Картина, която съдържа текст, графична колекция

Описанието е генерирано автоматичноТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

Факултет по изчислителна техника и автоматизация

Катедра „Софтуерни и интернет технологии“

**КУРСОВ ПРОЕКТ**

по дисциплината „УЕБ програмиране”

на тема: „Портал за недвижими имоти”

Вариант 5

|  |  |
| --- | --- |
| Изготвил: Михаил Бойчев | Проверил: |
| Специалност: СИТ |  |
| Група: 2а |  |
| Факултетен номер: 22621714 |  |

2024

Съдържание

[I. Задание на проекта 3](#_Toc120821980)

[II. Анализ на решението 4](#_Toc120821981)

[1. Структура за данните в програмата 4](#_Toc120821982)

[2. Реализация на условие A 4](#_Toc120821983)

[3. Реализация на условие B 4](#_Toc120821984)

[4. Реализация на условие C 5](#_Toc120821985)

[5. Реализация на условие D 5](#_Toc120821986)

[6. Реализация на условие E 6](#_Toc120821987)

[7. Реализация на условие F 7](#_Toc120821988)

[8. Реализация на условие … - допълнение първо 7](#_Toc120821989)

[9. Реализация на условие … - допълнение второ 8](#_Toc120821990)

[10. Реализация на допълнение трето 8](#_Toc120821991)

[III. Упътване за употреба 9](#_Toc120821992)

[1. Впишете съответната част от проекта 9](#_Toc120821993)

[2. Впишете съответната част от проекта 9](#_Toc120821994)

[IV. Примерно действие на програмата 10](#_Toc120821995)

[1. Условие A 10](#_Toc120821996)

[2. Условие B 10](#_Toc120821997)

[3. Условие C 10](#_Toc120821998)

[4. Условие D 10](#_Toc120821999)

[5. Условие E 10](#_Toc120822000)

[6. Условие F 11](#_Toc120822001)

[7. Допълнение първо 11](#_Toc120822002)

[8. Допълнение второ 11](#_Toc120822003)

[9. Допълнение трето 11](#_Toc120822004)

Задание и изисквания към проекта

Целта на курсовия проект е разработването на функционален уеб портал за продажба или отдаване под наем на недвижими имоти. Основните изисквания включват:

* **Търсене и филтриране:** Възможност за лесно намиране на имоти чрез различни критерии като локация, цена, тип на имота, площ и други.
* **Сортиране:** Опции за подреждане на резултатите по различни параметри, като цена, дата на публикуване.
* **CRUD функционалност:** Потребителите трябва да могат да добавят, редактират и изтриват обяви, включително различни типове имоти.
* **Управление на потребители:** Регистрация и вход за потребителите, с осигуряване на различни нива на достъп и привилегии – стандартни потребители и администратори с разширени права.
* **Детайлна информация за имотите:** Преглед на всяка обява с подробно описание, включващо текст, снимки, характеристики на имота и друга съответна информация.

Системата трябва да позовлява съхраняването на информация в базата данни под формата на таблици:

* Имоти (RealEstate)
* Потребители (User)
* Снимки (Images)
* Типове имоти (RealEstateType)
* Населени места (City)
* Жилищни квартали (Neighborhood)

Системата трябва да използва унифицирана форма за логване от всички потребители (достъпа до системата се осъществява след въведено потребителско име и парола).

За всяка от таблиците в базата данни се използва унифицирана форма за реализиране на стандартните операции: въвеждане, корекция, изтриване и търсене на данни

Реализация на проекта

Проектът е реализиран с помощта на Symfony – PHP framework, Twig за улеснена работа с HTML шаблони, Doctrine – ORM библиотека за работа с базата данни и PostgreSQL

Проектиране

* + Структура на пакетите

Проектът използва стандартната структура на пакетите и директориите, предлагана от Symfony, за да осигури добра организация и лесна поддръжка. Част от основните директории включват:

* + - config/ - съхранява всички конфигурационни файлове на приложението
    - src/ - основната директория, в която се намира логиката на приложението. Тя съдържа всички PHP класове, контролери и услуги.
      * src/Controller/: Тук се намират контролерите на приложението. Всеки контролер обработва HTTP заявки и връща отговори, като използва моделите и представянето на данни.
      * src/Entity/: Тази директория съдържа всички класове, които отговарят на таблиците в базата данни.
      * src/Repository/: В тази директория се намират директориите на класовете, които съдържат логиката за извличане на данни от базата.
      * src/Form – Тук се дефинират всички атрибути и елементи от една форма.
    - templates/ - В нея се намират всички Twig шаблони, които се използват за визуализиране на HTML съдържание. Имаме базов шаблон, от който се разширяват всички останали.
    - public/ - Съдържа всички файлове, които са достъпни за потребителите.
  + Структура на таблиците в БД

