**Само задание:**

Разработать информационную систему для соответствующей предметной области. Инструкция к выполнению практической части:   
Для разработки используйте предоставленный сохраненный файл с базой данных. Вам необходимо также разработать дизайн всех страниц для использования со смартфоном с разрешением 390x844 px. Дизайн можно представить в виде файлов изображений .png (отдельное изображение для каждой страницы), либо в виде .html файлов (отдельный файл для каждой страницы).   
Интегрировать дизайн в разрабатываемую информационную систему не требуется.   
Описание предметной области:   
Портал сознательных граждан «Нарушениям.Нет» представляет собой информационную систему для помощи полиции по своевременной фиксации нарушений правил дорожного движения. Перед тем как впервые воспользоваться услугами портала гражданин должен зарегистрироваться. В ходе регистрации он указывает данные о себе (ФИО, телефон, адрес электронной почты), логин и пароль.   
Войдя в систему, гражданин может сформировать заявление, указав номер автомобиля и описание нарушения.   
Заявления граждан хранятся в системе. В каждой заявке описание, номер автомобиля и статус заявки (новое, подтверждено или отклонено). После подачи заявления администратор может подтвердить или отклонить   
заявления.   
Основной функционал информационной системы:   
1. Страница регистрации. На данной странице необходимо предусмотреть добавление пользователя в систему. Пользователю необходимо предоставить возможность ввести логин, пароль, ФИО, телефон и адрес электронной почты. По кнопке «Зарегистрироваться» пользователь должен заноситься в базу.   
2. Страница авторизации. На данной странице необходимо предусмотреть возможность ввода логина и пароля для зарегистрированных пользователей. Попытки некорректного ввода логина и пароля должны сопровождаться сообщениями.   
3. Страница заявлений. На данной странице авторизованный пользователь имеет возможность просмотреть свои заявления со статусами, а также оставить новое заявление.   
4. Страница формирования заявления. Гражданин указывает: государственный регистрационный номер автомобиля и описание нарушения.   
5. Панель администратора. Доступ в панель администратора осуществляется по логину copp и паролю password. В панели администратора видны все заявления (ФИО подавшего, описание нарушения, номер автомобиля и статус заявления). Администратор может сменить статус на подтверждено или отклонено

**Гайд:**

1. **Установка Django:**

Убедитесь, что у вас установлен Django. Если нет, установите его с помощью:

**pip install django**

1. Создание проекта:

Создайте новый проект Django в предварительно готовой папке где хранится проект с помощью следующей команды:

**django-admin startproject demo**

Затем перейдите в созданный каталог проекта:

**cd demo**

1. **Создание приложения:**

Создайте приложение для вашей информационной системы:

**python manage.py startapp demo\_app**

1. Определение моделей:

Определите модели в файле models.py внутри приложения demo\_app модели отвечают за то как будет выглядить ваша база данных, а именно таблицы в ней.

from django.db import models  
from django.contrib.auth.models import AbstractUser  
# Модель Citizen (Гражданин) для хранения информации о зарегистрированных пользователях  
class Citizen(AbstractUser):  
 full\_name = models.CharField(max\_length=255) # Полное имя пользователя  
 phone = models.CharField(max\_length=15) # Номер телефона пользователя  
  
 # Добавляем related\_name для разрешения конфликта  
 groups = models.ManyToManyField(  
 'auth.Group',  
 related\_name='citizen\_set',  
 blank=True,  
 help\_text='The groups this user belongs to. A user will get all permissions granted to each of their groups.',  
 verbose\_name='groups',  
 )  
 user\_permissions = models.ManyToManyField(  
 'auth.Permission',  
 related\_name='citizen\_set',  
 blank=True,  
 help\_text='Specific permissions for this user.',  
 verbose\_name='user permissions',  
 )  
  
