, e.g., the `innerHTML` API.

Note. You can assume that the payment is always successful after clicking the checkout button.