Mini-projet Symfony

Easycubi

Elaboré par : Ahmed Dachraoui

Année universitaire : 2023/2024

Classe : L2DSI3-G1

1. Description de notre commerce (EasyCubi) :

EasyCubi : Votre Boutique de Rubik's Cube en Tunisie

EasyCubi est une boutique en ligne dédiée aux passionnés de Rubik's Cube en Tunisie. Nous offrons une large sélection de cubes pour tous les niveaux, des débutants aux experts. Que vous soyez à la recherche d'un défi stimulant ou simplement d'un moyen amusant de passer le temps, EasyCubi a le cube parfait pour vous.

Pourquoi choisir EasyCubi ?

- **Large sélection de cubes:** Nous proposons une large gamme de Rubik's Cubes, des modèles classiques aux puzzles les plus complexes. Vous trouverez forcément le cube qui vous convient.
- **Prix compétitifs:** Nous nous engageons à vous offrir les meilleurs prix sur les Rubik's Cubes en Tunisie.
- **Livraison rapide et fiable:** Nous expédions votre commande dans les plus brefs délais et vous assurons une livraison fiable.

Un marché en plein essor

Le Rubik's Cube est un jeu de puzzle populaire qui connaît un regain d'intérêt en Tunisie. De plus en plus de personnes se passionnent pour ce jeu stimulant et addictif. Cependant, il existe un manque de boutiques spécialisées dans la vente de Rubik's Cubes en Tunisie. C'est pourquoi EasyCubi est une opportunité unique de répondre à une demande croissante.

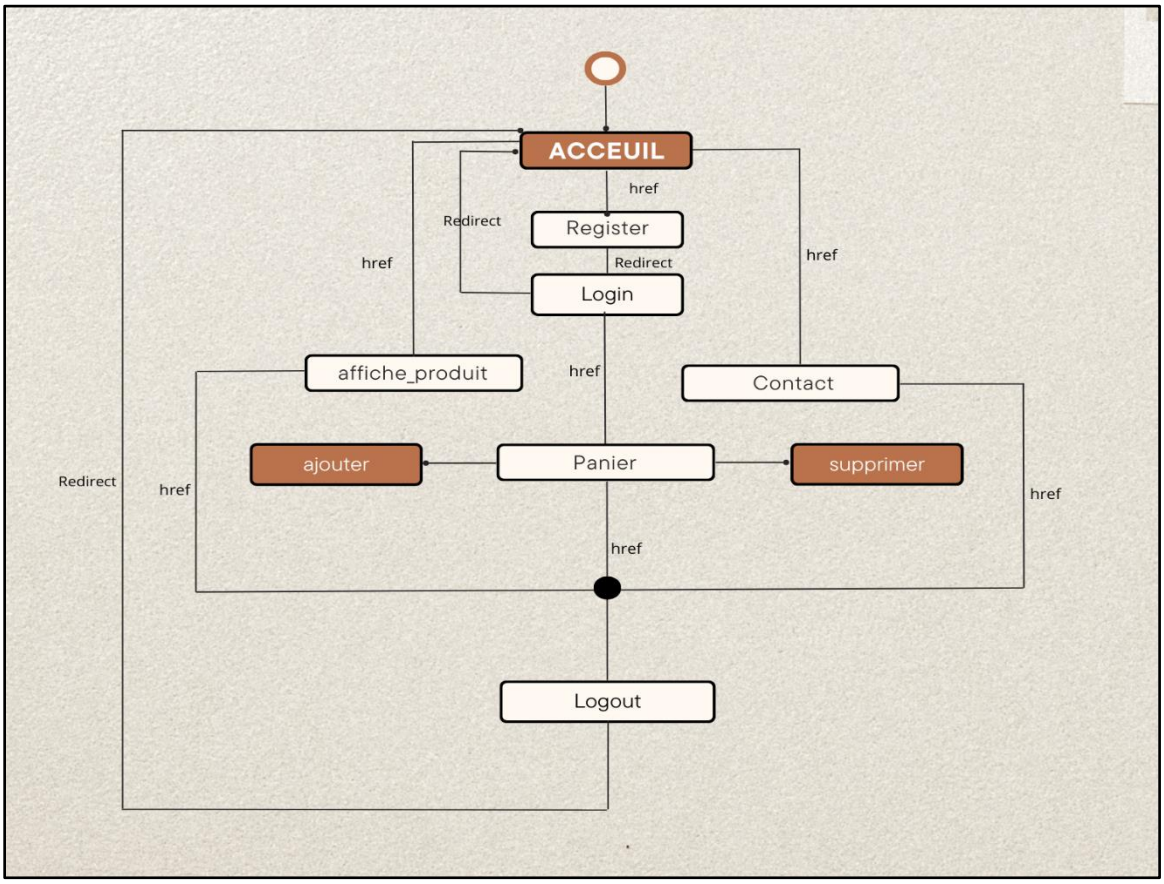
Votre opportunité

Avec EasyCubi, vous avez l'opportunité de créer une entreprise florissante dans un marché en plein essor. Vous pourrez partager votre passion pour le Rubik's Cube avec d'autres personnes et leur offrir une expérience d'achat unique.

