HaaS Web Market: subsystem specification

Context

HaaS (short for "Hygine as a Service") is our proposition for a web market suited for regular purchases of essential products such as blades or shampoos. As a platform, it should work together with suppliers to provide customers the best possible price for bulk delivery with a schedule.

For scalability and maintainability reasons that will be discussed in further sections, the system should be developed in a microservices architecture fashion. Our group is responsible for the "Consumers, Orders, Delivery and Payments" subsystem.

Application requirements

The developed application as a whole should be able to provide basic web market functionality, which involves, at a high level, listing products, creating and paying orders.

The system must be designed from the start with a foreseeable high number of users. This means that the system should be able to handle a large number of concurrent requests, with a high degree of availability and reliability. Furthermore, the system shall be maintainable by a large company with a large number of teams, which means that the system should be modular and easy to understand. For all these reasons, the system should be developed in a microservices architecture fashion.

Non-functional requirements

- The application handles a large number of concurrent requests with a high degree of availability, preferably 99.99%+, which is equivalent to at most 53 minutes of downtime per year. Large companies such as Amazon and Google actually do better than that, with 5+ "nines" of availability, but such a high degree of availability will be difficult to test and achieve with the limited resources (such as time) available.
- The application is easy to maintain by a large company with a large number of teams. This means that the system should be modular and easy to understand, allowing refactoring of modules of the codebase independently.

Functional requirements

The following user stories are the ones directly related to the subsystem we are developing, and serve as the functional requirements for the application.

- As a Consumer, I want to be able to see the available products and stock so that I can purchase them later.
- As a Consumer, I want to be able to create/edit an order so that I am able to receive the products associated with it.
- As a Consumer, I want to be able to cancel my order.
- As a Consumer, I want to be able to add products with desired price to my shopping basket so that I can be notified if there is a money saving opportunity.
- As a Consumer, I want to be able to schedule an order so that I receive it with a given periodicity.
- As a Consumer, I want to be able to pay my order using a desired payment method.
- As a Consumer, I want to be able to get my order delivered to a desired location so that I don't need to leave the building.
- As a Consumer, I want to be able to create an auto-generated order with some of my desired products at their best price.

Architecture

Domain model

Our subsystem is responsible for orders and everything related to them. This includes the products that are ordered, the payment methods used to pay for the order, the delivery methods used to deliver the order, and the consumers that place the orders.

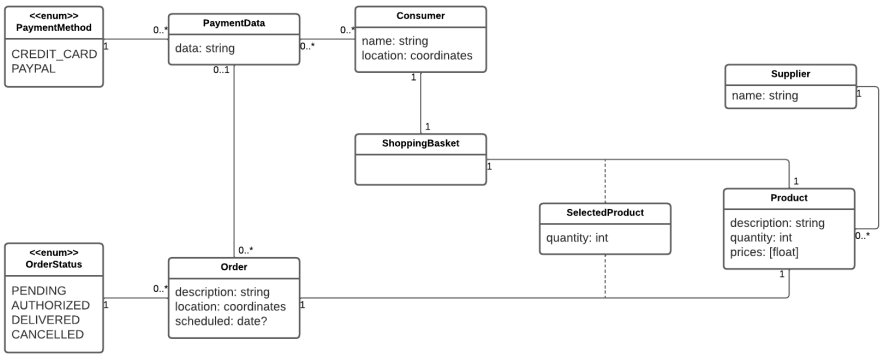

Figure 1: Domain model

While we are not responsible for *Supplier*, *Product* and *Category*, these are a direct requirement for the functioning of the subsystem and are therefore included here. This means that while other subsystems may model them differently, there

should be an interface between the subsystems that serializes them in a consistent way.

Services architecture

We are interested in the interactions between the services that we are responsible for, and the interactions between our services and the services of other subsystems (in this diagram, the connection to the *Stock Service*).

