Software Applications Development
WorldSkills 2026 National Competition
HUNGARY

Round 3 - Session 1

Submitted by:

Skills IT

Contents

1. Introduction

In this session you have been provided multiple services to integrate with, on top of that you must develop exact APIs based on the provided swagger and this documentation.

1.1 Description of project and tasks

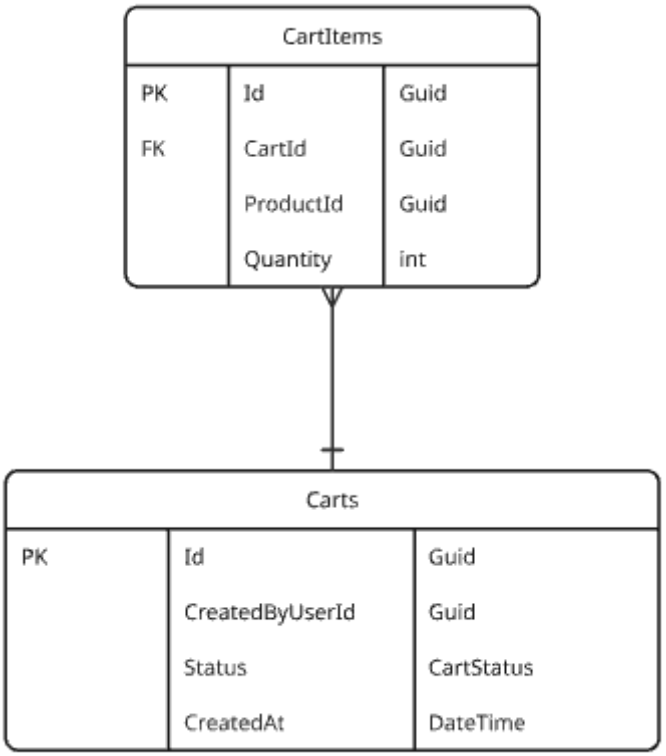
Your task is divided into two parts:

1. In the first part, you will have to create basic APIs to retrieve orders and items.
2. In the second part, you will create a cart functionality to place orders.

1.2 Database

You have been provided with a MySQL database, you must use this database. You can find connection information on the info page.

The database has the following structure. You must **not** change the schema and use this as-is.



1.3 How to submit your work

1. You have to submit all your in the git repository you have been provided with.
2. Your **README** file has to contain instructions on how to start your backend and frontend. We prefer you provide executable files or a built/deployed solution.

2. Part 1 – Basics

In this section, you need to create a backend in your chosen technology to expose existing information from API interfaces. This will be your utility features for the back office system, such as retrieving products and existing orders.

2.1 The user information

After a user authorizes themselves with the **auth-service/authentication/login** they are provided with a **Bearer token**, all front end implementations will provide you this **Bearer** token in the **Authorization** header in the following format:

Authorization: Bearer …token…

All APIs must validate that the person calling the API is allowed to access the resource. Using the **Authorization** header you must call the **auth-service/authentication/info API** this will return the user id and the role for the given authorization token.

GET auth-service/api/authentication/info
Response body**:**

```JSON
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "username": "…",
  "roles": [
    "…"
  ]
}
```

In case the authorization header is missing or the **auth-service** gives you an error of **401 Unauthorized** you must also return **401 Unauthorized** to the caller.

Using the **crm-service** you can retrieve the user details for the user provided by the **auth-service**.

GET crm-service/api/users/{id}
Response body**:**

```JSON
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "partnerId": "123e4567-e89b-12d3-a456-426614174000"
}
```

2.2 Products

Users want to see their current list of products and their stock. You must create an API that returns the list of products for the logged in user.

You can retrieve the list of products for the partner using the **crm-service:**

GET crm-service/api/products/partner?partnerId={partnerId}
Response body**:**

```
JSON
[
  {
    "id": "123e4567-e89b-12d3-a456-426614174000",
    "name": "…",
    "price": 1,
    "availableQuantity": 1,
    "stockQuantity": 1,
    "alertQuantity": 1
  }
]
```

API Schema

GET api/products
Response body**:**

```
JSON
[
  {
    "id": "123e4567-e89b-12d3-a456-426614174000",
    "name": "…",
    "availableQuantity": 1,
    "stockQuantity": 1,
    "price": 1,
    "warningType": "None"
  }
]
```

Warning type has 3 values: **None, LowStock, OutOfStock**

Using the stock quantity and alert quantity you must calculate the warning type. If the alert quantity is less than or equal to the stock quantity, then the warning type must be **None**. If the alert quantity is higher than stock quantity the warning type must be **LowStock**. In case the stock quantity is 0, then it must be **OutOfStock**.

2.2 Orders

Employees also want to see their orders and their statuses and also details. You must create an API that returns the list of orders for the logged in user.

