**Руководство программиста.**

**О программном стеке разработки.**

В качестве рабочего стека был выбран фреймворк Django.

Django – это бесплатный фреймворк для создания веб-приложений написанный на языке Python.

Как и любой другой фреймворк, «Джанго» — это программная платформа, на который разработчик добавляет дополнительные компоненты приложения, например:

* 1. Интерфейс пользователя.
  2. Скрипты (сценарии) работы элементов приложения/сайта.
  3. Программные библиотеки.
  4. Мультимедийный контент (фото, видео, аудио).
  5. Модули безопасности.
  6. Базы данных.

Фреймворк отвечает за базовые аспекты работы продукта. А подключаемые модули — за специфичные функции конкретного приложения.

Использование фреймворков позволяет:

* 1. Ускорить разработку.
  2. Упростить поддержку.
  3. Избежать ошибок в работе приложения.

**О программе.**

Весь Web проект состоит из 2 приложений – Kraus, Main.

В папке Kraus лежат все файла для настройки всего проекта. Там прописаны пути для статики, настройки debug, подключенные внутренние и сторонние библиотеки, домен и тд.

В папке Main развернут сам сайт.

В него входят следующие файлы:

1. Views – определяет функции, которые получают запросы пользователей, обрабатывают их и возвращают ответ (обработчик веб-страниц).

2. Models – хранит определение моделей, которые описывают используемые в приложении данные.

3. Apps – определяет конфигурацию приложения.

4. Admin – предназначен для административных функций, в частности, здесь производится регистрация моделей, которые используются в интерфейсе администратора.

5. Init – указывает интерпретатору python, что текущий каталог будет рассматриваться в качестве пакета.

6. папка migrations – предназначена для хранения миграций - скриптов, которые позволяют синхронизировать структуру базы данных с определением моделей.

7.urls – определяет url адреса для будущих страниц.

В качестве дополнительных библиотек были использованы – telepot, django-phonenumbers.

Для того чтобы развернуть проект на локальном сервере, необходимо иметь установленный фреймворк Django. Далее дойдя по каталогу до корневой папки с проектом в командной строке, нужно прописать python manage.py runserver. После чего сайт будет доступен по ссылке в консоли. Убедитесь, что у вас установлено виртуальное окружение venv.

На рисунке 1 изображен весь проект в каталоге.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – файловая архитектура сайта.

На сайте предусмотрена регистрация пользователя. Без регистрации пользователь не сможет связаться с менеджером для дальнейшей беседы о покупке товаров.

Сначала для регистрации было сделана форма, в которую пользователь будет заносить свои данные. После заполнения этих полей данные передаются в базу данных, где и совершается запись пользователя.

Ниже приведен код формы для регистрации.

# Создаём класс формы (регистрация)  
class RegisterUser(UserCreationForm):  
  
 # Здесь задаются начальные настройки для поля логин.  
 username = forms.CharField(label='Логин',  
 widget=forms.TextInput(attrs={'class': 'form-input', 'placeholder': 'Логин'}))  
  
 # Здесь задаются начальные настройки для поля email.  
 email = forms.EmailField(required=True, label='Email',  
 widget=forms.EmailInput(attrs={'class': 'form-input', 'placeholder': 'Email'}))  
  
 # Здесь задаются начальные настройки для полей с паролем.  
 password1 = forms.CharField(label='Пароль',  
 widget=forms.PasswordInput(attrs={'class': 'form-input', 'placeholder': 'Пароль'}))  
 password2 = forms.CharField(label='Повтор пароля',  
 widget=forms.PasswordInput(attrs={'class': 'form-input', 'placeholder': 'Повтор пароля'}))  
  
 # Здесь задаются начальные настройки для поля имя.  
 first\_name = forms.CharField(label='Имя',  
 widget=forms.TextInput(attrs={'class': 'form-input', 'placeholder': 'Имя'}))  
  
 # Здесь задаются начальные настройки для поля фамилия.  
 last\_name = forms.CharField(label='Фамилия',  
 widget=forms.TextInput(attrs={'class': 'form-input', 'placeholder': 'Фамилия'}))  
   
 class Meta:  
  
 # Свойство модели User  
 model = User  
  
 # Свойство назначения полей  
 fields = ('username', 'email', 'password1', 'password2', 'first\_name', 'last\_name')  
  
 # Сохранение в бахзу данных  
 def save(self, commit=True):  
 user = super().save(commit=False)  
   
 if commit:  
 user.save()  
 return user

После создания формы реализуется рендер страницы, к которой подвязывается форма для регистрации и формируется логика для отправки запроса в базу данных.

Ниже приведен код с рендером страницы регистрации.

# Страница с регистрацией и автоматическим логином  
def registration(request):  
  
 # Проверка на запрос.  
 if request.method == 'POST':  
 form = RegisterUser(request.POST)  
 profile\_form = UserProfileForm(request.POST)  
  
 # Проверка форм на валидность.  
 if form.is\_valid() and profile\_form.is\_valid():  
 user = form.save()  
  
 profile\_user = profile\_form.save(commit=False)  
 profile\_user.user = user  
 profile\_user.save()  
   
 username = form.cleaned\_data.get('username')  
 password = form.cleaned\_data.get('password1')  
   
 user = authenticate(username=username, password=password)  
 login(request, user)  
   
 return redirect('profile')  
 else:  
 form = RegisterUser()  
 profile\_form = UserProfileForm()  
   
 context = {'form': form, 'profile\_form': profile\_form}  
 return render(request, 'main/registration.html', context)

Для авторизация был подключен стандартный модуль Django для авторизации. Называется это модуль LoginView. Единственно, что изменено для авторизации – это форма для ввода данных.