 def \_\_str\_\_(self):  
 return self.username  
 def \_\_str\_\_(self):  
 return self.username  
# Модель Violation (Нарушение) для хранения информации о заявлениях о нарушениях правил дорожного движения  
class Violation(models.Model):  
 citizen = models.ForeignKey(Citizen, on\_delete=models.CASCADE) # Связь с моделью Citizen, указывает на зарегистрированного пользователя (гражданина)  
 car\_number = models.CharField(max\_length=20) # Государственный регистрационный номер автомобиля, к которому относится нарушение  
 description = models.TextField() # Описание нарушения  
 status\_choices = [  
 ('NEW', 'Новое'), # Статус нового заявления  
 ('CONFIRMED', 'Подтверждено'), # Статус подтвержденного заявления  
 ('REJECTED', 'Отклонено'), # Статус отклоненного заявления  
 ]  
 status = models.CharField(max\_length=20, choices=status\_choices, default='NEW') # Статус текущего заявления (по умолчанию 'NEW')

просто скопируйте для начало в среду разработки в ней куда понятней выглядит код. Для понимания прокоментил каждую строчку. Обратите внимания что тут я юзаю не стандартную модель пользователя и за чего пишу допольнительно код для решения конфликтов  
  
Если захотите потом могу объяснить как сделать на стандартной!  
  
Последние что надо сделать это в setting добавить строку

AUTH\_USER\_MODEL = 'demo\_app.Citizen'

Нужно это для того чтобы пользователи по дэфолту были нашей модели а не встроенные Django. Строчку можно добавить в любое место.

1. **Создание миграций и применение:**

Выполните следующие команды для создания миграций и применения их к базе данных:

**python manage.py makemigrations**

**python manage.py migrate**

После в папке должна появится база **db.sqlite3** это ваша база данных

1. Создание суперпользователя:

Создайте суперпользователя для доступа к административной панели:

**py manage.py createsuperuser** я создам admin c паролем 123  
выже смотрите по заданию как должен называться admin и какой пароль

после запустите сервер и проверьте работает ли вщщ у вас всё или нет для этого запускаем **python manage.py runserver** и переходим в [**http://127.0.0.1:8000/admin**](http://127.0.0.1:8000/admin)

Если появилась админка и авторизация сработала супер идём дальше.

1. **Создание административной панели:**

Зарегистрируйте модели в административной панели Django в файле admin.py вашего приложения.

# demo\_app/admin.py

from django.contrib import admin

from .models import Citizen, Violation

admin.site.register(Citizen)

admin.site.register(Violation)

Здесь мы просто указали какие таблицы подтягивать в админ панели.  
Осталось подтянуть саму базу в проект для этого в файле **settings.py** вашего проекта добавьте имя вашего приложения в список INSTALLED\_APPS.

INSTALLED\_APPS = [

# ...

'demo\_app',

]

Запускаем и проверяем отобразилось или нет

1. **Вёрстка:**

Ваша задача создать отдельные файлы .html для каждой страницы в папке templates внутри вашего приложения demo\_app.

Саму вёрстку можно посмотреть в проекте который я вам отправил.

После если хотим подключить CSS то в папке demo\_app надо создать папку static и в ней создать папку css и уже в неё положить наш css.

Теперь, чтобы подключить стили к вашим HTML, добавьте следующий код в секцию <head> каждого HTML-файла:

<link rel="stylesheet" type="text/css" href="{% static 'css/styles.css' %}">

1. **Подключаем HTML в Django**

В файле views.py он же файл отвечающий за обработку событий на странице. Надо добавить код обработки html страниц.

from django.shortcuts import render, redirect

from django.contrib.auth import authenticate, login

from django.contrib.auth.decorators import login\_required

from .models import Citizen, Violation

def registration(request):

# Реализация обработки регистрации

return render(request, 'registration.html')

def login(request):

# Реализация обработки авторизации

return render(request, 'login.html')

@login\_required

def statements(request):

return render(request, 'statements.html')

@login\_required

def create\_statement(request):

# Реализация создания нового заявления

return render(request, 'create\_statement.html')

@login\_required

def admin\_panel(request):

# Реализация отображения административной панели

return render(request, 'admin\_panel.html')

def change\_status(request, statement\_id):

# Реализация изменения статуса заявления пока просто заглушка

pass

Осталось добавить ссылки на нашу отработки для этого обратимся к файл urls.py в папке demo\_app если такого файла нет в данной папке создайте его. Такой же файл есть в корневой папке demo но пока не он нам нужен.