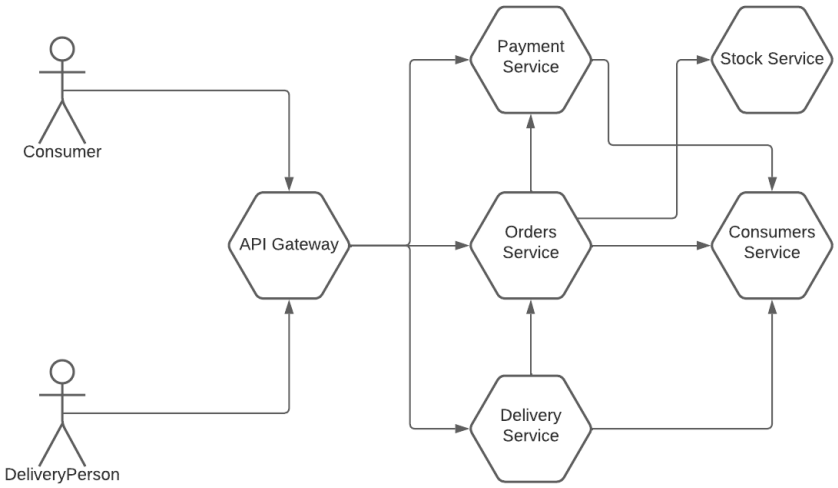

Figure 2: Services architecture

The (direct) connections may be intercepted, in the final system, by a gateway that will be responsible for routing the requests to the correct service and/or take care of authentication, authorization and observability. This will be skipped in early development stages and will be added later when the group responsible for the gateway delivers a working prototype that we are able to use.

Service operations and collaboration

Provided API operations

The proposed API aims to follow REST principles and be easy to use and understand for the client and other services. The operations are described below and are directly taken from https://haas-interaction.readme.io, where the API is documented.

Consumer Service This service aims to manage informations on the consumers using our platform.

It provides CRUD operations for clients' billing information and addresses as well as keeping the each client's current shopping basket.