Ниже приведен код для авторизации и ее форма.

# Форма для авторизации.  
class LoginUserFrom(AuthenticationForm):  
  
 # Здесь задаются начальные настройки для поля логин.  
 username = forms.CharField(label='Логин',  
 widget=forms.TextInput(attrs={'class': 'form-input'}))  
  
 # Здесь задаются начальные настройки для поля пароль.  
 password = forms.CharField(label='Пароль',  
 widget=forms.PasswordInput(attrs={'class': 'form-input'}))

# Страница с авторизацией пользователя  
class LoginUser(LoginView):  
 form\_class = LoginUserFrom  
 template\_name = 'main/Login.html'

Товары же выводятся посредствам базы данных. Все названия, описания и меда файлы товаров хранятся в базе данных. Все записи распределены по категориям товаров.

Пример модели консолей.

# Модель для консолей  
class Consoles(models.Model):  
  
 # Название полей в таблице консоли.  
 name = models.CharField('Название товара', max\_length=100)  
 size = models.CharField('Размеры', max\_length=100)  
 color = models.CharField('Цвет каркаса', max\_length=250)  
 price = models.CharField('Цена товара 1', max\_length=100)  
 price2 = models.CharField('Цена товара 2', max\_length=100)  
 tabletop = models.CharField('Столешница', max\_length=250)  
 description = models.CharField('Описание товара', max\_length=250)  
 img = models.ImageField('Изображение товара')  
  
 # Видимость записей в админсокй панели по их именам.  
 def \_\_str\_\_(self):  
 return f'{self.name}'  
  
 # Название категории в админской панели.  
 class Meta:  
 verbose\_name = 'Консоль'  
 verbose\_name\_plural = 'Консоли'

Здесь описаны поля, которые используются для записи товаров в данную категорию. Обязательно после описания моделей в файле models, нужно сделать миграции в базу данных. Делается это двумя консольными командами:

1. Python manage.py makemigrations
2. Python manage.py migrate

Первая консольная программа создает миграции и создает файлик с описанием всех моделей, которые вы сделали.

Пример представлен на рисунке 2.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – папка migrations.

Так же следует применять эту команду для обновления моделей, если вы ее изменили.

Вторая же консольная команда уже создает таблицу и раздел из описанных вами моделей.

Для выведения записей о товаре на саму товарную страницу используется ключ передачи данных. Благодаря ему можно, можно разбить все имеющиеся данные на блоки. Вывод всех этих блоках происходит в карточке товара.

Ниже приведен код рендера товара и вывод информации из базы данных на html странице.

{% if con.count > 0 %} {% for con in con %} {% if con.img %}  
<div class="t754 wrapper" style="margin-top: 50px;">  
 <div class="t754\_\_parent t754\_\_container\_mobile-grid">  
 <div class="t754\_\_col t-col t-col\_4 t-align\_center t-item t754\_\_col\_mobile-grid js-product" data-product-lid="{{con.id}}" style="min-width: 400px; min-height: 450px;">  
 <a class="js-product-link" href="#prodpopup">  
 <div class="t754\_\_content">  
 <div class="t754\_\_imgwrapper t1002\_\_picture-wrapper " style="padding-bottom:111.11111111111%;">  
 <div class="t754\_\_bgimg t-bgimg js-product-img" data-original="{{con.img.url}}" style="background-image:url('{{con.img.url}}')" data-lazy-rule="comm:resize"></div>  
 </div>  
 <div class="t754\_\_textwrapper">  
 <div class="t754\_\_title t-name t-name\_md js-product-name" data-auto-correct-font-size="rem" field="li\_title\_\_1497456130776" style="font-weight:700;"> <strong>{{con.name}}</strong> </div>  
 <span class="t-name\_\_price"><span class=""><bdi><span class="">{{con.price}}</span></bdi>  
 </span> – <span class=""><bdi><span class="">{{con.price2}}</span></bdi>  
 </span>  
 </span>  
 </div>  
 </div>  
 </a>  
 </div>  
 </div>

# Страница категории консоли.  
def consoles(request):  
 con = Consoles.objects.all()  
 if request.method == 'POST':  
 form = CommunicationForm(request.POST)  
 if form.is\_valid():  
 mail = form.cleaned\_data['email']  
 phone = form.cleaned\_data['phoneNumber']  
 message = "\*ЗАЯВКА С САЙТА\*:" + "\n" + "\*ПОЧТА\*: " + str(mail) + "\n" + "\*ТЕЛЕФОН\*: " + str(phone)  
 send\_message(message)  
 return redirect('main')  
 else:  
 form = CommunicationForm()  
 return render(request, 'main/Консоли.html', {"con": con, "form": form})

Для отправления заявок с сайта используется телеграмм бот. Настройка данного бота лежит в отельном файле, где указаны токен бота и чата, куда будут присылаться заявки.

Ниже приведен код с настройками бота.

# Параметра бота и чата  
token = '5518074987:AAGPbjmEIALGjtM6St\_OxAJfp7oN5s\_kl0s'  
my\_id = -1001865047465  
telegramBot = telepot.Bot(token)  
  
  
# Функция отправки сообщений (заявок)  
def send\_message(text):  
 telegramBot.sendMessage(-1001865047465, text, parse\_mode="Markdown")