# demo\_app/urls.py  
from django.urls import path  
from . import views # импорт views из вашего приложения  
  
urlpatterns = [  
 path('registration/', views.registration, name='registration'),  
 path('login/', views.login, name='login'),  
 path('statements/', views.statements, name='statements'),  
 path('create\_statement/', views.create\_statement, name='create\_statement'),  
 path('admin\_panel/', views.admin\_panel, name='admin\_panel'),  
 path('change\_status/<int:statement\_id>/', views.change\_status, name='change\_status'),  
  
]

После уже в файле urls.py добавим такой код  
  
# demo/urls.py  
from django.contrib import admin  
from django.urls import path, include # добавляем include для включения urls из приложения  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('', include('demo\_app.urls')), # включаем urls из приложения  
]

Здесь мы подключаем ссылки из приложения

Дополнительно в demo\_app/urls.py для удобства в ссылки добавим в начало эту строку

path('', views.registration, name='registration'),

нужна для того чтобы при заходе на сайт не нужно было вводить ссылку на конкретную страницу а уже сразу попадать на первоначальную страницу я поставил регистрацию вы можете и на авторизацию и создать отдельный index приветсвия

1. **Регистрация**

Для реализации функционала регистрации на основе предоставленной модели Citizen, вам нужно внести изменения в файлы views.py и forms.py. Давайте начнем с формы.

Сначала создадим форму для регистрации. Если у вас нет файла forms.py в вашем приложении, создайте его и добавьте следующий код:

from django import forms  
from .models import Citizen  
  
class RegistrationForm(forms.ModelForm):  
 class Meta:  
 model = Citizen  
 fields = ['username', 'password', 'full\_name', 'phone', 'email']  
 widgets = {  
 'password': forms.PasswordInput(), # Поле пароля будет отображаться как пароль  
 }

Эта форма использует модель Citizen и включает поля, которые вы хотите использовать при регистрации. Поле widgets для пароля указывает, что это поле должно отображаться как поле для ввода пароля.

Теперь обновим файл views.py для обработки регистрации. Добавьте следующий код:

Добавим в views импорт

from .forms import RegistrationForm  
from django.contrib.auth import login as auth\_login # Переименовываем функцию login

from django.contrib import messages

from django.contrib.auth.hashers import make\_password

def registration(request):  
 # Реализация обработки регистрации  
 if request.method == 'POST':  
 form = RegistrationForm(request.POST)  
 if form.is\_valid():  
 user = form.save(commit=False)  
 user.password = make\_password(form.cleaned\_data['password']) # Хешируем пароль  
 user.save()  
 messages.success(request, 'Регистрация прошла успешно!')  
 return redirect('index')  
 else:  
 form = RegistrationForm()  
  
 return render(request, 'registration.html', {'form': form})

Обратите внимание на import формы которую мы создали.

Этот код обрабатывает POST-запросы от формы регистрации. Если форма валидна, создается новый объект Citizen, ему присваивается хэшированный пароль с использованием set\_password, и после этого новый гражданин сохраняется в базе данных. После успешной регистрации пользователь автоматически входит в систему с использованием login.  
  
Также обратите внимание что тут мы используем функцию login для автоматической авторизации. Для теста можно убрать эту функцию.

1. **Авторизация**

Для начало обновим ссылку на авторизацию

Замените

path('login/', views.login, name='login'),

на

path('login/', user\_login, name='login'),

Обратите внимание, что мы использовали встроенные классы LoginView и LogoutView для обработки входа. Также в import добавьте

from .views import user\_login

Logout мы сейчас не используем он нам пригодится в будущем для выхода из учётки.

