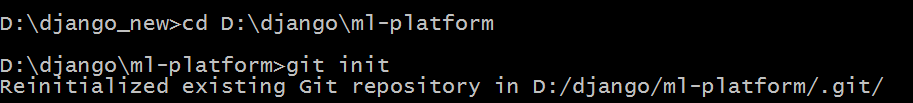
# Git

## Работа с локальным репозиторием

### Создание локального репозитория

Создание локального (на локальном компьетере) репозитория возможно двумя способами

Инициализация репозитория через существущую папку (делаем существующую папку репозиторием) git init



Git clone url (пишем свой адрес) (клонируем удаленный репозиторий с гитхаба)

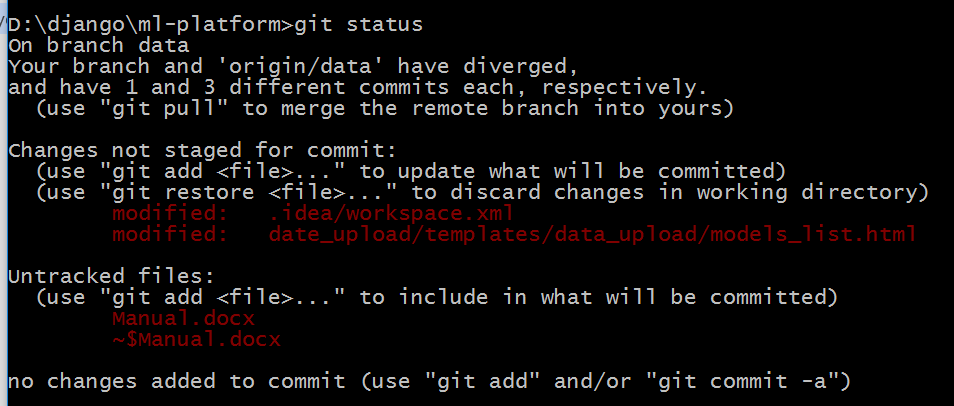
### Проверка статуса

Git status

Показывает в каком мы находимся бранче (отличия нашей локальной версии и текущего бранча)

есть ли там неотслеживаемые файлы (неотслеживаемые в смысле контроля версионности) untracked

Не добавленные в стейджинг (changes not staged for commit)



### Добавление файлов

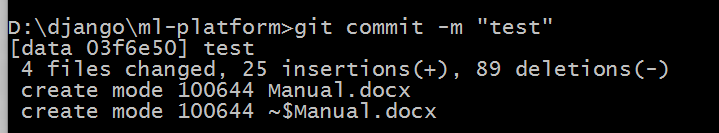
#### Git add .

Точка означает добавить все файлы

После изменений нужно делать git add иначе измененная часть файла не добавится в режим отслеживания

#### Git commit –m “message”

Фактически это записать изменений в файл .git



## Работа с удаленным репозиторием

### Добавление удаленного репозитория

Git remote add origin https:\git url

Подключение к нему git push –u origin branchname (заливаем данные в удаленный репозиторий)

### Мердж локального репозитория в удаленным git push

### Скачивания из удаленного репозитория в локальный git pull

### Проверка текушего удаленного репозитория git remote –v

### Изменения ссылки на удаленный репозиторий

Здесь важно указать, что имя репозитория - origin



# Django

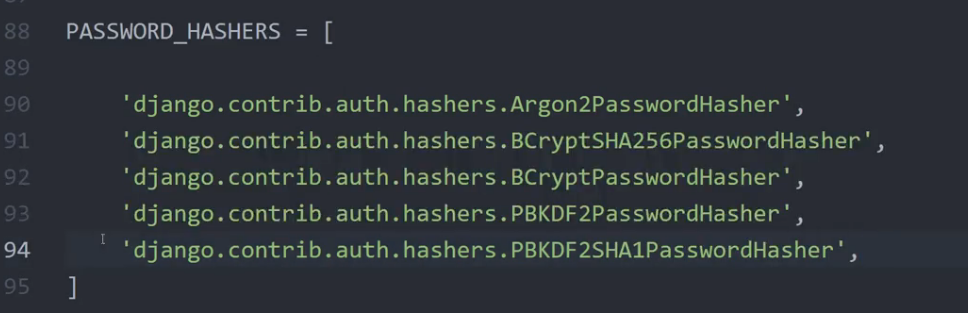
## Authorization

### Settings.py

#### Password hasher

Пользовательские пароли необходимо хэшировать, то есть хранить в зашифроанном виде.

В джанго есть встроенный набор хэшероав. Для некоторых нужно скачать дополнительные библиотеки. Джагно будет использовать их по порядку, нужно чтобы хотя бы один был установлен. При этом последний идет в базовой комплектации в джанго.