ERD структурна схема на таблиците

A diagram of a real estate

Description automatically generated

A diagram of a real estate

Description automatically generated

* + Диаграма на потребителските случаи

A diagram of a person's mind map

Description automatically generated

* + Activity диаграма

A diagram of a flowchart

Description automatically generated

Реализация на условията

Основни дефиниции

* + Модели
    - City – Представлява град, в който могат да бъдат разположени имоти. Включва идентификатор и име на града. Използва се за категоризиране на имотите по местоположение.
    - User – Представлява потребител на системата с атрибути като име, потребителско име, парола, имейл и роля. Възможностите му се определят от ролята, като може да бъде обикновен потребител или администратор.

### **RealEstate** – Представлява недвижим имот с атрибути като име, тип, площ, етаж, цена, адрес, описание, собственик и местоположение (град, квартал). Използва се за съхранение на информация за имоти, които могат да бъдат продавани или давани под наем.

### **Neighborhood** – Представлява квартал, в който се намира даден имот, с атрибути като име на квартала и връзка към съответния град. Използва се за филтриране на имотите по местоположение.

### **RealEstateType** – Представлява тип на имот (напр. апартамент, къща, офис), с атрибут като име на типа. Използва се за филтриране на имоти според тяхната категория.

### **RealEstateImages** – Представлява изображения, свързани с конкретен имот. Съдържа път към изображението и идентификатор на имота. Използва се за визуализиране на имотите в системата.

### **Favourites** – Представлява връзка между потребител и имоти, които е добавил към любимите си, с атрибути като идентификатор на потребителя, имота и връзката между тях. Използва се за съхранение на предпочитанията на потребителите.

* + Repository

Те съхраняват функции свързани със заявките за извличане от базата данни. Всеки модел има собствено repository, където могат да бъдат описани специфични заявки.

* + - RealEstateRepository – съдържа функция за търсене на имоти по град (за търсачката на началната страница) и сортиране на намерените имоти по цена и по площ.

public function findByCityIdSorted(int $cityId, string $sort): array  
{  
 $qb = $this->createQueryBuilder('r')  
 ->where('r.city = :cityId')  
 ->setParameter('cityId', $cityId);  
  
 switch ($sort) {  
 case 'price\_desc':  
 $qb->orderBy('r.estatePrice', 'DESC');  
 break;  
 case 'area\_asc':  
 $qb->orderBy('r.estateArea', 'ASC');  
 break;  
 case 'area\_desc':  
 $qb->orderBy('r.estateArea', 'DESC');  
 break;  
 case 'price\_asc':  
 default:  
 $qb->orderBy('r.estatePrice', 'ASC');  
 break;  
 }  
  
 return $qb->getQuery()->getResult();  
}  
 public function findByCityId(int $cityId): array {  
 return $this->createQueryBuilder('r')  
 ->andWhere('r.city = :cityId')  
 ->setParameter('cityId', $cityId)  
 ->getQuery()  
 ->getResult();  
 }

* + - CityRepository – съдържа заявка за търсене по име на град, а не по подразбиране (id).

public function findByName(string $name): ?City  
{  
 return $this->createQueryBuilder('c')  
 ->andWhere('c.cityName = :name')  
 ->setParameter('name', $name)  
 ->getQuery()  
 ->getOneOrNullResult();  
}

* + - NeighborhoodRepository – Управлява действията, свързани с кварталите. Включва методи за създаване, редактиране, изриване и филтриране на квартали.
    - RealEstateTypeRepository – Управлява действията, свързани с типовете имоти. Включва методи за създаване, редактиране и изтриване на типове имоти.
    - RealEstateImagesRepository – Управлява действията, свързани с изображенията на имотите. Включва методи за качване, редактиране и изтриване на изображения.
    - FavouritesRepository – Управлява действията, свързани с любимите имоти на потребителите. Включва методи за добавяне, премахване и преглед на любими имоти.
  + Контролери

Те представляват PHP функции, които прочитат информация от Request обекта и създават Response обект.