Добавим в views импорт

from django.contrib.auth import authenticate, login as auth\_login

для того чтобы не было конфликтов мы переименовали функцию login на user\_login

Теперь обновим нашу обработку login в views

def user\_login(request):  
 # Реализация обработки авторизации  
 if request.method == 'POST':  
 username = request.POST['username']  
 password = request.POST['password']  
 user = authenticate(request, username=username, password=password)  
 if user is not None:  
 auth\_login(request, user)  
 return redirect('index')  
 else:  
 return render(request, 'login.html', {'error': 'Неверные учетные данные'})  
  
 # Добавим возврат для случая GET-запроса  
 return render(request, 'login.html')

В файле views.py мы добавили новую функцию user\_login, которая обрабатывает вход пользователя. Если введенные учетные данные верны, пользователь входит в систему и перенаправляется на главную страницу (index). В противном случае мы отображаем страницу входа с сообщением об ошибке.

После того как мы это добавили идём тестировать)

1. **Выход**

Импортируем

from django.contrib.auth import logout

добавляем обработку

def user\_logout(request):  
 logout(request)  
 return redirect('index')

Теперь вам нужно создать соответствующий URL-путь и обновить шаблон, например:

Импортируем

from .views import user\_logout

добавляем путь

path('logout/', user\_logout, name='logout'),

После обновляем header

<p>Добро пожаловать, {{ user.username }}! <a href="{% url 'logout' %}">Выход</a></p>

Теперь у вас есть ссылка "Выход" для авторизованных пользователей.

1. **Создаём заявление**

Прежде всего, вам нужно создать форму для создания заявления, а затем обновить функцию create\_statement в соответствии с новой формой. Вот пример:

forms.py:

from .models import Violation

class StatementForm(forms.ModelForm):  
 class Meta:  
 model = Violation  
 fields = ['car\_number', 'description']

views.py:

@login\_required  
def create\_statement(request):  
 # Реализация создания нового заявления  
 if request.method == 'POST':  
 form = StatementForm(request.POST)  
 if form.is\_valid():  
 violation = form.save(commit=False)  
 violation.citizen = request.user # Привязываем заявление к текущему пользователю  
 violation.save()  
 return redirect('index')  
 else:  
 form = StatementForm()  
  
 return render(request, 'create\_statement.html', {'form': form})

create\_statement.html:

<!-- Страница формирования заявления -->  
{% load static %}  
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Формирование заявления</title>  
 <link rel="stylesheet" href="{% static 'css/styles.css' %}">  
</head>  
<body>  
 {% include 'header.html' %}  
 <h1>Страница формирования заявления</h1>  
 <form method="post" action="{% url 'create\_statement' %}">  
 {% csrf\_token %}  
 {{ form.as\_p }}  
 <button type="submit">Создать заявление</button>  
 </form>  
</body>  
</html>

1. **Выводим заявления пользователя**

Чтобы отобразить заявления пользователя, вам нужно получить все заявления, связанные с текущим пользователем, и передать их в шаблон. Вот как вы можете сделать это:

В файле views.py:

@login\_required  
def statements(request):  
 user\_statements = Violation.objects.filter(citizen=request.user)  
 return render(request, 'statements.html', {'user\_statements': user\_statements})

statements.html:

<!-- Страница заявлений -->  
{% load static %}  
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Заявления</title>  
 <link rel="stylesheet" href="{% static 'css/styles.css' %}">  
</head>  
<body>  
 {% include 'header.html' %}  
 <h1>Страница заявлений</h1>  
 <!-- Вывод информации о заявлениях пользователя -->  
 {% if user\_statements %}  
 <ul>  
 {% for statement in user\_statements %}  
 <li>{{ statement.car\_number }} - {{ statement.description }} - {{ statement.status }}</li>  
 {% endfor %}  
 </ul>  
 {% else %}  
 <p>У вас нет заявлений.</p>  
 {% endif %}  
{% endblock %}  
</body>  
</html>

Если возникает ошибка с внешними ключами попробуйте удалить базу и миграции, и создать миграции заново