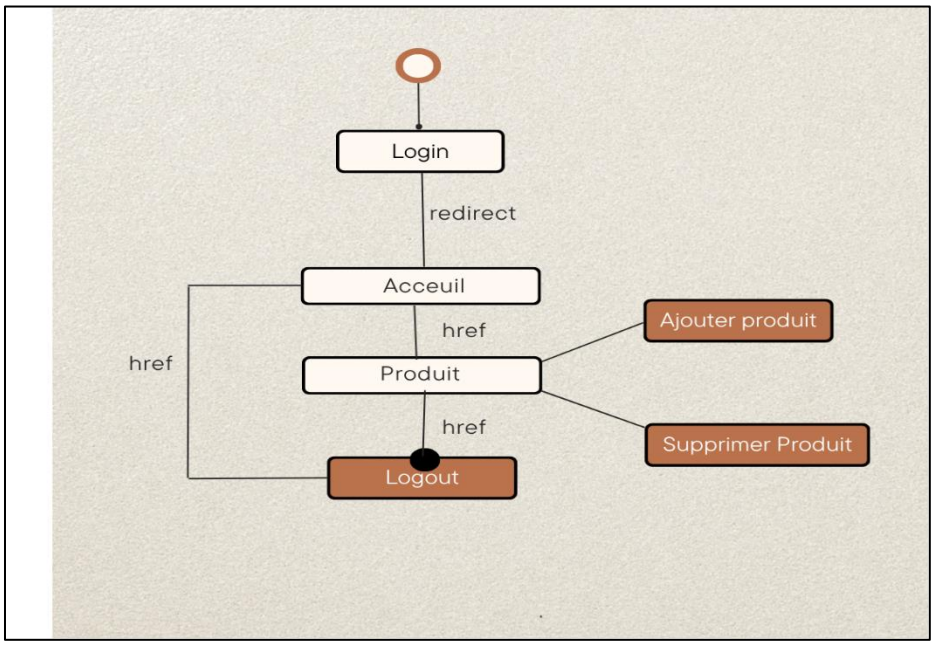
En résumé, EasyCubi est une boutique en ligne qui répond à un besoin réel du marché tunisien. Avec une large sélection de produits, des prix compétitifs et un excellent service client, EasyCubi est bien placée pour devenir la boutique de Rubik's Cube leader en Tunisie.