* + - RealEstateController – включва методи за визуализиране на дадена обява на имот, както и за сортиране на страницата с имоти (RealEstateRepository). В Route задаваме пътя на страницата, който в случая съдържа id на града, по който филтрираме имотите. Сортирането се извършва чрез повторна заявка от базата данни, като в нея подреждаме по съответния критерий. Задаваме и името на приложението, което след това може да използваме, за да извикаме определена функция от контролера. Накрая визуализираме страница като задаваме пътя на определен шаблон.

#[Route('/real-estate/{cityId<\d+>}', name: 'app\_estate\_show')]  
public function show( int $cityId, RealEstateRepository $realEstateRepository, Request $request): Response  
{  
  
 $sort = $request->query->get('sort', 'price\_asc');  
 $realEstates = $realEstateRepository->findByCityIdSorted($cityId, $sort);  
 if (!$realEstates) {  
 throw $this->createNotFoundException('Real estate not found');  
 }  
  
  
 return $this->render('estate/show.html.twig', [  
 'realEstates' => $realEstates,  
 'sort'=>$sort  
 ]);  
}

* + - RegistrationController – Контролерите могат да се използват и за създаване на форми – такава е формата за регистрация. При зареждане на страницата се създава нов обект от тип User, както и нова форма от тип RegistrationFormType. Извършва се валидация и от формата могат да бъдат извлечени данни, които са ни нужни за допълнителна обработка, преди да бъдат запазени в базата данни. В случая паролата се криптира за допълнителна сигурност и се задава профилна снимка по подразбиране. След като регистрацията е завършена потребителя бива логнат автоматично.

#[Route('/register', name: 'app\_register')]  
public function register(Request $request, UserPasswordHasherInterface $userPasswordHasher, Security $security, EntityManagerInterface $entityManager): Response  
{  
 $user = new User();  
 $form = $this->createForm(RegistrationFormType::class, $user);  
 $form->handleRequest($request);  
  
 if ($form->isSubmitted() && $form->isValid()) {  
 /\*\* @var string $plainPassword \*/  
 $plainPassword = $form->get('plainPassword')->getData();  
 // encode the plain password  
 $user->setPassword($userPasswordHasher->hashPassword($user, $plainPassword));  
 $user->setProfileImage('assets/images/profile-default.jpg');  
 $user->setCreatedAt(new \DateTimeImmutable());  
 $entityManager->persist($user);  
 $entityManager->flush();  
  
 // do anything else you need here, like send an email  
  
 return $security->login($user, 'form\_login', 'main');  
 }  
  
 return $this->render('registration/register.html.twig', [  
 'registrationForm' => $form,  
 ]);  
}

* + - SecurityController – създава формата за влизане в потребителския акаунт, както и управлява логиката за излизане от акаунт.

1. #[Route(path: '/login', name: 'app\_login')]  
public function login(AuthenticationUtils $authenticationUtils): Response  
{  
 // get the login error if there is one  
 $error = $authenticationUtils->getLastAuthenticationError();  
  
 // last username entered by the user  
 $lastUsername = $authenticationUtils->getLastUsername();  
  
 return $this->render('security/login.html.twig', [  
 'last\_username' => $lastUsername,  
 'error' => $error,  
 ]);  
}  
  
#[Route(path: '/logout', name: 'app\_logout')]  
public function logout(): void  
{  
 throw new \LogicException('This method can be blank - it will be intercepted by the logout key on your firewall.');

}

* + - ListingController – отговаря за създаването на нови обяви за недвижими имоти. Той създава нов обект от тип RealEstate, обработва данните от формата за създаване на обява, качва изображения и ги свързва с имота. Данните се записват в базата данни. След успешното добавяне на обявата, потребителят се пренасочва към основната страница. Ако възникне грешка, се показва съобщение за грешка. Класът осигурява обработка на данни, качване на файлове и взаимодействие с базата данни. Формата позволява и качване на няколко файла наведнъж. Те се записват в масив, а след това се записват в таблицата и във файловата система на програмата, като всеки един от тях получава специален идентификатор. Достъп до тази форма имат само потребители с ролята администратор. #[isGranted('ROLE\_ADMIN')]

