Лабораторная работа № 5: Веб-программирование с использованием фреймворка Django

1. Запустите онлайн среду разработки "Codeanywhere" с рабочим пространством для программирования на Python с использованием фреймворка Django версии 1.9 по ссылке:

`https://codeanywhere.com/`

Для входа в систему необходимо иметь учетную запись на одном из следующих сервисов: Google+, GitHub, Bitbucket, Facebook. После входа в аккаунт аутентифицируйте приложение, перейдя по ссылке в письме, присланном на электронную почту аккаунта. Из предложенного списка платформ в среде "Codeanywhere" (окно "Connection Wizard") необходимо выбрать Django (Ubuntu 14.04) и в поле "Name", расположенном в верхней части окна, ввести имя проекта. Нажатием кнопки "Create" Вы подтверждаете создание проекта с указанными параметрами. Будет создано два каталога:

- `django` – содержит файлы фреймворка Django;
- `projectname` – содержит файлы проекта.

Фреймворк Django

Django является свободно распространяемым фреймворком для написания веб-приложений на языке программирования Python. В качестве основы для проектирования веб-приложений используется шаблон проектирования MVC (Model-View-Controller), включающий в себя:

- Model (Модель) – служит для предоставления запрошенных пользователем данных, может изменять свое состояние в зависимости от полученных от контроллера команд.
- View (Представление) – служит для отображения данных модели пользователю, может изменять выдаваемые данные в зависимости от изменений модели.
- Controller (Контроллер) – служит для обработки действий пользователя, управляет изменениями, происходящими в модели.

При взаимодействии с интерфейсом веб-приложения пользователь отдает команды контроллеру (например, добавить запись в базу данных), контроллер обрабатывает команду и обновляет модель (запись добавлена в базу данных), после чего страница веб-приложения (представление) обновляется (запись отображается пользователю).

Основные файлы проекта

В список основных файлов проекта входят следующие:

- `manage.py` – устанавливает настройки для проекта из указанного файла настроек;
- `projectname/settings.py` – файл настроек проекта, хранящийся в каталоге `projectname`;
- `projectname/urls.py` – файл настроек URL (ссылок) для каждого представления, хранящийся в каталоге `projectname`;
- `projectname/wsgi.py` – устанавливает настройки для проекта при необходимости использования WSGI-технологии;
- `projectname/__init__.py` – файл, указывающий Python, что текущий каталог является пакетом Python.

Рассмотрим настройки проекта, размещаемые в файле `settings.py`. Изначально в настройках указываются:

- `BASE_DIR` – путь к основной папке проекта;
- `SECRET_KEY` – секретный ключ, служащий для обеспечения дополнительной безопасности данных;
- `DEBUG` – флаг, идентифицирующий запуск проекта в режиме тестирования;
- `ALLOWED_HOSTS` – позволяет указать варианты записи домена и субдоменов веб-приложения для обеспечения дополнительной безопасности;
- `INSTALLED_APPS` – список всех приложений проекта, включая базовые приложения Django (sessions, messages и т.д.). Для подключения собственных разработанных приложений необходимо указать путь к папкам приложений, используя точку в качестве разделителя;
- `MIDDLEWARE_CLASSES` – список подключенных приложений фреймворка `MIDDLEWARE`, позволяющих использовать базовые обработчики некоторых действий пользователя, например, связывать запрос пользователя с текущим идентификатором сессии;
- `ROOT_URLCONF` – содержит путь к основному файлу списка URL (`urls.py`);
- `TEMPLATES` – описывает настройки шаблонов серверной (backend) и клиентской (представления) частей проекта, путь к папкам, в которых хранятся шаблоны, рекомендуется делать относительным с записью в следующем виде: `os.path.join(BASE_DIR, 'templates')`, где `join` – стандартная функция объединения строк для двух путей: главного пути `os.path`, пути `BASE_DIR`, и пути к нужному каталогу `templates`, переданных в качестве аргумента. Пользовательские шаблоны настраиваются в параметре `DIRS`;
- `WSGI_APPLICATION` – путь к приложению WSGI;
- `DATABASES` – словарь, описывающий основные настройки используемой базы данных:
    - `ENGINE` – драйвер базы данных, указывается один из существующих в Django драйверов:
        - Для MySQL: `'django.db.backends.mysql'`
        - Для PostgreSQL: `'django.db.backends.postgresql'`
        - Для SQLite: `'django.db.backends.sqlite3'`
        - Для Oracle: `'django.db.backends.oracle'`
    - `NAME` – имя базы данных;
    - `USER` – имя пользователя для автоматического подключения к базе данных;
    - `PASSWORD` – пароль для автоматического подключения к базе данных;
    - `HOST` – адрес, на котором расположена база данных;
    - `PORT` – порт, на котором расположена база данных;
- `AUTH_PASSWORD_VALIDATORS` – словарь, приводящий описание валидаторов, отвечающих за проверку сложности паролей пользователей;
- `TIME_ZONE` – временная зона сервера
- `LANGUAGE_CODE` – язык приложений проекта, необходимо, чтобы флаг `USE_I18N` был установлен как `True`;
- `USE_I18N` – флаг, показывающий статус включения системы перевода Django (подстановки нужного языка);
- `USE_L10N` – флаг, показывающий статус включения системы локализации Django, если флаг установлен как `True`, то время, дата и т.п. будут отображаться в

