**Django WEB приложение ConstructionProjectManagement**

**Въведение**

Настоящото Django приложение e на основа на идея възникнала на базата на моят дългогодишен опит в инвестиционните процеси, които се извършват при реализиране на строителни обекти. Това е сложен процес и в него има много участници в зависимост от категорията и сложноста на строежа генерират се много документи в целия процес от проектирането, одобрението и до неговото изпълнени, което е съпътствано от много одобрения и разрешителни от държавната администрация. Всичко това изисква да се изгради добре структурирана база от данни, която да е достъпна през web пространството, добре подредена с достъпен интерфейс. В момента организацията на документацията се случва на сървър, ако може да се нарече такъв, в който едни служители създават папки, а други качват документи без никакъв ред и еталон за наименуване на папки и файлове, което много затруднява работния процес.

Основната структура на Django проекта ConstructionProjectManagement е на основата на участници в реализиране на един инвестиционен проект и те са възложител/клиент, проектанти, строител и строителен надзор/консултант съгласно основния закон за реализиране на строителната дейност ЗУТ. Настоящия Django проект е съвсем базов проект, който може да послужи за надграждане до нещо много голямо и сложно, като софтуер и да има приложение в строителния бизнес.

**Структура на Django проекта**

Основните приложения в Django проекта са свързани с описаните по горе участници в строителния процес: възложител/клиент, проектанти, строител и строителен надзор/ консултант наименовани съответно така:  **designers, builders и supervisions** и свързващото ги приложение **projects,** отнасящо се за инвестиционния проект. Има и две приложения с общо предназначение: **accounts** – отговарящо за регистриране и логване на потребителите, както и **common** отговарящо за показване на общо достъпни страници за всякакви потребители от цялото интернет пространство, в които е описана обща информация за дейноста на фирмата осигуряваща финансиране и в последсвие ползвател на изградения строителен обект. В случая съм направил две странички с измислено портфолио на фирмата на възложителя клиент, като логото е мое авторско изпълнение включващо инициалите на моето име и фамилия. Снимката на бекграунда в темплейта е също мое авторско изпълнение при участието ми в един проект, като инвеститорски контрол. И на края има едно последно приложение **tasks** отговарящо за поставените задачи от страна на възложителя/клиента към останалите участници в строителния процес, както и за техните доклади **reports** към дадената задача.

* Приложение **accounts:**

В models.py съм създал два модела **AppUser** и **Profile** свързани с OneToOne връзка в полето user. В AppUser съм предвидил **AppUserManager** написан в managers.py наследяващ BaseUserManager на Django контролиращ входа и изхода на базата данни за правата на потребителите, като и определя и създаването на user и superuser. Предвидил съм и **CustomAuthBackend** в authentication.py проверява дали потребителското име и паролата съществуват (удостоверяване на потребителя) и осигурява възможност потребителя да се логва, като с email така и с username. За съдаване автоматично на профил към логнатия потребител без да има данни съм задал signals.py с метод create\_profile. В admi.py съм създал два класа AppUserAdmin и ProfileAdmin за user и неговия profile

Формата при създаване на акаунт е **AppUserCreationForm** наследява **UserCreationForm** от Django. За профила създавам основна **ProfileBaseForm** наследена от forms.ModelForm, която се наследява от **ProfileEditForm** и **ProfileDeleteForm.** В views.py създадох основно клас основни изгледи (CBVs) с основната логика при обработка и рендериране на заявките в различните темплейтите и свързването им към съответния URL. Първият CBV е NewAccountView наследен от CreateView на Django за създаване на темплейта за регистрация "register/". Другите CBVs са **DetailProfileView** наследява LoginRequiredMixin, DetailView, **EditProfileView** наследява UserPassesTestMixin, PermissionRequiredMixin, UpdateView, **DeleteProfileView** наследява PermissionRequiredMixin, UserPassesTestMixin, DeleteView свързани с обработка на профила на акаунта. Както споменах по-горе създаването на празна форма за профила се извършва с signals. PermissionRequiredMixin има за задача да проверява правата на потребителя и дали той е логнат, съответно LoginRequiredMixin проверява дали потребителя е логнат. Основно при всички приложения използвам тези два миксина за да разделя кой CBV може да „работи“ за съответния user, който има съответните права делегирани в групата към която принадлежи. Така където имам DetailView или ListView използвам LoginRequiredMixin така от групата на Other, където са логнати потребители не участващи в никакви проекти, но по закон имат достъп до тази информация *(Възложители, които работят със закона за обществени поръчки са длъжни на преоставят информация по основни параметри на проекта)* в другите UpdateView, CreateView и DeleteView, където нямат прав се филтрират с PermissionRequiredMixin.