You can retrieve the orders for a partner using the **erp-service:**

GET erp-service/api/orders/partner?partnerId={partnerId}
Response body**:**

```
JSON
[
  {
    "id": "123e4567-e89b-12d3-a456-426614174000",
    "partnerId": "123e4567-e89b-12d3-a456-426614174000",
    "orderedByUserId": "123e4567-e89b-12d3-a456-426614174000",
    "orderedAt": "2025-04-26T11:38:55.692Z",
    "status": "Created",
    "workerUserId": "123e4567-e89b-12d3-a456-426614174000",
    "priority": 1,
    "items": [
      {
        "id": "123e4567-e89b-12d3-a456-426614174000",
        "productId": "123e4567-e89b-12d3-a456-426614174000",
        "quantity": 1,
        "price": 1
      }
    ]
  }
]
```

You will also need to load all product names to create the response for your API, you can retrieve the product names via the **crm-service:**

GET crm-service/api/products/{productId}
Response body**:**

```
JSON
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "name": "…",
  "availableQuantity": 1,
  "price": 1
}
```

API Schema

GET api/orders

Response body**:**

```
JSON
[
  {
    "id": "123e4567-e89b-12d3-a456-426614174000",
    "orderedAt": "2025-04-26T11:39:40.293Z",
    "status": "Created",
    "totalPrice": 1,
    "items": [
      {
        "productName": "…",
        "quantity": 1,
        "price": 1,
        "totalPrice": 1
      }
    ]
  }
]
```

The price of each item is meant for one item, so for the total price use the following formula:

totalPrice = price * quantity

The total price for an order is the sum of the items' total price.

Status has the following values: **Created, Started, Gathering, Gathered, Delivering, Delivered**

3. Part 2 – Carts

In this section, you will have to create new APIs to handle cart functionalities. This will be used, so people can order available products.

The implemented responses must strictly follow the provided specification. If something is not specified, please refer to the example backend provided to you.

For all APIs you don't have to handle quantities zero and negative numbers.

3.1 Get cart API

A user will have at most 1 open cart at any given moment. This means that if a user has an open cart that cart must be used for all functionalities and if there is not an open cart available for the user, one must be created for the user. All modification APIs will return the same cart model (later used as "**cart model"**) which is the following:

```JSON
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "createdAt": "2025-04-26T11:39:40.293Z",
  "totalPrice": 1,
  "items": [
    {
      "id": "123e4567-e89b-12d3-a456-426614174000",
      "productName": "…",
      "quantity": 1,
      "price": 1,
      "totalPrice": 1
    }
  ]
}
```

The get cart API must retrieve the currently open cart for the user, as mentioned before, when there is no open cart for the user, one must be created for the user.

API Schema

GET api/carts
Response body: **cart model**

3.2 Add item to cart API

Now the user has a cart which he can interact with. A user must be able to add items to the cart. Each item in the cart is connected to a product. You must make sure that the product exists and is available, you can load the product details with the already known **crm-service** product api.

In case the user wants to add the same product to the cart, you must increase the existing item's quantity rather than adding a new item to the cart.

API Schema

POST api/carts/items
Request body:

```JSON
{
  "productId": "123e4567-e89b-12d3-a456-426614174000",
  "quantity": 1
}
```

Response body: **cart model**

If the product is missing or the product is not available in the requested quantity you must return a **400 Bad request** with the following content:

```JSON
"Product {productId} has insufficient quantity"
```

3.3 Update cart item quantity API

In case a user changes their mind about their ordered quantity they can change it using the update quantity API.

API Schema

PATCH api/carts/items/{id}
Request body:

```JSON
{
  "newQuantity": 5
}
```

Response body: **cart model**

If the item is not found on the open cart you must return **404 Not found** with the following content:

```
JSON
"Item {id} not found in cart"
```

If the product is missing or the product is not available in the requested quantity you must return a **400 Bad request** with the following content:

```
JSON
"Product {productId} has insufficient quantity"
```

3.4 Remove cart item API

A user must be able to remove an item from their cart.

API Schema

DELETE api/carts/items/{id}
Response body: **cart model**

If the item is not found on the open cart you must return **404 Not found** with the following content:

```
JSON
"Item {id} not found in cart"
```

3.5 Place the order API

After placing the items in the cart the user must be able to place the order in the **erp-service:**

POST erp-service/api/orders/place
Request body**:**

```
JSON
{
  "partnerId": "123e4567-e89b-12d3-a456-426614174000",
  "orderedByUserId": "123e4567-e89b-12d3-a456-426614174000",
  "items": [
    {
      "productId": "123e4567-e89b-12d3-a456-426614174000",
      "quantity": 1,
      "price": 1
    }
  ]
}
```

API Schema

POST api/carts

If the placing of the order is successful you must close the cart and return **204 No Content.**

If the cart is empty you must return a **400 Bad request** with the following content:

```
JSON
"Cart is empty"
```

If any of the products are not available in the requested quantity you must return a **400 Bad request** with the following content (you can return with the first item/product):

```
JSON
"Product {productId} has insufficient quantity"
```

6. Additional information

- Some media, icons and text have been provided for you in the media files. You are free to use these, but you can also create your own, as long as the application is still fit for purpose. **You should not use any other media files (e.g. downloaded videos, images, icons, etc.)**.

- Clean code and user interface accessibility are also important considerations.
- Do not hardcode API responses as another database will be used for testing.