Практическое занятие №2

## Задача 1: Доработка модели данных.

Реализация связи многие ко многим в Джано возможна несколькими способами:

1. Автоматическое создание связи, как в примере ниже.

|  |
| --- |
| from django.db import models  class Topping(models.Model):  # ...  pass  class Pizza(models.Model):  # ...  toppings = models.ManyToManyField(Topping) |

В данной реализации явно объявлено две таблицы, но в базе данных будет создана ассоциативная сущность - третья таблицы.

1. Ручное создание ассоциативной сущности, если в ней необходимо сохранить какие-то параметры.

|  |
| --- |
| from django.db import models  class Person(models.Model):  name = models.CharField(max\_length=128)   def \_\_str\_\_(self):  return self.name  class Group(models.Model):  name = models.CharField(max\_length=128)  members = models.ManyToManyField(Person, through='Membership') #в данной строке through='Membership' указывает на таблицу, которая будет использоваться, как ассоциативная сущность.   def \_\_str\_\_(self):  return self.name  class Membership(models.Model):  person = models.ForeignKey(Person, on\_delete=models.CASCADE)  group = models.ForeignKey(Group, on\_delete=models.CASCADE)  date\_joined = models.DateField()  invite\_reason = models.CharField(max\_length=64) |

### Задача: правильно настроить связь между автомобилем, владением и владельцем.

## Задача 2: Работа с представлениями.

В прошлой практической работе было создано первое представление. Те кто был в танке и ничего не понял могут в целях повторения разобрать пример ниже:

### Пример для повторения

После того, как проект готов, мы можем создать представление в geeks/views.py,

|  |
| --- |
| # import Http Response from django from django.http import HttpResponse # get datetime import datetime   # create a function def geeks\_view(request):  # fetch date and time  now = datetime.datetime.now()  # convert to string  html = "Time is {}".format(now)  # return response  return HttpResponse(html) |

Давайте пройдемся по этому коду по одной строке за раз:

* Сначала мы импортируем класс HttpResponse из django.http модуля вместе с библиотекой datetime в Python.
* Далее мы определим функцию с именем geeks\_view. Это функция просмотра. Каждая функция представления принимает объект HttpRequest в качестве первого параметра, который обычно называется запросом.
* Представление возвращает объект HttpResponse, который содержит сгенерированный ответ. Каждая функция представления отвечает за возврат объекта HttpResponse .

Давайте приведем это представление в рабочее состояние, в geeks/urls.py,

|  |
| --- |
| from django.urls import path   # importing views from views..py from .views import geeks\_view   urlpatterns = [  path('time/', geeks\_view), ] |

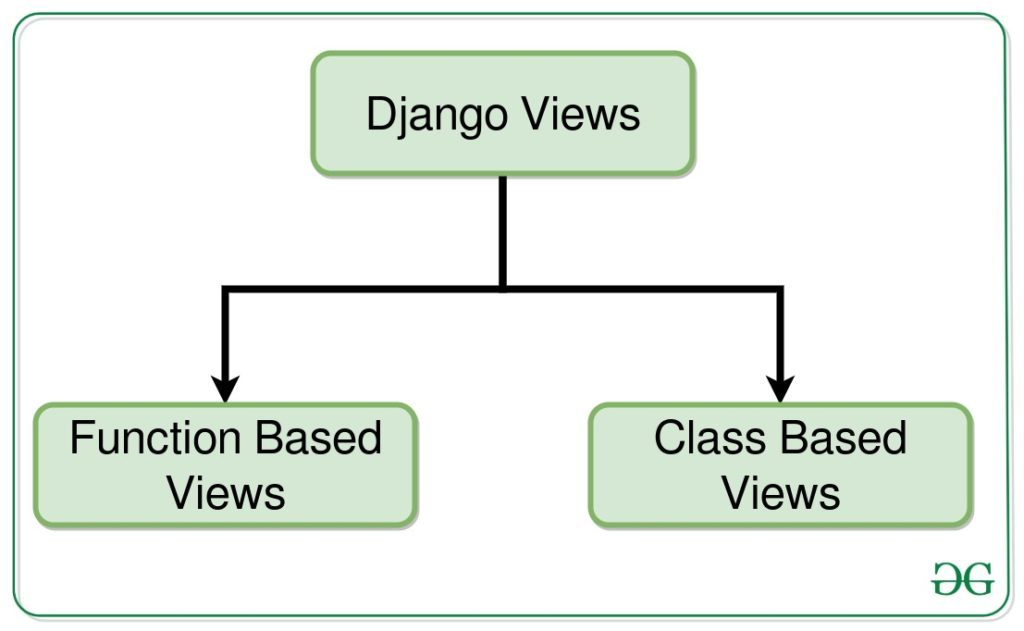
Далее запустите сервер и перейдите посетите средуюзий юрл адрес <http://127.0.0.1:8000/>time/ ,



### Типы контроллеров (views) в django

Представления в Джанго бывают двух типов:

* Представления на основе функций.
* Представления на основе классов.



