1. Entity model

Apart from the requested classes a new **BaseEntity** is added to mark all entities with create and update timestamps. If user authentication was implemented it would be desirable to also include info about who created and modified each entity.

This is done with doctrine extensions Timestampable and blameable.

There is an eternal discussion about if exposing an auto-numeric id is more insecure or not but what it is certain is that the number of records in a table is private information that should never be exposed to a normal user.

A public uuid is used to identify every resource and a private autonumeric-id is used to handle all internal relations. With this approach we have the speed of an integer FK and the privacy of an uuid.

Since the relation between projects and employees is ManyToMany and an extra attribute (role) is needed to describe the relation, a new entity **EmployeeProjectRelation** is created.

2. Project dependencies

This is the list of dependencies installed with composer

- **symfony/orm-pack** - Doctrine ORM
- **doctrine/annotations** - Needed to describe how the entities will be reflected in the database
- **stof/doctrine-extensions-bundle** - Doctrine extensions for the Timestampable
- **api** - Api platform
- **orm-fixtures (dev)** - needed for generating data fixtures
- **fzaninotto/faker** - used for generating data fixtures with more realistic data
- **symfony/phpunit-bridge (dev)** - for testing purposes
- **symfony/browser-kit (dev)** - for functional tests
- **dama/doctrine-test-bundle (dev)** - for making sure the database is not modified after running the tests

3. Api

The entities CRUD operations are handled with api-platform and three extra endpoints were implemented within the EmployeeController class

This is the whole list of the methods available

```
root@f254b3b1020f:/var/www/html# php bin/console debug:route
 ------------------------------------------------- -------- -------- ------ ----------------------------------------------
  Name                                              Method   Scheme   Host   Path
 ------------------------------------------------- -------- -------- ------ ----------------------------------------------
  _preview_error                                    ANY      ANY      ANY    /_error/{code}.{_format}
  _wdt                                              ANY      ANY      ANY    /_wdt/{token}
  _profiler_home                                    ANY      ANY      ANY    /_profiler/
  _profiler_search                                  ANY      ANY      ANY    /_profiler/search
  _profiler_search_bar                              ANY      ANY      ANY    /_profiler/search_bar
  _profiler_phpinfo                                 ANY      ANY      ANY    /_profiler/phpinfo
  _profiler_search_results                          ANY      ANY      ANY    /_profiler/{token}/search/results
  _profiler_open_file                               ANY      ANY      ANY    /_profiler/open
  _profiler                                         ANY      ANY      ANY    /_profiler/{token}
  _profiler_router                                  ANY      ANY      ANY    /_profiler/{token}/router
  _profiler_exception                               ANY      ANY      ANY    /_profiler/{token}/exception
  _profiler_exception_css                           ANY      ANY      ANY    /_profiler/{token}/exception.css
  app_employee_listemployees                        GET      ANY      ANY    /api/employees_raw
  app_employee_getemployeescsv                      GET      ANY      ANY    /api/employees/csv_export
  app_employee_postemployeescsv                     POST     ANY      ANY    /api/employees/csv_import
  app_test_test                                     GET      ANY      ANY    /test
  api_entrypoint                                    ANY      ANY      ANY    /api/{index}.{_format}
  api_doc                                           ANY      ANY      ANY    /api/docs.{_format}
  api_jsonld_context                                ANY      ANY      ANY    /api/contexts/{shortName}.{_format}
  api_companies_get_collection                      GET      ANY      ANY    /api/companies.{_format}
  api_companies_get_item                            GET      ANY      ANY    /api/companies/{id}.{_format}
  api_employees_get_collection                      GET      ANY      ANY    /api/employees.{_format}
  api_employees_post_collection                     POST     ANY      ANY    /api/employees.{_format}
  api_employees_get_item                            GET      ANY      ANY    /api/employees/{id}.{_format}
  api_employees_project_relations_get_subresource   GET      ANY      ANY    /api/employees/{id}/project_relations.{_format}
  api_employee_project_relations_get_collection     GET      ANY      ANY    /api/employee_project_relations.{_format}
  api_employee_project_relations_get_item           GET      ANY      ANY    /api/employee_project_relations/{id}.{_format}
  api_projects_get_collection                       GET      ANY      ANY    /api/projects.{_format}
  api_projects_get_item                             GET      ANY      ANY    /api/projects/{id}.{_format}
 ------------------------------------------------- -------- -------- ------ ----------------------------------------------
```

3.1 Api-platform

Using entity annotations for api-platform definitions can get messy really quick! Seeing entity files with 3-4x more api annotation code than php code is not uncommon so using yaml files is recommended.

These yaml files live in the config/api_platform folder and define the exposed operations and how they behave.

Available operations

- **Company**: getItem, getCollection
- **Employee**: getItem, getCollection, postItem, getProjectsSubResource
- **Project**: getItem, getCollection
- **EmployeeProjectRelatioin**: getItem, getCollection

All these operations define which serialization groups to use in both serialization and deserialization for reading data back from the user.

Using the right serialization groups is crucial for performance as nested entities can grow really quick and the serialization process become too slow

All get-collection operations are paginated and special attention is put in not serializing nested entities with OneToMany or ManyToMany relations. This is, in general, a good practice as you can never be 100% sure about the number of entities that are hanging from the one you are serializing.

