Лабораторная работа №7

Авторизация, работа с формами и Django Admin.

Задание и порядок выполнения

Основная цель данной лабораторной работы – научиться обрабатывать веб-формы на стороне приложения, освоить инструменты, которые предоставляет Django, по работе с формами. Также в этой лабораторной работе вы освоите инструменты Django по работе с авторизацией и реализуете простейшую авторизацию. Напоследок, вы познакомитесь с инструментом администрирования Django – как в несколько строчек кода сделать панель администратора сайта.

1. Создайте view, которая возвращает форму для регистрации.

   Поля формы:
   - Логин
   - Пароль
   - Повторный ввод пароля
   - Email
   - Фамилия
   - Имя

2. Создайте view, которая возвращает форму для авторизации.

   Поля формы:
   - Логин
   - Пароль

3. При отправке формы регистрации во view проверять каждый параметр по правилам валидации, если валидация всех полей пройдена, то создавать пользователя и делать перенаправление на страницу логина, а ошибки, если они есть, выводить над формой.

   Правила валидации:
   - Логин не меньше 5 символов
   - Пароль не меньше 8 символов
   - Пароли должны совпадать
   - Все поля должны быть заполнены
   - Логин – уникален для каждого пользователя

4. При возникновении ошибок в момент отправки формы, введенные значения в полях ввода, кроме пароля, не должны исчезать.

5. Переписать view регистрации с использованием Django Form, правила валидации удалить из view, использовать встроенный механизм валидации полей.

6. Во view авторизации реализовать логин при POST запросе. При успешной авторизации должен происходить переход на страницу успешной авторизации.

7. Страница успешной авторизации должна проверять, что пользователь авторизован. Иначе делать перенаправление на страницу авторизации.

8. Реализовать view для выхода из аккаунта.

9. Заменить проверку на авторизацию на декоратор login_required

10. Добавить superuser'а через комманду manage.py

11. Подключить django.contrib.admin и войти в панель администрирования.

12. Зарегистрировать все свои модели в django.contrib.admin

13. Для выбранной модели настроить страницу администрирования:
    - Настроить вывод необходимых полей в списке
    - Добавить фильтры
    - Добавить поиск
    - Добавить дополнительное поле в список

Теория и примеры

Обработка веб-форм

Создадим простейшую форму в необходимом нам шаблоне:

```
<form method="POST">
  {% csrf_token %}
  <label>
     Логин:
     <input type="text" name="username">
  </label>
  <label>
     Пароль:
     <input type="password" name="password">
  </label>
  <label>
     Повторите ввод:
     <input type="password" name="password2">
  </label>
  <button type="submit">Зарегистрировать</button>
</form>
```

Так как мы создаем POST форму (которая может привести к изменениям данных), нам следует побеспокоится о Cross Site Request Forgeries. Благодаря Django это очень просто. В общем, все POST формы должны использовать тег {% csrf_token %}.

После того, как пользователь нажмет на кнопку “Зарегистрировать”, в ту же самую view, отправится POST запрос в теле которого будут содержаться параметры нашего запроса. Для доступа к этим параметрам необходимо обращаться к словарю переменной request – request.POST. Простейшая view, которая обрабатывает данную форму, производит валидацию и регистрирует пользователя может выглядить вот так:

```
def registration(request):
  errors = []
  if request.method == 'POST':
    username = request.POST.get('username')
    if not username:
      errors.append('Введите логин')
    elif len(username) < 5:
      errors.append('Логин должен привышать 5 символов')

    password = request.POST.get('password')
    if not password:
      errors.append('Введите пароль')
    elif len(password) < 6:
      errors.append('Длинна пароля должна превышать 6
символов')
    password_repeat = request.POST.get('password2')

    if password != password_repeat:
      errors.append('Пароли должны совпадать')

    if not errors:
      #              ...

      return HttpResponseRedirect('/login/')

  return render(request, 'registration.html', {'errors': errors})
```

Прим.: Если пользователь не передал необходимы параметр, то при обращении по ключу (request.POST[‘password’]) будет выкинуто исключение KeyError. Поэтому надо сначала проверять наличие того или иного элемента в словаре перед его использованием, либо использовать методы get()/setdefault().

Django Forms

При увеличении количества параметров и правил валидации view будут становиться громоздкими и запутанными. В Django предусмотрен встроенный механизм для работы с формами.

Сердце всего механизма – класс Form. Как и модель в Django, которая описывает структуру объекта, его поведение и представление, Form описывает форму, как она работает и показывается пользователю.

Как поля модели представляют поля в базе данных, поля формы представляют HTML <input> элементы. (ModelForm отображает поля модели в виде HTML <input> элементов, используя Form. Используется в админке Django.)

Поля формы сами являются классами. Они управляют данными формы и выполняют их проверку при отправке формы. Например, DateField и FileField работают с разными данными и выполняют разные действия с ними.

Поле формы представлено в браузере HTML “виджетом” - компонент интерфейса. Каждый тип поля представлен по умолчанию определенным классом Widget, который можно переопределить при необходимости.

Создание форм в Django

Попробуем заменить ручную обработку всех параметров POST запроса на форму. Создадим класс формы:

```
class RegistrationForm(forms.Form):
  username = forms.CharField(min_length=5, label='Логин')
  password = forms.CharField(min_length=6, widget=forms.PasswordInput, label='Пароль')
  password2 = forms.CharField(min_length=6, widget=forms.PasswordInput,
label='Повторите ввод')
```

В качестве widget у поля пароля мы указали forms.PasswordInput – это ничто иное, как <input type=”password” …>, по-умолчанию у CharField type=”text”.

Теперь отредактируем саму view:

```
def registration(request):
  if request.method == 'POST':
    form = RegistrationForm(request.POST)
    if form.is_valid():
      #          ...

      return HttpResponseRedirect('/login/')
  else:
    form = RegistrationForm()

  return render(request, 'registration.html', {'form': form})
```

Видите? Кода стало гораздо меньше, волшебный метод is_valid() все проверки сделает за нас, при этом, если возникнут ошибки, то передаст их в переменную формы errors, которую можно вывести в шаблоне.

Но это еще не все, мы можем сократить и наш шаблон:

```
<form method="POST">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Зарегистрироваться</button>
</form>
```

Метод as_p() преобразует наш объект формы в HTML форму, и сам позаботится о выводе ошибок.

Подробнее о формах вы можете прочесть в документации:
http://djbook.ru/rel1.9/topics/forms/index.html

Авторизация

Для начала работы с модулем авторизации необходимо проверить, что файле settings.py в переменной INSTALLED_APPS есть **'django.contrib.auth'**, и что в MIDDLEWARE есть **'django.contrib.sessions.middleware.SessionMiddleware'** и **'django.contrib.auth.middleware.AuthenticationMiddleware'**

Миграции

Пример кода авторизации:

```
user = authenticate(username=username, password=password)
if user:
  login(request, user)
  return HttpResponseRedirect('/success/')
```

Логаут использует

```
request.user
request.user.is_authenticated()
```