#### Представления на основе функций

Представления на основе функций пишутся с использованием функции в python, которая получает в качестве аргумента объект HttpRequest и возвращает объект HttpResponse. Представления, основанные на функциях, обычно делятся на 4 основных стратегии, а именно CRUD (Создать, Получить, Обновить, Удалить). CRUD является основой любого фреймворка, который используется для разработки.

Давайте создадим представление списка на основе функций для отображения экземпляров модели.

давайте создадим модель, из которой мы будем создавать экземпляры через наше представление. В project\_first\_app/models.py,

|  |
| --- |
| # import the standard Django Model # from built-in library from django.db import models   # declare a new model with a name "GeeksModel" class GeeksModel(models.Model):    # fields of the model  title = models.CharField(max\_length = 200)  description = models.TextField()    # renames the instances of the model  # with their title name  def \_\_str\_\_(self):  return self.title |

После создания этой модели нам нужно выполнить две команды, чтобы создать базу данных для одной и той же.

|  |
| --- |
| Python manage.py makemigrations  Python manage.py migrate |

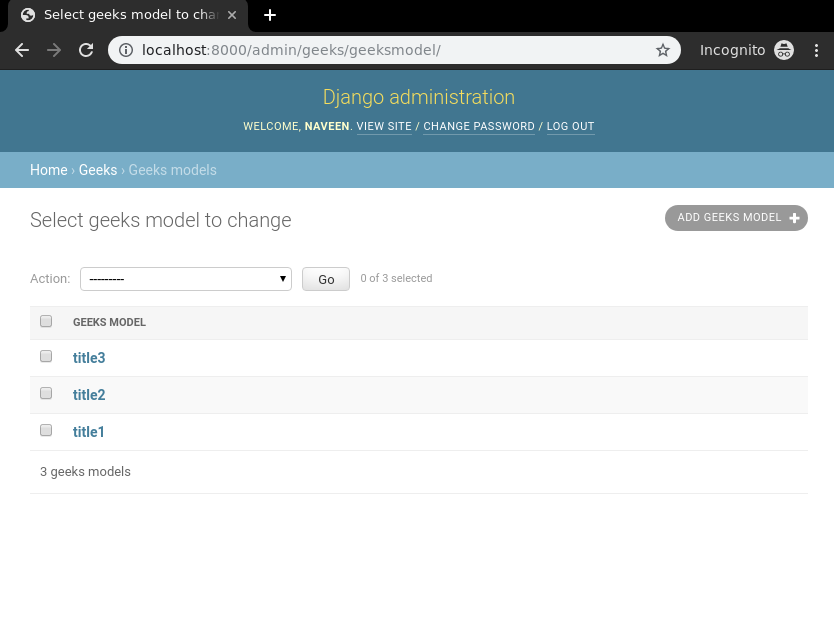
Теперь давайте создадим несколько экземпляров этой модели, используя shell, запустим bash.

|  |
| --- |
| Python manage.py |

Введите следующие команды

|  |
| --- |
| >>> от geeks.models импорт GeeksModel >>> GeeksModel.objects.create (  название = "title1",  Описание = "description1"). сохранить () >>> GeeksModel.objects.create (  название = "title2",  Описание = "description2"). сохранить () >>> GeeksModel.objects.create (  название = "title2",  Описание = "description2"). сохранить () |

Теперь у нас все готово для бэк-энда. Убедитесь, что экземпляры были созданы из <http://localhost:8000/admin/geeks/geeksmodel/>



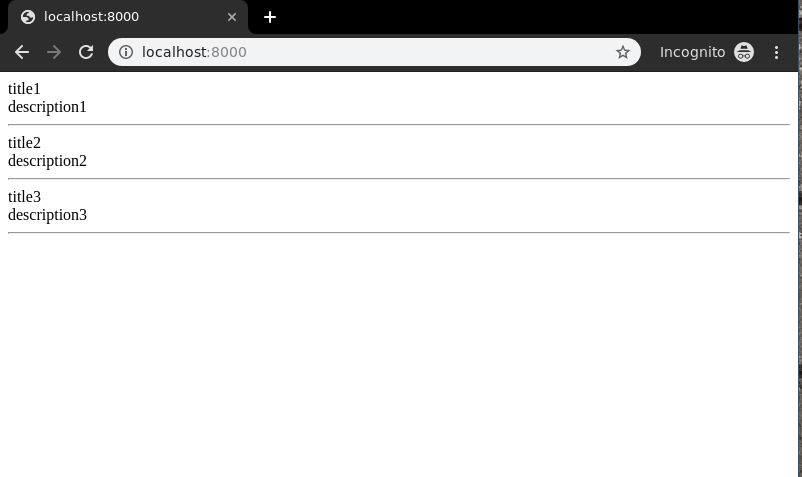
Давайте создадим вид и шаблон для того же самого. В geeks/views.py,