This service also implements functionality to get the best products for a client at the best value, with alarms for price drops, recomendations on each product category based on consumer preference and habits and recommended shopping carts using consumer defined parameters. To do so it uses information from past orders, desired or/and requested products and dynamic pricing, leveraging sales.


```
{
  "openapi": "3.1.0",
  "info": {
    "title": "Consumer service",
    "version": "1.0.0"
  },
  "paths": {
    "/api/consumers": {
      "post": {
        "summary": "Create a new Consumer",
        "requestBody": {
          "required": true,
          "content": {
            "application/json": {
              "example": {
                "name": "John Doe",
                "email": "johndoe@example.com"
              }
            }
          }
        },
        "responses": {
          "201": {
            "description": "Consumer created successfully",
            "content": {
              "application/json": {
                "example": {
                  "consumer_id": 4
                }
              }
            }
          }
        }
      }
```

```
},
"/api/consumers/{id}": {
  "get": {
    "summary": "Get consumer information",
    "parameters": [
      {
        "name": "id",
        "in": "path",
        "required": true,
        "schema": {
          "type": "integer"
        }
      }
    ],
    "responses": {
      "200": {
        "description": "Successful response",
        "content": {
          "application/json": {
            "example": {
              "consumer_id": 4,
              "name": "John Doe",
              "email": "johndoe@example.com"
            }
          }
        }
      }
    }
  },
  "put": {
    "summary": "Update consumer information",
    "parameters": [
      {
        "name": "id",
        "in": "path",
        "required": true,
        "schema": {
          "type": "integer"
        }
      }
    ],
    "requestBody": {
      "required": true,
      "content": {
        "application/json": {
          "example": {
```

```
            "name": "Updated Name"
          }
        }
      }
    },
    "responses": {
      "200": {
        "description": "Consumer information updated successfully",
        "content": {
          "application/json": {
            "example": {
              "message": "Consumer information updated"
            }
          }
        }
      }
    }
  },
  "delete": {
    "summary": "Delete consumer information",
    "parameters": [
      {
        "name": "id",
        "in": "path",
        "required": true,
        "schema": {
          "type": "integer"
        }
      }
    ],
    "responses": {
      "204": {
        "description": "Consumer information deleted successfully"
      }
    }
  }
},
"/api/consumers/{id}/shopping-cart": {
  "get": {
    "summary": "Get shopping cart of a consumer",
    "parameters": [
      {
        "name": "id",
        "in": "path",
        "required": true,
        "schema": {
```

```
        "type": "integer"
      }
    }
  ],
  "responses": {
    "200": {
      "description": "Successful response",
      "content": {
        "application/json": {
          "example": {
            "shopping_cart": [
              {
                "product_id": 456,
                "name": "Product A",
                "quantity": 2
              },
              {
                "product_id": 789,
                "name": "Product B",
                "quantity": 1
              }
            ]
          }
        }
      }
    }
  }
},
"put": {
  "summary": "Update consumer's shopping cart",
  "parameters": [
    {
      "name": "id",
      "in": "path",
      "required": true,
      "schema": {
        "type": "integer"
      }
    }
  ],
  "requestBody": {
    "required": true,
    "content": {
      "application/json": {
        "example": {
          "shopping_cart": [
```

```
                {
                  "product_id": 456,
                  "quantity": 3
                },
                {
                  "product_id": 789,
                  "quantity": 2
                }
              ]
            }
          }
        }
      },
      "responses": {
        "200": {
          "description": "Shopping cart updated successfully",
          "content": {
            "application/json": {
              "example": {
                "message": "Shopping cart updated"
              }
            }
          }
        }
      }
    },
    "delete": {
      "summary": "Clear shopping cart",
      "parameters": [
        {
          "name": "id",
          "in": "path",
          "required": true,
          "schema": {
            "type": "integer"
          }
        }
      ],
      "responses": {
        "204": {
          "description": "Shopping cart cleared successfully"
        }
      }
    }
  },
  "/api/notifications/clients/{id}": {
```

```
"post": {
  "summary": "Register for price drop notifications on a given product",
  "parameters": [
    {
      "name": "id",
      "in": "path",
      "required": true,
      "schema": {
        "type": "integer"
      }
    }
  ],
  "requestBody": {
    "required": true,
    "content": {
      "application/json": {
        "example": {
          "product_id": 456,
          "threshold_price": 50
        }
      }
    }
  },
  "responses": {
    "201": {
      "description": "Notification registration successful",
      "content": {
        "application/json": {
          "example": {
            "message": "Notification registered"
          }
        }
      }
    }
  }
},
"get": {
  "summary": "Get registered price drop notifications",
  "parameters": [
    {
      "name": "id",
      "in": "path",
      "required": true,
      "schema": {
        "type": "integer"
      }
```

```
      }
    ],
    "responses": {
      "200": {
        "description": "Successful response",
        "content": {
          "application/json": {
            "example": {
              "notifications": [
                {
                  "product_id": 456,
                  "threshold_price": 50
                }
              ]
            }
          }
        }
      }
    }
  }
},
"/api/recomendations/clients/{users}": {
  "get": {
    "summary": "Get recommended products",
    "parameters": [
      {
        "name": "users",
        "in": "path",
        "required": true,
        "schema": {
          "type": "string"
        }
      },
      {
        "name": "category",
        "in": "query",
        "schema": {
          "type": "string"
        }
      }
    ],
    "responses": {
      "200": {
        "description": "Successful response",
        "content": {
          "application/json": {
```

```
            "example": {
              "recommendations": [
                {
                  "product_id": 123,
                  "name": "Recommended Product 1"
                },
                {
                  "product_id": 789,
                  "name": "Recommended Product 2"
                }
              ]
            }
          }
        }
      }
    }
  },
"/api/recomendation/clients/{id}": {
  "get": {
    "summary": "Get recommended shopping-cart",
    "parameters": [
      {
        "name": "id",
        "in": "path",
        "required": true,
        "schema": {
          "type": "integer"
        }
      }
    ],
    "responses": {
      "200": {
        "description": "Successful response",
        "content": {
          "application/json": {
            "example": {
              "shopping_cart": [
                {
                  "product_id": 123,
                  "name": "Recommended Product 1"
                },
                {
                  "product_id": 789,
                  "name": "Recommended Product 2"
                }
```

```
                ]
              }
            }
          }
        }
      }
    }
  }
}
```