#[isGranted('ROLE\_ADMIN')]  
#[Route('/add-listing', name: 'app\_listing')]  
public function listing(Request $request, EntityManagerInterface $entityManager, SluggerInterface $slugger,  
 #[Autowire('public/assets/images/estates/')] string $imageDirectory): Response  
 {  
 $estate = new RealEstate();  
 $user = $this->getUser();  
 $form = $this->createForm(AddListingType::class, $estate);  
 $form->handleRequest($request);  
  
 if($form->isSubmitted() && $form->isValid()) {  
  
 $images = $form->get('imageUpload')->getData();  
 if ($images) {  
 foreach ($images as $image) {  
 $originalFilename = pathinfo($image->getClientOriginalName(), PATHINFO\_FILENAME);  
 $safeFilename = $slugger->slug($originalFilename);  
 $newFilename = $safeFilename . '-' . uniqid() . '.' . $image->guessExtension();  
  
 try {  
 $image->move($imageDirectory, $newFilename);  
 } catch (FileException $e) {  
 $this->addFlash('error', 'An error occurred while uploading the image. Please try again.');  
 $this->get('logger')->error('File upload error: ' . $e->getMessage());  
 }  
  
 $realEstateImage = new RealEstateImages();  
 $realEstateImage->setImagePath($imageDirectory. $newFilename);  
 $realEstateImage->setRealEstate($estate);  
  
 $entityManager->persist($realEstateImage);  
  
 }  
 }  
  
// $estate->addRealEstateImage($realEstateImage);  
 $estate = $form->getData();  
 $estate->setUserId($user);  
 $estate->setDateAddedAt(new \DateTimeImmutable());  
 $entityManager->persist($estate);  
 $entityManager->flush();  
 return $this->redirectToRoute('app\_listing');  
 }  
  
 return $this->render('listing/add-listing.html.twig', [  
 'realEstateForm' => $form->createView(),  
 ]);  
 }

* + - * ProfileController – работи по идентичен начин на ListingController, но вместо масив от файлове, тук той е само един и бива обработен и записан по същия начин.

#[IsGranted('ROLE\_USER')]  
#[Route('/profile', name: 'app\_profile')]  
public function profile(Request $request, EntityManagerInterface $entityManager, SluggerInterface $slugger,  
 #[Autowire('public/assets/images/avatars/')] string $profilePictureDirectory ): Response  
{  
 $user = $this->getUser();  
 $form = $this->createForm(ProfileType::class, $user);  
 $form->handleRequest($request);  
  
 if ($form->isSubmitted() && $form->isValid()) {  
 $profilePicture = $form->get('profilePicture')->getData();  
 if ($profilePicture) {  
 $originalFilename = pathinfo($profilePicture->getClientOriginalName(), PATHINFO\_FILENAME);  
 $safeFilename = $slugger->slug($originalFilename);  
 $newFilename = $safeFilename.'-'.uniqid().'.'.$profilePicture->guessExtension();  
 try {  
 $profilePicture->move($profilePictureDirectory, $newFilename);  
 } catch (FileException $e) {  
 // Handle exception if something happens during file upload  
 $this->addFlash('error', 'An error occurred while uploading the profile picture. Please try again.');  
 $this->get('logger')->error('File upload error: ' . $e->getMessage());  
 }  
 $user->setProfileImage($profilePictureDirectory . $newFilename);  
 }  
 $entityManager->persist($user);  
 $entityManager->flush();  
 return $this->redirectToRoute('app\_profile');  
 }  
  
 return $this->render('user/profile.html.twig', [  
 'profileForm' => $form,  
 ]);  
}

* + Forms

Формите се построяват използвайки Builder pattern. Към него се добавят отделни атрибути от съответния тип желан да се визуализира във формата.

* + - RegistrationFormType - използван за създаване на регистрационния формуляр за нови потребители. Той включва полета за потребителско име, съгласие с условията и парола. Полето за парола се повтаря, за да се гарантира, че потребителят я въвежда правилно. Прилагат се ограничения за валидиране на входните данни, като например гарантиране, че условията са приети и паролата отговаря на изискванията за дължина. Данните от формуляра се свързват с класа User.