|  |
| --- |
| from django.shortcuts import render   # relative import of forms from .models import GeeksModel     def list\_view(request):  # dictionary for initial data with   # field names as keys  context ={}    # add the dictionary during initialization  context["dataset"] = GeeksModel.objects.all()    return render(request, "list\_view.html", context) |

Создание шаблона в templates/list\_view.html:

|  |
| --- |
| <div class="main">    {% for data in dataset %}.    {{ data.title }}<br/>  {{ data.description }}<br/>  <hr/>    {% endfor %}   </div> |

Давайте проверим, что там находится на <http://localhost:8000/>



Подобные представления на основе функций могут быть использованы для просмотра, создания, добавления и удаления данных.

#### Представления на основе классов

Представления на основе классов предоставляют альтернативный способ реализации представлений в виде объектов Python вместо функций. Они не заменяют функциональные представления, но имеют определенные различия и преимущества по сравнению с функциональными представлениями:

* Организация кода, связанного с конкретными методами HTTP (GET, POST и т. Д.), Может решаться отдельными методами вместо условного ветвления.
* Объектно-ориентированные методы, такие как mixins (множественное наследование), могут использоваться для разделения кода на повторно используемые компоненты.

Представления на основе классов проще и эффективнее в управлении, чем представления на основе функций. Представление на основе функций с тоннами строк кода может быть преобразовано в представления на основе классов только с несколькими строками. Это где объектно-ориентированное программирование вступает в силу.

##### Пример:

В geeks/views.py,

|  |
| --- |
| from django.views.generic.list import ListView from .models import GeeksModel   class GeeksList(ListView):    # specify the model for list view  model = GeeksModel |

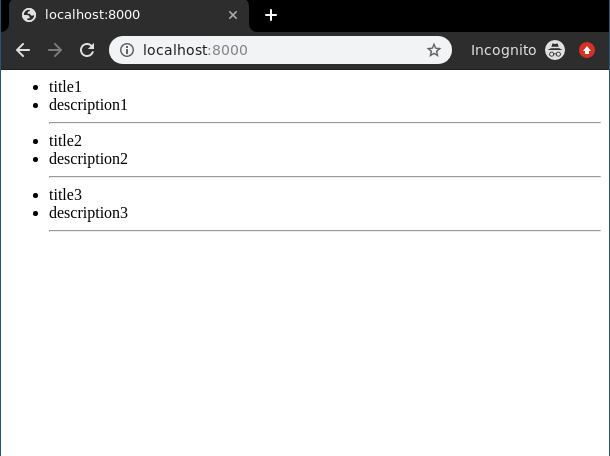
Теперь создайте путь URL, чтобы отобразить представление. В гиках / urls.py,

|  |
| --- |
| from django.urls import path   # importing views from views..py from .views import GeeksList urlpatterns = [  path('', GeeksList.as\_view()), ] |

Создание шаблона в templates/geeks/geeksmodel\_list.html,

|  |
| --- |
| <ul>  <!-- Iterate over object\_list -->  {% for object in object\_list %}  <!-- Display Objects -->  <li>{{ object.title }}</li>  <li>{{ object.description }}</li>    <hr/>  <!-- If objet\_list is empty -->  {% empty %}  <li>No objects yet.</li>  {% endfor %} </ul> |

Давайте проверим, что находится на <http://localhost:8000/>



### Задачи:

1. Реализовать вывод всех владельцев функционально. Добавить данные от трех владельцах минимум. Должны быть реализованы контроллер (views) и шаблоны (temlates).

2. Реализовать вывод всех автомобилей на основе классов. Добавить данные о трех автомобилях минимум. Должны быть реализованы контроллер (views) и шаблоны (temlates).