2. Map du l'application (routes, redirection, liens, ...) :

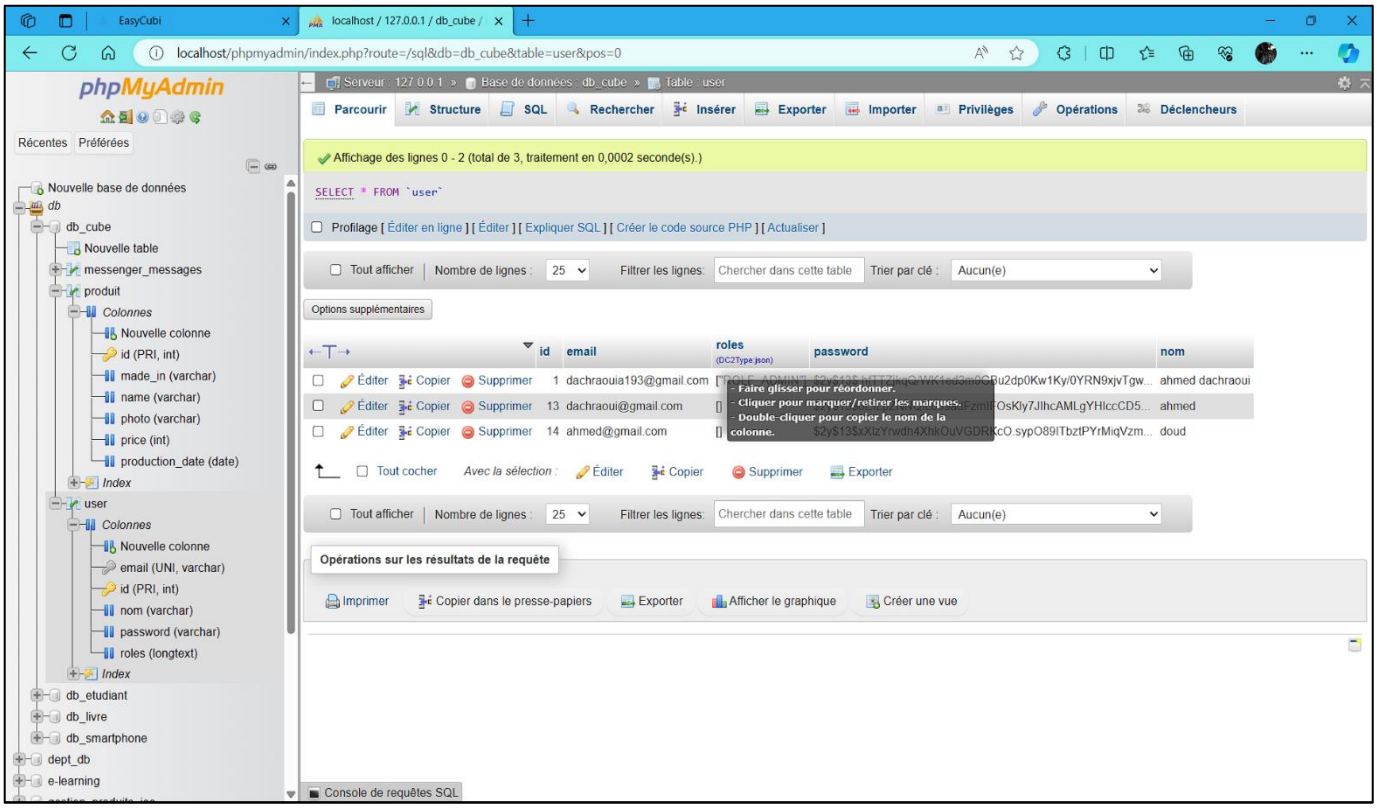
A. Coté Client :



B. Coté Admin :



3. Structure de base de donnés :



4. Description des fichiers de configuration touchés :

A. Page security.yaml :

```
security:
    # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
    # https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider
    providers:
        # used to reload user from session & other features (e.g. switch_user)
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
```

```
            security: false
        main:
            lazy: true
            provider: app_user_provider
            form_login:
                login_path: app_login
                check_path: app_login
                enable_csrf: true
                default_target_path: app_index
            logout:
                path: app_logout
                # where to redirect after logout
                # target: app_any_route

    # Easy way to control access for large sections of your site
    # Note: Only the *first* access control that matches will be used
    access_control:
        - { path: '^/login', roles: PUBLIC_ACCESS }
        - { path: '^/register', roles: PUBLIC_ACCESS }
        - { path: '^/index', roles: ROLE_USER }
        - { path: '^/cart', roles: ROLE_USER }
        - { path: '^/produit', roles: ROLE_ADMIN }
        - { path: '^/produit/add', roles: ROLE_ADMIN }
        - { path: '^/remove', roles: ROLE_ADMIN }
        - { path: '^/contact', roles: PUBLIC_ACCESS}
```

B. routes.yaml :

```
controllers:
    resource:
        path: ../src/Controller/
        namespace: App\Controller
    type: attribute
index:
    path: /index
    controller: App\Controller\IndexController::index
cart:
    path: /cart
    controller: App\Controller\CartController::index
cart_add:
    path: /add/{id}
    controller: App\Controller\CartController::add
cart_remove:
    path: /remove/{id}
    controller: App\Controller\CartController::remove
```

C. services.yaml

```
parameters:
    images_directory: '%kernel.project_dir%/public/images'
services:
    # default configuration for services in *this* file
    _defaults:
        autowire: true      # Automatically injects dependencies in
your services.
        autoconfigure: true # Automatically registers your services as
commands, event subscribers, etc.

    # makes classes in src/ available to be used as services
    # this creates a service per class whose id is the fully-qualified
class name
    App\:
        resource: '../src/'
        exclude:
            - '../src/DependencyInjection/'
            - '../src/Entity/'
            - '../src/Kernel.php'
```