Delivery Service This service communicates with an external service to orchestrate the delivery process of an order.

```
{
  "openapi": "3.1.0",
  "info": {
    "title": "Delivery service",
    "version": "1.0.0"
  },
  "paths": {
    "/api/delivery": {
      "post": {
        "summary": "Delivery request",
        "requestBody": {
          "required": true,
          "content": {
            "application/json": {
              "example": {
                "order_id": 3,
                "delivery_address": "123 Main St, City"
              }
            }
          }
        },
        "responses": {
          "201": {
            "description": "Delivery request successful",
            "content": {
              "application/json": {
                "example": {
                  "message": "Delivery request placed"
                }
              }
            }
```

```
          }
        }
      }
    }
  }
}
```

Orders Service This service manages all the orders for the platform, keeping a repository of past orders.

Orders might be on-demand, scheduled or created automatically given certain consumer-defined parameters (in the case of surprise boxes).

Order cancellation or updates are only allowed if the payment wasn't yet processed. After being authorized, the order supports only state updates.

```
{
  "openapi": "3.1.0",
  "info": {
    "title": "Orders",
    "version": "1.0.0"
  },
  "paths": {
    "/api/orders/clients/{id}": {
      "get": {
        "summary": "Get all the orders of a client",
        "parameters": [
          {
            "name": "id",
            "in": "path",
            "required": true,
            "schema": {
              "type": "integer"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "Successful response",
            "content": {
              "application/json": {
                "example": {
                  "orders": [
                    {
                      "order_id": 1,
                      "client_id": 123,
```

```
                  "status": "Pending"
                },
                {
                  "order_id": 2,
                  "client_id": 123,
                  "status": "Shipped"
                }
              ]
            }
          }
        }
      }
    }
  },
"/api/orders": {
  "post": {
    "summary": "Submit a new order",
    "requestBody": {
      "required": true,
      "content": {
        "application/json": {
          "example": {
            "client_id": 123,
            "products": [
              {
                "product_id": 456,
                "quantity": 2
              },
              {
                "product_id": 789,
                "quantity": 1
              }
            ]
          }
        }
      }
    },
    "responses": {
      "201": {
        "description": "Order created successfully",
        "content": {
          "application/json": {
            "example": {
              "order_id": 3
            }
```

```
            }
          }
        }
      }
    }
  },
  "/api/orders/{id}": {
    "put": {
      "summary": "Update an existing order (status update or cancellation)",
      "parameters": [
        {
          "name": "id",
          "in": "path",
          "required": true,
          "schema": {
            "type": "integer"
          }
        }
      ],
      "requestBody": {
        "required": true,
        "content": {
          "application/json": {
            "example": {
              "status": "Cancelled"
            }
          }
        }
      },
      "responses": {
        "200": {
          "description": "Order updated successfully",
          "content": {
            "application/json": {
              "example": {
                "message": "Order status updated to Cancelled"
              }
            }
          }
        }
      }
    }
  }
}
```

Payments Service This service communicates with a payment service to charge the client and accept an order.


```
{
  "openapi": "3.1.0",
  "info": {
    "title": "Payment service",
    "version": "1.0.0"
  },
  "paths": {
    "/api/payment": {
      "post": {
        "summary": "Order payment",
        "requestBody": {
          "required": true,
          "content": {
            "application/json": {
              "example": {
                "order_id": 3,
                "payment_method": "Credit Card",
                "total_amount": 100
              }
            }
          }
        },
        "responses": {
          "201": {
            "description": "Payment successful",
            "content": {
              "application/json": {
                "example": {
                  "message": "Payment processed successfully"
                }
              }
            }
          }
        }
      }
    }
  },
  "x-readme": {
    "explorer-enabled": true,
    "proxy-enabled": true,
    "samples-enabled": true
  }
}
```

Expected external interfaces

We are expecting other subsystems to allow, in a documented way, the following interactions that are necessary for the correct functioning of our subsystem:

- *Stock Service*: we need to be able to query the stock of a given product, and to be notified when the stock of a given product changes. This may be done using a message queue, for example.
- External payment service: we need to be able to pay an order using a given payment method.
- External delivery provider: we need to be able to programmatically request delivery of our orders using a delivery provider.