1. **Админ панель**

Добавим небольшие изменения в моделях а именно добавим поле статус которое содержит либо admin либо user

class Citizen(AbstractUser):  
 STATUS\_CHOICES = [  
 ('Admin', 'Админ'),  
 ('User', 'Пользователь'),  
 ]  
 full\_name = models.CharField(max\_length=255) # Полное имя пользователя  
 phone = models.CharField(max\_length=15) # Номер телефона пользователя  
 status = models.CharField(max\_length=20, choices=STATUS\_CHOICES, default='User')  
 # Добавляем related\_name для разрешения конфликта  
 groups = models.ManyToManyField(  
 'auth.Group',  
 related\_name='citizen\_set',  
 blank=True,  
 help\_text='The groups this user belongs to. A user will get all permissions granted to each of their groups.',  
 verbose\_name='groups',  
 )  
 user\_permissions = models.ManyToManyField(  
 'auth.Permission',  
 related\_name='citizen\_set',  
 blank=True,  
 help\_text='Specific permissions for this user.',  
 verbose\_name='user permissions',  
 )  
  
 def \_\_str\_\_(self):  
 return self.username

Обновим файл admin\_panel.html

<!-- Панель администратора -->  
{% load static %}  
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Панель администратора</title>  
 <link rel="stylesheet" href="{% static 'css/styles.css' %}">  
</head>  
<body>  
 {% include 'header.html' %}  
 <h1>Панель администратора</h1>  
  
 <!-- Вывод информации о заявлениях -->  
 <ul>  
 {% for statement in statements %}  
 <li>  
 {{ statement.citizen.full\_name }} - {{ statement.car\_number }} - {{ statement.status }}  
 - <a href="{% url 'change\_status' statement.id %}">Изменить статус</a>  
 </li>  
 {% endfor %}  
 </ul>  
 <!-- Форма для изменения статуса заявления -->  
  
</body>  
</html>

Создайте файл change\_status.html

{% load static %}  
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Изменение статуса заявления</title>  
 <link rel="stylesheet" href="{% static 'css/styles.css' %}">  
</head>  
<body>  
 <h2>Изменение статуса заявления</h2>  
 <h1>NEW = Новое CONFIRMED=Подтверждено REJECTED=Отклонено</h1>  
 <p>{{ statement.citizen.full\_name }} - {{ statement.car\_number }} - Текущий статус: {{ statement.status }}</p>  
  
 <form method="post" action="{% url 'change\_status' statement.id %}">  
 {% csrf\_token %}  
 <label for="new\_status">Новый статус:</label>  
 <textarea name="new\_status" id="new\_status" rows="4" cols="50">{{ statement.status }}</textarea>  
 <button type="submit">Изменить статус</button>  
 </form>  
  
 <a href="{% url 'admin\_panel' %}">Вернуться к списку заявлений</a>  
</body>  
</html>

Добавим импорты в views

from django.http import HttpResponseForbidden

from django.shortcuts import redirect, get\_object\_or\_404

Добавим обработчик в views

def admin\_panel(request):  
 # Реализация отображения административной панели  
 statements = Violation.objects.all()  
 return render(request, 'admin\_panel.html', {'statements': statements})  
  
def change\_status(request, statement\_id):  
 # Реализация изменения статуса заявления  
 statement = get\_object\_or\_404(Violation, id=statement\_id)  
  
 if request.method == 'POST':  
 if request.user.is\_authenticated and request.user.status == 'Admin':  
 new\_status = request.POST.get('new\_status')  
 statement.status = new\_status  
 statement.save()  
 return redirect('admin\_panel')  
 else:  
 return HttpResponseForbidden("Доступ запрещен. У вас нет прав на изменение статуса.")  
  
 return render(request, 'change\_status.html', {'statement': statement})

Теперь пользователь с статусов админ имеет доступ к вкладке админ панель где может вносить изменения в статус. В админ панели Django это тоже можно потому решать вам нужна она или нет.

На этом гайд закончен, если ты всё сделал у тебя работает то ты молодец, осталось теперь всё понять и запомнить! Нюансов много но если я смог понять то и ты сможешь!