IMS Evaluation Application

You'll be provided with a zip archive containing the source code for an IMS inventory application. The goal is to implement and/or fix portions of it so that it becomes usable. Complete as many of the provided tasks as possible. These tasks can be found in the Tasks section, if you don't need/want to setup a server to complete this exercise you can skip the Setup web server section below. However, by setting up the server it may aid in visualizing the problem and validating/testing your solutions.

Rules for the exercise:

1. JavaScript and/or Typescript may be used
2. Don't inline JS/TS scripts or CSS style sheets in the HTML/Twig templates, use the JS/TS/CSS files provided and/or create any required files in the asset directory
3. Any useful JS/TS libraries can be installed via npm, pnpm, yarn, bun, etc...
4. Any useful PHP libraries you need can be installed via composer
5. Try to utilize tools and functionality provided by the Symfony framework in your solutions where possible
6. Be sure to format your code changes
7. `composer install` and `asset-map:compile` commands should execute successfully when all tasks are complete
8. Tests are not required

To get started, extract the contents of the zip to a directory on your system, optionally setup the web server and then work on completing the Tasks given.

This exercise is expected to take 1-2 hrs provided you are familiar with PHP and Symfony but should take up to 3 hrs if not. Your contact person from IMS will provide you with the time and dates required for completion of the exercise

Setup for web server (Optional)

Ensure the following are installed on your system:

1. PHP (8.3 or greater) - can manually download at PHP Downloads
2. Composer - can manually download at Composer
3. Symfony CLI - can manually download at Symfony CLI

Once PHP is installed, update its configuration to enable the following extensions

- sqlite extension
- ctype extension
- iconv extension

You can run the `symfony check:requirements` command to validate your install

```
# Navigate to root directory of this application
cd <this_application_dir>
```

```
# Install symfony libraries
composer install

# Compile assets
php bin/console asset-map:compile

# Create database in the /var directory
php bin/console doctrine:schema:create -v

# Start symfony server
symfony serve
```

> Changes to JavaScript/Typescript and CSS files will require that you re-run the `php bin/console asset-map:compile` in order to see the changes.

Required tasks

Git

1. In the application directory initialize a new git repository
2. Add all files in the application directory to git and commit the changes with the message "First IMS commit"
3. Create and switch to a new branch called `ims`
4. All commits from this point should now go into this `ims` branch

Inventory

1. Use JavaScript/Typescript to update inventory list count on the `inventory` page, update the header element with the text '(x)'
2. If an inventory entry description contains the string `aut` (case insensitive) anywhere in the description, make the entire description text bold for that entry
3. If the weight of an item is greater than 50 **pounds** change the color of the text to red (#851a31)
4. Change the color of the text for length, width and height to blue (#054069).
5. Make all SKUs in the table **bold**

Suppliers

1. Add the following fields and their related getters/setters to the supplier entity:
    - phone (string, nullable) - supplier's phone number, matching the pattern 'xxx-xxxx' where x is a single digit between 0-9, add validation for this
    - email (string, nullable) - the supplier's email
    - website (string, nullable) - the supplier's website
    - country (string, nullable) - the country where the supplier resides

```
# If using the web server, update the database
# Navigate to root directory of this application
cd <this_application_dir>
# Can check if your changes are valid
php bin/console doctrine:schema:validate
```

```
# Update database with changes
php bin/console doctrine:schema:update --dump-sql --force
```

2. Implement the index method in the supplier controller by having the route display a page with a similar layout to that of the index method of the `inventory` page but instead lists the suppliers and all of the entity's fields in a table.
3. In addition to the supplier fields include the following field in the supplier table list,
    - totalItemCount (int) - the number of items associated with the supplier
    - totalItemCost (float) - the total cost of the items associated with the supplier
4. Use JavaScript/Typescript to update supplier list count on the new created `supplier` page to update the supplier list header to show the count similar to what was done in `task #2`.
5. If supplier phone number is invalid(does not match pattern 'xxx-xxxx') change the color of the text to red (#851a31)

API

1. Create the following API routes in the `ApiController` class matching the corresponding methods given:
    - [GET] /api/inventory - list of all `inventory` items with all their fields, returns JSON with 200 HTTP code
    - [GET] /api/suppliers - list of all `suppliers` with all their fields, returns JSON with 200 HTTP code
    - [GET] /api/suppliers/{supplier_id} - get the `supplier` with all its fields that matches the given supplier_id
        - if the supplier_id does not exist return a 404 HTTP error code
        - if the supplier exists return JSON of the supplier with all the fields present with 200 HTTP code
    - [POST] /api/suppliers - create new `supplier` with the body of the request requiring the name of the new `supplier`,
        - returns JSON of the new supplier with 201 HTTP code
    - [PATCH] /api/suppliers/{supplier_id} - edit the 'name' and/or 'phone' fields of the `supplier` with the matching supplier_id
        - if there is a name entry in the body and it is null or it is a string and equals '' return an adequate error in JSON with a 400 HTTP error code
        - if there is a phone entry in the body and it does not match the validation pattern in the entity return an adequate error in JSON with a 400 HTTP error code
        - if the name and/or phone was changed return no content with a 204 HTTP code
        - if both the name and phone was not changed returns no content with a 204 HTTP code
    - [DELETE] /api/suppliers/{supplier_id} - delete `supplier` with the matching supplier_id, a cascading deletion of related entities is fine here.

- if the supplier does not exist return JSON with a suitable message and a 404 HTTP error code
- if the deletion was successful, return no content with a 200 HTTP code

Git (Final step only attempt prior to submission)

It is advised to create a copy of the application directory just in case something goes wrong here

1. Ensure that all your changes are committed to the `ims` branch
2. Merge your `ims` branch into the `main/master` branch without doing a fast forward
3. Tag the most recent commit on the `main/master` branch with the value `final`

Submit your project

Create a zip archive of application directory without the `node_modules`, `vendor`, `var` directory present. Then return that archive to the IMS contact person.