Coding Assignment

Sections

1. Create a Symfony application (version 4 or 5) using symfony/skeleton
    a. This should **NOT** be via symfony/website-skeleton
2. Use Symfony Security to create form-based authentication
3. Expand upon the authentication system from step two to support authentication in API calls
4. Build the secured homepage to support multiple tenants determined by a query string value

Section 1

Use whichever method you have available to you to create a new symfony application using the *symfony/skeleton* template.

This should **NOT** be done using the *symfony/website-skeleton* for the purpose of this test.

Section 2

Using the Symfony Security library, create a login form and require authentication before going to a secured second page. Store your users in the *users* table found in the provided database.

Section 3

Using Authenticators, extend your security configuration to support authentication using an API key. The preferred method of including the authentication key should be via the *Authorization* header.

Section 4

Section 4.1

Create a means by which each request dynamically sets which database doctrine should connect to at runtime based upon a query-string variable *tenant*. Use the value passed via the query string as a lookup value to search in the *directory.tenants* database by the name column, and use the *tenant_db* column to define which database to connect to.

Section 4.2

Create an API that is only accessible to authenticated users (using the API key authentication method from **section 3**) that takes a *tenant* query string value (which will be used by the code implemented in **section 4.1**) and returns an array of *categories* with their associated *products*.

Section 4.3

In the secured page after authenticating, use javascript to call the API, load the data, and display it in a table on the secured page.

Bonus: Add a dropdown on the page which allows you to toggle between the available tenants, selecting a tenant will call the API and reload the data

Section 4.4

Add a simple webform that supports creating a new product, this webform does not need to be styled and can be on a separate webpage with a basic post request flow, but should allow for adding a new product to the database specified by the selected tenant, and contain enough feedback to indicate success/failure states.

Submission

Preferred form of submission is a link to a git repository (github/bitbucket etc) that can be viewed by the assessor. If this is not available, a zip file containing the codebase as well as a README text file containing any required instructions on running the codebase.

Resources

- **Symfony Security Documentation**
  - https://symfony.com/doc/current/security.html
  - https://symfony.com/doc/current/security/guard_authentication.html

Database Connection Information

USERNAME	testuser
PASSWORD	bghnla2ecy60aph6
HOST	insite-interviewing-db-do-user-1938513-0.b.db.ondigitalocean.com
PORT	25060

Note: you only have access to the following databases with the listed permissions.

- directory
  - SELECT
- tenant_one
  - SELECT, INSERT
- tenant_two
  - SELECT, INSERT

Recommendations

1. Do the best you can, you are not required to complete every step and will be assessed on the quality of work submitted, not how much work is submitted.
2. We want to get an idea of your thought process, comments which give insights into the choices and decisions you make will help with that.
3. The database schema is provided and pre-defined, build your doctrine models to match the schema. **DO NOT CREATE YOUR OWN.**
4. **Things to avoid**
    a. Database queries executed in the controller
    b. Not using separation of concerns
    c. Lack of code structure or design patterns
    d. Large number of items passed in via constructor injection
    e. Use of raw strings over constants and variables
    f. variables/methods/classes with confusing names
    g. Not using consistent code style
5. **Things we will mark highly**
    a. Unit tests
    b. Separation of concerns
    c. Good code hygiene (DRY etc)
    d. Good use of data structures
    e. Strong use of types

Database Definitions

directory

```
CREATE TABLE "users" (
  "id" int NOT NULL AUTO_INCREMENT,
  "email" varchar(64) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
NULL,
  "password" varchar(128) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
NOT NULL,
  "api_key" varchar(128) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
DEFAULT NULL,
  "date_created" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  "last_updated" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  "enabled" bit(1) NOT NULL DEFAULT b'1',
  PRIMARY KEY ("id"),
  UNIQUE KEY "idx_username" ("email"),
  UNIQUE KEY "idx_username_pass" ("email","password"),
  UNIQUE KEY "idx_apikey" ("api_key"),
  KEY "idx_enabled" ("enabled")
);
```

```
CREATE TABLE "tenants" (
  "id" int NOT NULL AUTO_INCREMENT,
  "tenant_name" varchar(256) CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci NOT NULL,
  "tenant_db" varchar(128) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
NOT NULL,
  "date_created" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  "last_updated" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  "enabled" bit(1) NOT NULL DEFAULT b'1',
  PRIMARY KEY ("id"),
  UNIQUE KEY "idx_name" ("tenant_name"),
  KEY "idx_enabled" ("enabled")
);
```

tenant_xxx

```
CREATE TABLE "categories" (
  "id" int NOT NULL AUTO_INCREMENT,
  "name" varchar(256) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
NULL,
  "date_created" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  "last_updated" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  "enabled" bit(1) NOT NULL DEFAULT b'1',
  PRIMARY KEY ("id"),
  UNIQUE KEY "idx_name" ("name"),
  KEY "idx_enabled" ("enabled")
);
```

```
CREATE TABLE "products" (
  "id" int NOT NULL AUTO_INCREMENT,
  "name" varchar(256) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
NULL,
  "price" decimal(10,0) NOT NULL,
  "category_id" int NOT NULL,
  "date_created" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  "last_updated" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  "enabled" bit(1) NOT NULL DEFAULT b'1',
  PRIMARY KEY ("id"),
  UNIQUE KEY "idx_name" ("name"),
  KEY "idx_enabled" ("enabled"),
  KEY "category_id" ("category_id"),
  CONSTRAINT "fk_category_id" FOREIGN KEY ("category_id") REFERENCES
"categories" ("id") ON DELETE CASCADE ON UPDATE CASCADE
);
```