## Задача 3: Работа с формами и представлениями.

### Формы на основе функций

Необходимо создать ModelForm для этой модели. В этой статье Вы можете посмотреть дополнительную информацию о создании форм осонованных на моделях - [Django ModelForm](https://geeksforgeeks.org/django-modelform-create-form-from-models/). Далее создайте forms.py в папке Вашего приложения.

|  |
| --- |
| from django import forms from .models import GeeksModel     # creating a form class GeeksForm(forms.ModelForm):    # create meta class  class Meta:  # specify model to be used  model = GeeksModel    # specify fields to be used  fields = [  "title",  "description",  ] |

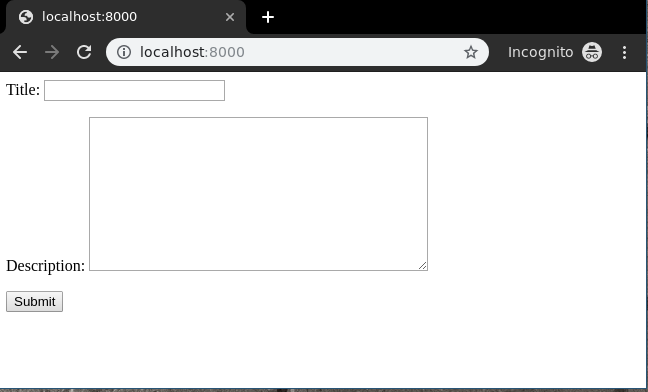
Если Вы все сделали правильно, серверная часть готова. Далее создадим шаблон для клиента. В geeks/views.py:

|  |
| --- |
| from django.shortcuts import render   # relative import of forms from .models import GeeksModel from .forms import GeeksForm #импортируем только-что созданную форму   def create\_view(request):  # dictionary for initial data with   # field names as keys  context ={}    # add the dictionary during initialization  form = GeeksForm(request.POST or None) #создаем экземпляр формы, отсылаем в него данные из формы (из полей в браузере)  if form.is\_valid(): #Проверка формы на корректность (валидация)  form.save()  context['form']= form  return render(request, "create\_view.html", context) |

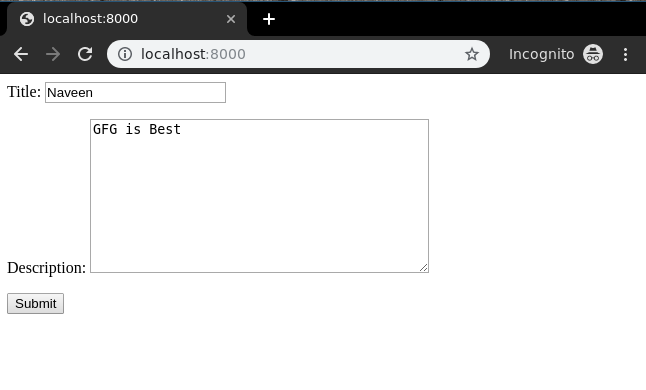
Создание шаблона в templates/create\_view.html:

|  |
| --- |
| <form method="POST" enctype="multipart/form-data">    <!-- Security token -->  {% csrf\_token %}    <!-- Using the formset -->  {{ form.as\_p }}    <input type="submit" value="Submit"> </form> |

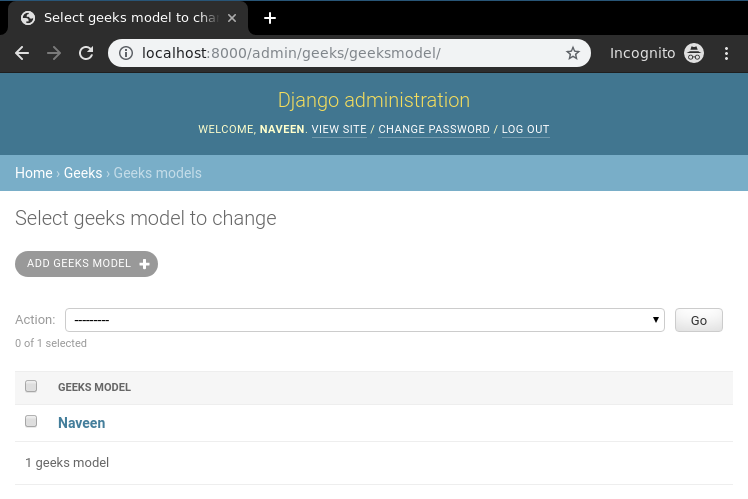
Страница получаемая при запуске сервера <http://localhost:8000/>:



Необходимо попробовать ввести данные в этой форме:



Создание представления работает, и мы можем проверить это, используя экземпляр, созданный через панель администратора.



Таким образом, можно создать представление для модели в Django.