5. Les Codes significatifs :

A.CartController.php :

```
<?php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;
use Symfony\Component\HttpFoundation\Request;
use App\Repository\ProduitRepository;

class CartController extends AbstractController
{
    #[Route('/cart', name: 'app_cart')]
    public function index(Request $request,ProduitRepository
$produitRepository): Response
    {
        $session = $request->getSession();
        $panier = $session->get('panier', []);
        $panierData = [];
        $total = 0;
```

```
        foreach ($panier as $id => $quantity) {
            $product = $produitRepository->find($id);
            $price = $product->getPrice();
            $total += $price * $quantity;
            if ($product) {
                $panierData[] = [
                    'id' => $product->getId(),
                    'name' => $product->getName(),
                    'price' => $price,
                    'quantity' => $quantity,
                ];
            }
        }

        return $this->render('cart/index.html.twig', [
            'items' => $panierData,
            'total' => $total,
        ]);
    }
    public function add($id, Request $request)
    {
        $session = $request->getSession();
        $panier = $session->get('panier', []);
        if (!empty($panier[$id]))
            $panier[$id]++;
        else
            $panier[$id] = 1;

        $session->set('panier', $panier);
        return $this->redirectToRoute('cart');
    }
    public function remove($id, Request $request)
    {
        $session = $request->getSession();
        $panier = $session->get('panier', []);
        if (!empty($panier[$id]))
            $panier[$id]--;
        if ($panier[$id] <= 0)
            unset($panier[$id]);

        $session->set('panier', $panier);
        return $this->redirectToRoute('cart');
    }

}
```

B.ProduitController.php:

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use App\Entity\Produit;
use App\Form\ProduitType;
use Symfony\Component\HttpFoundation\Request;
use Doctrine\Persistence\ManagerRegistry;

class ProduitController extends AbstractController
{
    private $doctrine;

    public function __construct(ManagerRegistry $doctrine)
    {
        $this->doctrine = $doctrine;
    }

    #[Route('/produit', name: 'app_produit')]
    public function index(): Response
    {
        $products = $this->doctrine->getRepository(Produit::class)-
>findAll();

        return $this->render('produit/index.html.twig', [
            'products' => $products,
        ]);
    }

    #[Route('/produit/add', name: 'app_produit_add')]
    public function add(Request $request): Response
    {
        $produit = new Produit();
        $form = $this->createForm(ProduitType::class, $produit);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $photo_prod = $form->get('photo')->getData();
            $originalFilename = pathinfo($photo_prod-
>getClientOriginalName(), PATHINFO_FILENAME);
            $newFilename = $originalFilename . '-' . uniqid() . '.' .
$photo_prod->guessExtension();
```

```
            $photo_prod->move($this->getParameter('images_directory'),
$newFilename);
            $produit->setPhoto($newFilename);

            $entityManager = $this->doctrine->getManager();
            $entityManager->persist($produit);
            $entityManager->flush();

            return $this->redirectToRoute('app_produit');
        }

        return $this->render('produit/form.html.twig', [
            'form' => $form->createView(),
        ]);
    }

    #[Route('/produit/edit/{id}', name: 'app_produit_edit')]
    public function edit(int $id, Request $request): Response
    {
        $produit = $this->doctrine->getRepository(Produit::class)-
>find($id);
        if (!$produit) {
            throw $this->createNotFoundException('No product found for
id '.$id);
        }

        $form = $this->createForm(ProduitType::class, $produit);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $photo_prod = $form->get('photo')->getData();
            if ($photo_prod) {
                $originalFilename = pathinfo($photo_prod-
>getClientOriginalName(), PATHINFO_FILENAME);
                $newFilename = $originalFilename . '-' . uniqid() . '.'
. $photo_prod->guessExtension();
                $photo_prod->move($this-
>getParameter('images_directory'), $newFilename);
                $produit->setPhoto($newFilename);
            }

            $this->doctrine->getManager()->flush();

            return $this->redirectToRoute('app_produit');
        }
```

```
        return $this->render('produit/form.html.twig', [
            'form' => $form->createView(),
        ]);
    }

    #[Route('/produit/delete/{id}', name: 'app_produit_delete')]
    public function delete(int $id): Response
    {
        $produit = $this->doctrine->getRepository(Produit::class)-
>find($id);
        if ($produit) {
            $entityManager = $this->doctrine->getManager();
            $entityManager->remove($produit);
            $entityManager->flush();
        }

        return $this->redirectToRoute('app_produit');
    }
}
```

C.ProduitType.php:

```
<?php
namespace App\Form;

use App\Entity\Produit;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\FileType;
use Symfony\Component\Form\Extension\Core\Type\CountryType;
use Symfony\Component\Form\Extension\Core\Type\MoneyType;
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Validator\Constraints\File;

class ProduitType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array
$options): void
    {
        $builder
        ->add('name')
        ->add('price', MoneyType::class)
        ->add('production_date', DateType::class)
```

```
        ->add('made_in', CountryType::class)
        ->add('photo', FileType::class,

        ['constraints'=> new File(['mimeTypes'=>['image/jpeg',
        'image/jpg',
        'image/png',
        ]
        ])
        ] )
        ->add('OK', SubmitType::class)
        ;
    }
    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Produit::class,
        ]);
    }
}
```

6. Principales commandes:

bin/console make:controller index

Success!

$ bin/console doctrine:database:create

Created database `db_cube` for connection named default

$ php bin/console make:user

The name of the security user class (e.g. User) [User]:

>

Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:

>

Enter a property name that will be the unique "display" name for the user (e.g. email, username, uuid) [email]:

>

Does this app need to hash/check user passwords? (yes/no) [yes]:

>

Success!

$ php bin/console make:entity

Class name of the entity to create or update (e.g. VictoriousChef):

> User

User

Your entity already exists! So let's add some new fields!

New property name (press <return> to stop adding fields):

> nom

Field type (enter ? to see all types) [string]:

> string

string

Field length [255]:

>

Can this field be null in the database (nullable) (yes/no) [no]:

> no

updated: src/Entity/User.php

Add another property? Enter the property name (or press <return> to stop adding fields):

>

Success!

$ php bin/console make:migration

Success!

$ bin/console doctrine:schema:create

Creating database schema...

[OK] Database schema created successfully!

$ php bin/console make:registration-form

Creating a registration form for App\Entity\User

Do you want to add a #[UniqueEntity] validation attribute to your User class to make sure duplicate accounts aren't created? (yes/no) [yes]:

>

Do you want to send an email to verify the user's email address after registration? (yes/no) [yes]:

> no

Do you want to automatically authenticate the user after registration? (yes/no) [yes]:

>

What route should the user be redirected to after registration?:

> 14

14

Do you want to generate PHPUnit tests? [Experimental] (yes/no) [no]:

> no

Success!

$ php bin/console make:auth

What style of authentication do you want? [Empty authenticator]:

[0] Empty authenticator

[1] Login form authenticator

> 1

$ php bin/console make:security:form-login

Choose a name for the controller class (e.g. SecurityController) [SecurityController]:

>

Do you want to generate a '/logout' URL? (yes/no) [yes]:

>

Do you want to generate PHPUnit tests? [Experimental] (yes/no) [no]:

>

created: src/Controller/SecurityController.php

created: templates/security/login.html.twig

updated: config/packages/security.yaml

Success!

$ bin/console make:entity produit

You can always add more fields later manually or by re-running this command.

New property name (press return to stop adding fields):

> name

Field type (enter ? to see all types) [string]:

>

Field length [255]:

>

Can this field be null in the database (nullable) (yes/no) [no]:

>

Add another property? Enter the property name (or press return; to stop adding

fields):

> price

Field type (enter ? to see all types) [string]:

> integer

integer

Can this field be null in the database (nullable) (yes/no) [no]:

> no

updated: src/Entity/Produit.php

Add another property? Enter the property name (or press ‘return’ to stop adding

fields):

> production_date

Field type (enter ? to see all types) [string]:

>date

date

Can this field be null in the database (nullable) (yes/no) [no]:

>

Add another property? Enter the property name (or press <return> to stop adding

fields):

> made_in

Field type (enter ? to see all types) [string]:

>

Field length [255]:

>

Can this field be null in the database (nullable) (yes/no) [no]:

>

Add another property? Enter the property name (or press return to stop adding fields):

>photo

Field type (enter ? to see all types) [string]:

>

Field length [255]:

>

Can this field be null in the database (nullable) (yes/no) [no]:

>

Add another property? Enter the property name (or press return; to stop adding fields):

>

Success!

$ bin/console doctrine:schema:update --force

Updating database schema...

1 query was executed

[OK] Database schema updated successfully!

$ bin/console make:form Produit

The name of Entity or fully qualified model class name that the new form will be bound to (empty for none):

> Produit

Produit

created: src/Form/ProduitType.php

Success!

$ bin/console make:controller produit

Success!

$ bin/console make:controller afficheProduit

Success!

bin/console make:controller Contact

created: src/Controller/ContactController.php

created: templates/contact/index.html.twig

Success!

7. Les captures nécessaires :