#### Password validation

Это кусок кода вводит валидацию пользовательского пароля (правила допустимых паролей, чтобы не были слишком простыми например)

AUTH\_PASSWORD\_VALIDATORS = [  
 {  
 'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',  
 },  
]

### Models.py

Создание модели хранения данных о пользователях. Это фактические создание в базе таблицы User, в которой будет храниться информация по нашим юзерам. При желании можно добавить дополнительные атрибуты юзера.

from django.db import models  
from django.contrib.auth.models import User

class UserProfileInfo (models.Model):  
  
 user = models.OneToOneField(User,on\_delete=models.CASCADE)  
  
 def \_\_str\_\_(self):  
 return self.user.username

### Forms.py

Создаем форму для регистрации юзера. Создаем форму, которая будет запрашивать пароль и обращаться к модели данных юзера.

from django.contrib.auth.models import User

class UserForm (forms.ModelForm):  
 password = forms.CharField(widget=forms.PasswordInput())  
  
 class Meta():  
 model = User  
 fields = ['username','email','password']

### Admin.py

Регистрация созданной модели, чтобы иметь к ней доступ через админскую консоль

from django.contrib import admin  
  
from date\_upload.models import UserProfileInfo  
  
admin.site.register(UserProfileInfo)

### Templates

#### Registratrion.htm

{%extends "data\_upload/base.html"%}  
  
{% block body\_block %}  
  
<div class="jumbotron">  
 {% if registered %}  
 <h1>Thank you for registration</h1>  
 {% else %}  
 <h1>Register here</h1>  
 <form enctype="multipart/form-data" method="post" >  
 {% csrf\_token %}  
 {{ user\_form.as\_p}}  
 {{ profile\_form.as\_p }}  
 <input type = "submit" name = "" value = "Register ">  
 </form>  
 {% endif %}  
</div>  
  
{% endblock %}

#### Index.html

Прописываем наследование от базового тимплейта

{%extends "data\_upload/base.html"%}  
  
{% block body\_block %}  
  
<div class="jumbotron">  
 <h1>Welcome Vovan</h1>  
</div>  
  
{% endblock %}

#### Base.html

Базовый тимплейт от которого будет все наследоваться

Здесь мы прописываем навигационную панель, которая будет на всех страницах

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Base</title>  
</head>  
<body>  
 <nav class="navbar navbar-expand-lg navbar-light bg-light">  
 <div class="navbar-nav">  
 {# Django Home Link / Admin Link / Register Link#}  
 <a class="navbar-brand" href="{% url 'index' %}">Home</a>  
 <a class="nav-item nav-link" href="{% url 'admin:index' %}">Admin</a>  
 <a class="nav-item nav-link" href="{% url 'basic\_app:register' %}">Register</a>  
  
 {# Some logic on what to display for last item#}  
 {% if user.is\_authenticated %}  
 <a class="nav-link" href="{% url 'logout' %}">Logout</a>  
 {% else %}  
 <a class="nav-link" href="{% url 'basic\_app:user\_login' %}">Login</a>  
 {% endif %}  
 </div>  
 </nav>  
  
 <div class="container">  
 {% block body\_block %}  
 {% endblock %}  
 </div>  
  
  
 </body>  
</html>

### Urls.py

Добавляем возможность подключения файла urls из приложения date\_upload с помощью функции include

from django.conf.urls import url, include

urlpatterns = [  
  
 url(r'^$',views.index,name='index'),  
 url(r'^admin/',admin.site.urls),  
 url(r'^date\_upload/',include('date\_upload.urls')),  
 url (r'^formpage/',views.form\_name,name='form\_name'),  
 url (r'^models/models\_list/',views.models\_data,name='models\_data'),  
 url (r'^models/model\_upload/',views.models\_data\_upload,name='model\_upload'),  
]

urls.py в приложении

from django.conf.urls import url  
from date\_upload import views  
  
app\_name = 'data\_upload'  
  
urlpatterns = [  
 url(r'^register/$',views.register,name='register')  
]

### Views.py

def register(request):  
  
 registered = False  
  
 if request.method == 'POST':  
  
 # Get info from "both" forms  
 # It appears as one form to the user on the .html page  
 user\_form = UserForm(data=request.POST)  
  
 # Check to see both forms are valid  
 if user\_form.is\_valid():  
  
 # Save User Form to Database  
 user = user\_form.save()  
  
 # Hash the password  
 user.set\_password(user.password)  
  
 # Update with Hashed password  
 user.save()  
  
 # Registration Successful!  
 registered = True  
  
 else:  
 # One of the forms was invalid if this else gets called.  
 print(user\_form.errors)  
  
 else:  
 # Was not an HTTP post so we just render the forms as blank.  
 user\_form = UserForm()  
  
 # This is the render and context dictionary to feed  
 # back to the registration.html file page.  
 return render(request,'data\_upload/registration.html',  
 {'user\_form':user\_form,  
 'registered':registered})

### Superuser creation