1. public function buildForm(FormBuilderInterface $builder, array $options): void  
{  
 $builder ->add('username')  
  
 ->add('agreeTerms', CheckboxType::class, [  
 'mapped' => false,  
 'constraints' => [  
 new IsTrue([  
 'message' => 'You should agree to our terms.',  
 ]),  
 ],  
 ])  
 ->add('plainPassword', RepeatedType::class, [  
 'type' => PasswordType::class,  
 // instead of being set onto the object directly,  
 // this is read and encoded in the controller  
 'mapped' => false,  
 'first\_options' => ['label' => 'Password'],  
 'second\_options' => ['label' => 'Repeat Password'],  
 'invalid\_message' => 'The password fields must match.',  
 'attr' => ['autocomplete' => 'new-password'],  
 'constraints' => [  
 new NotBlank([  
 'message' => 'Please enter a password',  
 ]),  
 new Length([  
 'min' => 6,  
 'minMessage' => 'Your password should be at least {{ limit }} characters',  
 // max length allowed by Symfony for security reasons  
 'max' => 4096,  
 ]),  
 ],  
 ]);

* + - AddListingType - Използван за създаване на формуляр за добавяне на нова обява за недвижим имот. Включва полета за име на имота, площ, етаж, дали е обзаведен, дали е за отдаване под наем, цена, адрес, година на построяване, тип на имота, град, квартал и описание. Добавено е поле за качване на изображения с ограничения за размер и тип на файла. Извършва се трансформиране на стойностите за град и квартал чрез специални трансформатори. Данните от формуляра се свързват с класа RealEstate.

public function buildForm(FormBuilderInterface $builder, array $options): void {  
 $years = range(2025, 1900);  
 $choices = array\_combine($years, $years);  
  
 $builder ->add('estateName', TextType::class, [  
 ])  
 ->add('estateArea', TextType::class, [  
 'label' => 'Estate Area',  
 'attr' => [  
 'placeholder' => 'sqm.',  
 ],  
 ])  
 ->add('estateFloor')  
 ->add('isFurnished', CheckboxType::class,[  
 'required' => false,  
 ])  
 ->add('isForRent', CheckboxType::class,[  
 'required' => false,  
 ])  
 ->add('estatePrice')  
 ->add('estateAddress')  
 ->add('yearBuilt', ChoiceType::class, [  
 'label' => 'Year Built',  
 'choices' => $choices,  
 'placeholder' => 'Select Year',  
 ])  
 ->add('type', EntityType::class, [  
 'class' => RealEstateType::class,  
 'choice\_label' => fn(RealEstateType $type) => sprintf('%s', $type->getTypeName()),  
 'placeholder' => 'Select Type' ])  
 ->add('city', TextType::class, [  
 'invalid\_message' => 'That is not a valid city name',  
 ])  
 ->add('neighborhood', TextType::class, [  
 'invalid\_message' => 'That is not a valid neighborhood name',  
 ])  
 ->add('estateDescription', TextareaType::class, [  
 'required' => false,  
 ])  
 ->add('imageUpload', DropzoneType::class, [  
 'attr' => [  
 'placeholder' => '',  
 ],  
// 'data\_class' => RealEstateImages::class,  
 'mapped' => false,  
 'required' => true,  
 'multiple' => true,  
 'constraints' => [  
 new All([  
 'constraints' => [  
 new File([  
 'maxSize' => '10240k',  
 'mimeTypes' => [  
 'image/jpeg',  
 'image/png',  
 ],  
 'mimeTypesMessage' => 'Please upload a valid image format (jpg, png)',  
 ])  
 ],  
 ])  
 ],  
 ]);  
  
  
 $builder  
 ->get('city')  
 ->addModelTransformer($this->cityNameTransformer);  
 $builder  
 ->get('neighborhood')  
 ->addModelTransformer($this->neighborhoodNameTransformer);  
 }

* + - CityNameTransformer - Използва се за трансформиране на данни между формуляра и базата данни при обработка на името на град. Трансформаторът има две основни функции:
      * transform(): Преобразува обекта City в неговия идентификатор (ID), за да бъде използван в полето на формуляра. Ако стойността е null, се връща празен низ.
      * reverseTransform(): Преобразува входната стойност от полето на формуляра (име на град) в обект City. Ако такъв град не съществува, се създава нов обект City, който се записва в базата данни. Този клас ни позволява да работим директно с имената на градовете, вместо техните id-та.

public function transform($value): string {  
 if (null === $value) {  
 return ' ';  
 }  
  
 return $value->getId();  
 }  
  
 public function reverseTransform($value): ?City {  
 if (!$value) {  
 throw new TransformationFailedException('City name cannot be empty.');  
 }  
 $city = $this->entityManager ->getRepository(City::class)  
 ->findOneBy(['cityName' => $value])  
 ;  
  
 if (null === $city) {  
 $city = new City();  
 $city->setCityName($value);  
 $this->entityManager->persist($city);  
 $this->entityManager->flush();  
// throw new TransformationFailedException(sprintf(  
// 'City with name "%s" does not exist!',  
// $value  
// ));  
 }