To deal with these kind of cases an entity sub-resource is the way to go and you can see one example in employee projects

You can get an employee with the api endpoint
'/api/employees/{employeeId}'

You can get the projects the employee is working in with the subresource
'/api/employees/{employeeId}/project_relations'

The fetched subresources contain the role and the projects this user is working in.

Also, pay attention to how the employee is validated in the post operation. Apart from constraint annotations an **EmployeeContraintValidator** class is executed every time the post employee method is called.

3.2 Api controllers

You can also mix api-platform with more classic symfony controllers, especially when not using common CRUD operations.

Three methods were developed within the EmployeeController:

- ***'api/employees_raw'*** : developed to meet the requirements of the test, this endpoint returns a list of all employees in the database.
- ***'api/employees/csv_export'***: exports a csv file following the given format
- ***'api/employees/csv_import***': Imports a csv file following the given format

The workflow here would be that the EmployeeController gets the needed parameters from the http request, fetches the required entities from the repositories and passes the request data and entities to the EmployeeManager which is responsible for all the logic regarding the Employees.

Once the manager has finished it returns the result of the operation to the controller which generates a http response.

To meet the format requirements custom normalizers and denormalizers were implemented for date format handling

4. Data Fixtures

Orm-fixtures and fzaninotto/faker bundles were used to create all dummy data in the database.

This is done in the src/DataFixtures/AppFixtures.php file and you can populate both the normal and test databases running the command (within the php docker container)

```
php bin/console doctrine:fixtures:load
php bin/console doctrine:fixtures:load --env=test
```

5. Testing

Both unit and functional tests were provided for this exercise. They live in the tests folder and can be executed with the command (within the docker container)

```
php bin/phpunit
```

A separate database is used for testing and symfony is configured so it works with this database while working in the testing environment.

The test count should be higher in a working project but this is just to showcase a practical example of how it can be done.

5.1 Unit testing

Basic unit testing is done for two utils classes, DateTimeUtilsTest and StringUtilsTest.

5.2 Functional testing

For the functional tests we are going to hit company, employee and projects methods.

The **company** and **project** tests both check that:

- The first page of the collection can be fetched and at least one element is returned
- A single item can be fetched and decoded from the given json. Also checks that the name decoded from the json matches the one stored in the database
- Try to fetch an item we know it does not exist and returns a 404

The **employee** tests are a little more extensive and they make sure that:

- ‘api/employees_raw’ method responds ok and returns all employees from the

- The first page of the collection can be fetched and at least one employee is returned. This is the api-platform get collection method.
- The post employee method is tested with multiple valid and invalid case scenarios
- Download the CSV file from ‘api/employees/csv_export’, check the file has as many lines as employees are in the DB and upload the file again without errors.
- Upload csv employees file
    - Call the method without any file and fails
    - Upload a file with a single unmodified employee and succeeds
    - Upload a file with a single employee with a modified email and fails
    - Upload a file with a single employee without a company and fails

6. Elastic search ELK Stack

6.1 Elastic Search

There are many ways to store your logs in elastic search.

The first attempt was to connect symfony monolog output directly with elastic and while it worked it wasn't the best option.

The log message formatting had to be done in symfony and with every user Request a connection with elastic had to be done.

If we have a high traffic website with hundreds of requests per minute, well.... that's a lot of connections!

6.2 LogStash

In a second iteration logstash was installed. It is supposed to behave like an elastic search front door.

It can handle input messages from many sources, apply formatting filters to these messages and send them to elastic.

The message formatting was not done in symfony anymore and now elastic can receive data from many sources.

Even if all these sources have different formats, we can apply message filters in logstash and normalize them all.

One problem though. We still have to make a connection with logstash for every request and symfony still knows "too much" about Elastic.
A more transparent solution would be desirable.

What if can achieve the same result without symfony knowing anything about logstash-elastic anymore?

6.3 FileBeats

All monolog customizations were removed and now we configure symfony to use it's default log configuration. That is storing everything in plain log text files.

Install Filebeats and let it monitor these files and send all the changes to logstash.

Symfony does not know anything about elastic anymore, it's 100% decoupled and if desired, we can totally remove or replace elastic logging without symfony even noticing.

We dont need another symfony-logstash connection per request and performance penalties are all removed. (Filebeats by default sends new logs to Logstash every 10 seconds)

6.4 Kibana

Kibana is also added to the stack to manage all elastic data in a comfortable way.

You can try it out in your browser with 'localhost:5602'.

Hit some endpoints with the provided postman collection and take a look at "observability/logs" to check how your system is behaving and to make queries of your logs.

7. Closing comments

The main goal to include elastic in the stack was to save logs but also a small integration was added for our entity model.

The BaseEntityManager now has a ‘store’ method that after validating and storing an entity in our database, it sends it to elastic search.

This is used in the fixture data generation and with a data persister in the "POST employee" api-platform endpoint.

If an entity is updated through a CSV file upload these changes are currently **not** reflected in elastic.

A new endpoint was also added to try out one of the more interesting features in elastic search, the fuzzy search operations.

/api/employees/fuzzySearch?searchQuery=???

This method allows the user to make fuzzy search queries for the firstName of the employee. Try to look in the database for one employee firstName and pass it as the “seachQuery” parameter changing one letter. It will return the user even if it is not a perfect match.