UserPassesTestMixin проверявате достъп до профила на потребителя с test\_func(self) дали логнатия потребител достъпва неговият профил.

* Приложение **common**:

Както споменах по горе отговаря за показване на общо достъпни страници за всякакви потребители има два CBVs - HomeView и HomeExView наследени от TemplateView на Django и съответно template\_name за темплейтите.

* Приложенията **designers, builders и supervisions**:

Те са почти индентични като модели затова създадох в mixins.py в папка core AbstractBaseMixin наследен от models.Model, където съм описал еднаквите полетата на моделите. Отделните модели на приложенията наследяват AbstractBaseMixin и към всеки модел създавам *document = models.FileField* и *project = models.ManyToManyField* също еднакви, но раличаващи се по *upload\_to= и related\_name=.* Създал съм валидатор SizeValidator в validators.py в папка core за ограничаване на големината на качвания файл към съответната директория създадени в media\_files. В mixins.py в папка core създадох ContractBaseForm наследява forms.ModelForm, която се наследява от основна форма (BuilderBaseForm, DesignerBaseForm и SupervisionBaseForm) в forms.py към съответното приложение. Основната форма има поле *project = forms.ModelMultipleChoiceField* с параметри *queryset=Project.objects.all()* *widget=forms.CheckboxSelectMultiple* и require, label, както и *class Meta с model = ... и fields = '\_\_all\_\_'.* Използвам forms.ModelMultipleChoiceField, защото project е създаден като ManyToManyField. По този начин като създавам нов designer, builder или supervision определям към кой проект/и ще участва. После тази основна форма се наследява от другите форми за ...CreateForm, ...EditForm, ...DetailsForm, ...DeleteForm като Delete формата наследява и ReadOnlyFieldsMixin, за да може преди изтриване формата да се покаже със съответните данни.

* Приложение **projects**:

Към модела на това приложение са вързани с ManyToMany моделите на **designers, builders и supervisions.** В неговия модел има стандартни полета и едно по различно *development=models.CharField(...choices=DevelopmentChoices.choices)*. Където Development-Choices се намира в choices.py в този клас са описани елементи представляващи по-важни документи съставящи се по време на строителство даващи представа на какъв етап от строителството е строежа:

- 'Building permit' – Разрешение за строеж

- 'Protocol 2' – Протокол за откриване на строителството

- 'Order Book' – Заповедна книга

- 'ACT 10' – Акт за временно спиране на строежа

- 'ACT 11' - Акт за пускане на строежа

- 'Statutory Act 15' – Акт за предаване на готовия строеж от строителя към възложителя/клиента

- 'Protocol 17' – Протокол за 72 часово тестване (проби) на строежа

- 'Protocol 16' – Протокол от съставен от Държавна приемателна комисия

- 'Permission Usage' - Разрешение за ползване.

За формите създадох ProjectBaseForm наследен от forms.ModelForm, която се наследява от ProjectCreateForm, ProjectEditForm, ProjectDetailsForm и ProjectDeleteForm. В файла views.py използвам CBVs като по долу съм описал какви проперти и методи съдържат стандартно както и другите приложения.

* Приложениe **tasks,**

Това приложение е да се организират задчите и докладите към тях за съответния проект. В създал съм два модела *Task и Report*. В Task съм създал поле *project = models.ForeignKey* към съответния *Project* и *author = models.ForeignKey* към *UserModel* съответстващ на текущия потребител, с цел при генериране на задачата да се вземат данните от профила за попълване в нея. По същия начин в Report съм създал поле *task = models.ForeignKey* към съответния Task и поле *reporter = models.ForeignKey към UserModel* съответстващ на текущия потребител, с цел при генериране на задачата да се вземат данните от профила за попълване в нея. Разбира се има други полета са описване на задаите и докладите и полета за генериране на дата и час, като за задачите авоматично се задава дата и час при създаване, а за докладите дата и часа се променят при всяко редактиране или създаване. В файла views.py използвам CBVs като по долу съм описал какви проперти и методи съдържат стандартно както и другите приложения.