### Формы на основе классов

Представления на основе классов автоматически настраивают взаимодействие с формами. Нужно только указать, для какой модели создать представление Create для полей. Затем CreateView на основе классов автоматически попытается найти шаблон в app\_name/modelname\_form.html. В нашем случае это так geeks/templates/geeks/geeksmodel\_form.html. Давайте создадим наше представление на основе классов. В geeks/views.py:

|  |
| --- |
| from django.views.generic.edit import CreateView from .models import GeeksModel   class GeeksCreate(CreateView):    # specify the model for create view  model = GeeksModel    # specify the fields to be displayed    fields = ['title', 'description'] |

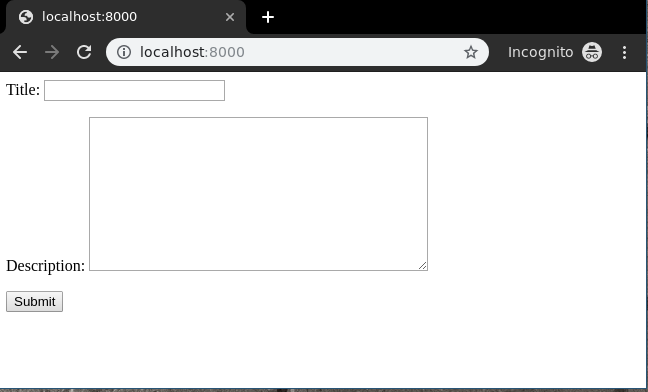
Теперь создайте путь URL, чтобы отобразить представление. В гиках / urls.py,

|  |
| --- |
| from django.urls import path   # importing views from views..py from .views import GeeksCreate urlpatterns = [  path('', GeeksCreate.as\_view() ), ] |

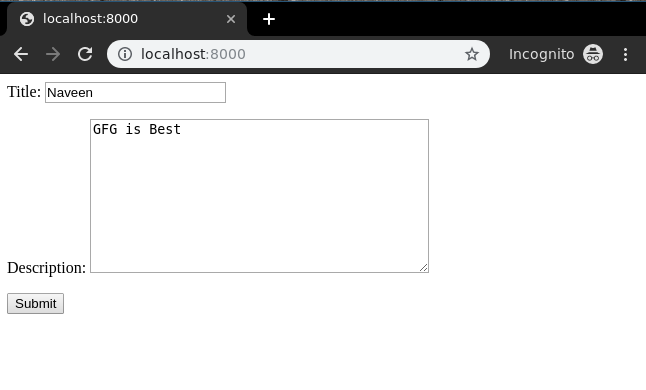
Создание шаблона в templates/geeks/geeksmodel\_form.html,

|  |
| --- |
| <form method="POST" enctype="multipart/form-data">    <!-- Security token -->  {% csrf\_token %}    <!-- Using the formset -->  {{ form.as\_p }}    <input type="submit" value="Submit"> </form> |

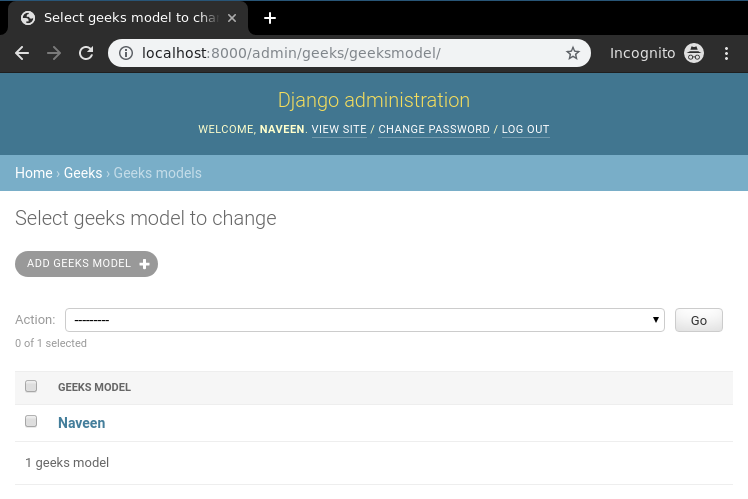
Страница получаемая при запуске сервера <http://localhost:8000/>:



Теперь давайте попробуем ввести данные в этой форме,



Создание представления работает, и мы можем проверить это, используя экземпляр, созданный через панель администратора.



Таким образом, можно создать представление для модели в Django.

### Задачи:

1. Реализовать форму ввода всех владельцев функционально. Добавить данные от **еще** трех владельцах минимум. Должны быть реализованы форма(Form), контроллер (views) и шаблоны (temlates).

2. Реализовать форму ввода всех автомобилей на основе классов. Добавить данные о **еще** трех автомобилях минимум. Должны быть реализованы форма(Form), контроллер (views) и шаблоны (temlates).