В mixins.py в папка core създадох TaskBaseForm и ReportBaseForm наследяват forms.ModelForm. TaskBaseForm се наследява от TaskMainForm, която от своя страна се наследява от TaskCreateForm, TaskEditForm, TaskDetailsForm, TaskDeleteForm. TaskMainForm има поле project = forms.ModelChoiceField с параметри queryset=Project.objects.all() widget=forms.Select и require, label, както и class Meta с model = ... и fields = '\_\_all\_\_'. Използвам forms.ModelChoiceField, защото project е създаден като ForeignKeyField. По този начин като създавам нова задача определям към кой проект ще се отнася. Delete формата наследява и ReadOnlyFieldsMixin, за да може преди изтриване формата да се покаже със съответните данни. Посъщия начин организирам и наследяването ReportBaseForm и отново създавам само, че task = forms.ModelChoiceField (защото task е създаден като ForeignKeyField) с параметри queryset=Task.objects.all() widget=forms.Select и require, label, както и class Meta с model = ... и fields = '\_\_all\_\_'. По този начин като създавам нов doklad определям към коя задача ще се отнася.

При Delete формите наследявам и ReadOnlyFieldsMixin, за да може преди изтриване формата да се покаже със съответните данни.

За създаване на доклад към съответната задачата в „task-detajl.html“ създавам бутон *<button><a href="{% url 'report-add' %}?task\_id={{ task.pk }}">Add for this Task</a></button>*, където прикачвам параметър към заявката *?task\_id={{ task.pk }}* с текъщия pk на задачата, който използвам в ReportCreateView, при валидиране на формата. Създавам *task\_id = self.request.GET.get('task\_id')* от заявката GET взимам прикачения параметър и с него достъпвам task = Task.objects.get(pk=task\_id) с пренесения *task\_id.*

В CBVs на приложенията използвам следните проперти и методи:

- model = задаване на модела,

- form\_class = задаване на формата,

- template\_name = задаване на темплейта за рендериране,

- pk\_url\_kwarg = задаване на pk различен от този по default при динамичните URL,

- permission\_required = 'група.права\_модел' зададени права в групата описани в admin/,

- get\_context\_data(self) динамично представяне на context, в него може да се вмъкнат и данни от други модели,

- def get\_success\_url(self) пренасоване при динамичен URL,

- success\_url = статично пренасоване към URL,

- get\_initial(self) – взимане на данните от текущия обект във формата на dict,

- get\_form\_kwargs(self) – промяна на съдаржанито на формата,

- def form\_valid(self, form) – проверяваме формата преди да я запишем в дадения модел,

* **URL**

Имат следната огранизация:

**основен проект urls.py**

'admin/'

'accounts/', include('accounts.urls')

'common/', include('common.urls')

'projects/', include('projects.urls')

'builders/', include('builders.urls')

'designers/', include('designers.urls')

'supervisions/', include('supervisions.urls')

'tasks/', include('tasks.urls')

**accounts.urls.py**

"register/"

"login/"

"logout/"

"profile/<int:pk>/details/'

'profile/<int:pk>/edit/'

'profile/<int:pk>/delete/'

**common.urls.py**

'home/'

'home-ex/'

**projects.urls.py**

''

'create/'

'<int:pro\_pk>/details/'

'<int:pro\_pk>/edit/'

'<int:pro\_pk>/delete/'

**builders.urls.py**

''

'create/'

'<int:pk>/details'

'<int:pk>/edit/'

**designers.urls.py**

''

'create/'

'<int:pk>/details'

'<int:pk>/edit/'

**supervisions.urls.py**

''

'create/'

'<int:pk>/details'

'<int:pk>/edit/'

**tasks.urls.py**

''

'create/'

'<int:pk>/detail/'

'<int:pk>/edit/'

'<int:pk>/delete/'

'report-add/'

'report/<int:rep\_pk>/detail/'

'report/<int:rep\_pk>/edit/'

'report/<int:rep\_pk>/delete/'

* **Инструкция за стартиране на web приложението**

Приложението е настроено да стартира със следните свойства

One instance of the project, Hot reloading , Servers static files

1. Отваряне на приложението в PyCharm

2. Създаване на виртуална среда и активиране с команди

python3 -m venv .venv

.venv\Scripts\activate # Windows

3. Написване на .env файла със следните настройки:

* + SECRET\_KEY = ' ......... '
  + DEBUG = ' ......... '
  + ALLOWED\_HOSTS = ' ......... '
  + NAME = ' .................. '
  + USER = ' ................ '
  + PASSWORD = ' ............ '
  + HOST = ' .............. '
  + PORТ = ' ............ '

4. Инсталиране на програмите в requirements.txt с команда

pip install -r requirements.txt

5. Направа на миграции

python manage.py migrate

6. Зареждане на видовете групи за правата на потребителите

python manage.py loaddata initial\_groups.json

python manage.py migrate

7. Създаване на суперпотребител

python manage.py createsuperuser

8. Стартиране на приложението локално с команда:

python manage.py runserver

Изготвил: Андриан Стоянов