соответствии с установленной локализацией;

- `USE_TZ` – флаг использования параметра `TIME_ZONE`;
- `STATIC_URL` – путь к статическим файлам, например, CSS-стилям. Для указания множества папок собственных статических файлов необходимо использование параметра `STATICFILES_DIRS`.

Для добавления дополнительных страниц и адресов ссылок в проект необходимо в файл `urls.py` добавить соответствующую ссылку, например:

`url(r'^admin/', admin.site.urls)` – подключение всех ссылок указанных в файле `urls.py` каталога `"admin/site/"` (базовый каталог Django из пакета `django.contrib`).

Для указания конкретной функции, вызываемой при переходе по ссылке, используют следующий формат записи:

```
url(r'^auth/$', views.auth, name='auth'),
```

где:

- `r'^auth/$'` – ссылка для перехода
- `views.auth` – путь к функции `auth` в файле `views`, подключаемом к `urls.py`
- `name='auth'` – сокращенное для перехода по ссылкам напрямую из кода проекта

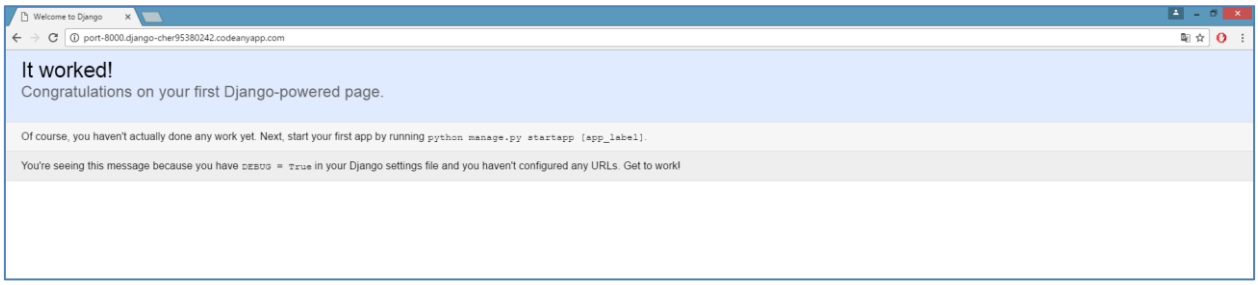
Для подключения обработчиков представлений для указания имен функций используется стандартная запись с указанием имени файла (`views`), содержащего нужные функции:

```
from projectname import views
```

Запуск проекта из консоли осуществляется с помощью команды

```
python manage.py runserver
```

2. Запустите созданный проект. При работе в среде Codeanywhere можно использовать функцию "Run project". Перейдите по ссылке проекта, указанной в конфигурации и справке к проекту, и убедитесь, что проект успешно запускается, при этом на странице должно отображаться следующее:



3. Создайте каталог `templates` в каталоге проекта. Добавьте в каталог файл `index.html`:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>
        {% block title %}
            Hello
        {% endblock %}
    </title>
</head>
<body>
    <div class="header">
        Hello World from render!
    </div>
</body>
</html>
```

Установите путь к каталогу в параметре `TEMPLATES.DIRS` в настройках проекта.

4. Создайте файл `views.py`:

```
from django.http import HttpResponse
from django.shortcuts import render

def index(request):
    return HttpResponse("Hello, world!")

def indexRender(request):
    return render(request, 'index.html', {})
```

5. Дополните файл `urls.py`, добавьте ссылки:
   - пустая ссылка для главной страницы, которую обрабатывает функция `index`
   - ссылка "hello", которую обрабатывает функция `indexRender`

6. Запустите проект. Перейдите по ссылкам на главную страницу и страницу hello. Удостоверьтесь, что при переходе по ссылкам выводится следующая информация:
   - при переходе на главную страницу:
     `Hello, world!`
   - при переходе на страницу hello:
     `Hello World from render!`

Работа с представлениями

Для обработки представлений и отображения их пользователю необходимо создать файл, содержащий функции обработки запросов пользователя и необходимых данных (например, `views.py`). Обработчики представляют собой функции, возвращающие в качестве результата, например, готовое представление или же результат выполнения переадресации на другую страницу. В качестве аргументов обязательно принимают объект, представляющий запрос пользователя, из которого в дальнейшем могут быть получены значения переданных на сервер параметров. Для этого используются

следующие функции:

- `HttpResponse` – отправляет ответ сервера пользователю, ответ указывается в скобках после имени функции: `HttpResponse(…);`
- `HttpResponseRedirect` (пакет `django.http`) – осуществляет переадресацию на указанный в скобках адрес: `HttpResponseRedirect(…);`
- `render` – рендеринг (подготовка) страницы для выдачи пользователю, в скобках после имени функции указывается объект запроса, адрес шаблона, словарь передаваемых параметров, например:
  `render(request, 'photoorg/albumimages.html', {'isAuthor': isAuthor, 'album': album, 'images': imagesToDisplay})`
- `reverse` (пакет `django.core.urlresolvers`) – получение ссылки на страницу по описанию, также могут быть переданы дополнительные аргументы.

Для выполнения переадресации на другую страницу, ссылку на которую необходимо получить, используется следующий формат записи:

```
HttpResponseRedirect(reverse('photos:images', args=(albumid,)))
```

Чаще всего в виде представлений в веб-приложениях выступают HTML-страницы, сформированные вручную или же с помощью некоторого шаблона. В Django используется собственный шаблонизатор, позволяющий удобно работать с передаваемыми данными и форматом страниц.

Для хранения шаблонов страниц веб-приложений создается папка с шаблонами, путь к которой указывается в соответствующем параметре в настройках. Для работы с шаблоном необходимо вставить необходимый блок кода в фигурные скобки вида `{% … %}`. Подстановка значения переданных переменных осуществляется путем указания имени переменной в двойных фигурных скобках, например `{{surname}}` или `{{person.surname}}`.

Наиболее часто используемые функции шаблонов:

- `{% load staticfiles %}` – обеспечивает подгрузку статических файлов, например, стилей, к странице, прописывается в начале шаблона;
- `{% static "styles.css" %}` – подключает статический файл (например, файл CSS-стилей `styles.css`) к странице, находя его в указанных с помощью `STATICFILES_DIRS` папках;
- `{% extends 'templatePath' %}` – указывает странице, что текущий шаблон является расширением шаблона с путем `templatePath`, может использоваться с целью упрощения кода, например, не будет необходимости добавлять в каждый шаблон шапку главной страницы и т.д., прописывается в начале шаблона;
- `{% url mainProcessor:urlName %}` – подстановка (например, в тег `<a></a>`) ссылки `urlName` (сокращенное имя ссылки) из `mainProcessor` (имя приложения проекта);
- `{% block blockTitle %} … {% endblock %}` – позволяет определять блоки с определенным именем (`blockTitle`) и указанным содержимым на странице, при наследовании шаблонов может использоваться для сохранения местоположения блока на странице, но изменения его содержимого (в этом случае прописывается в начале шаблона);
- `{% for variable in container %} … {% endfor %}` – цикл `for` для итерации по массиву переданных в шаблон данных, например:

```
{% for a in publicAlbums %}
     <option value="{{ a.id }}">{{ a.title }}</option>
{% endfor %}
```

Здесь `a` – объект последовательности общедоступных альбомов (`publicAlbums`), для которого создается элемент страницы `<option>` с подстановкой соответствующего идентификатора (`a.id`) и названия (`a.title`).

- `{% if condition %}…{% elif %}…{% else %} {% endif %}` – условие (блоки `elif` и `else` являются необязательными), например:

```
{% if albumList %}
    Number of albums: {{ albumList.length }}
{% else %}
    No albums.
{% endif %}
```

С полным списком доступных блоков шаблонов Django можно ознакомиться по ссылке:

`https://docs.djangoproject.com/en/1.9/ref/templates/builtins/`

Формат JSON

JSON является форматом, удобным для передачи, обработки, изменения и доступа к данным, который представляется аналогично ассоциативному массиву парами ключ-значение:

```
{
   "firstname": "Иван",
   "lastname": "Иванов",
   "address": {
       "streetAddress": "Kroverksky pr. 49",
       "city": "Saint Petersburg",
       "postalCode": 197101
   },
   "education": [
       "school": {
           "number": 123,
           "name": "Global school"
       },
       "university": {
           "name": "ITMO",
           "yearStart": 2011,
           "yearEnd": 2015
       },

   ]
}
```

Поля (ключи) объекта JSON представляют собой строки, в то время как их значения, указанные через двоеточие, могут являться следующими типами данных:

- число;
- строка;
- одномерный массив;

- литералы: `true`, `false`, `null`;
- объекты, также представляемые в аналогичном JSON формате

Для работы с форматом JSON в языке Python существует специальный пакет `json`, предоставляющий функции для разбора, создания и обработки данных этого формата, например:

- `dumps(obj, [..])` – производит сериализацию (перевод) объекта `obj` в формат JSON, также при вызове функции могут быть указаны дополнительные параметры, ознакомиться с которыми можно по приведенной ниже ссылке;
- `loads(jsonObj, [..])` – производит десериализацию данных в формате JSON в списки, словари и т.д., также при вызове функции могут быть указаны дополнительные параметры

Стандартное расширение файла для данных типа JSON: `.json`

С полным списком функций и их аргументов можно ознакомиться по ссылке:

`https://docs.python.org/3/library/json.html`

7. Создайте файл `json_data.json` со структурой согласно описанию:

Каждый человек (руководитель, сотрудник, студент) в описанной структуре характеризуется фамилией, именем, отчеством, должностью и имеет табельный номер. Каждое подразделение (ректорат, бухгалтерия, мегафакультет, факультет, кафедра) помимо указанных в описании данных характеризуется идентификатором, названием, имеет руководителя (информация о руководителе представляется согласно приведенному описанию человека) и сотрудников, представленных в виде списка описывающих их объектов.
Университет характеризуется названием, местоположением (с указанием отдельно индекса, города и адреса) и имеет руководителя. В университете существует два типа подразделений: административные и научно-образовательные. К административным подразделениям относятся ректорат и бухгалтерия, описанные характеристиками подразделений, указанными выше. К научно-образовательным подразделениям относятся мегафакультеты, каждый из которых также имеет список факультетов. Факультет помимо базовых параметров также содержит список кафедр факультета, для которых список сотрудников представлен списком преподавателей кафедры. Также кафедра имеет список образовательных программ, каждая из которых характеризуется названием, номером, руководителем, дисциплиной (указывается название дисциплины) и годом обучения. Год обучения представляет собой структуру, для которой прописывается непосредственно год обучения в виде строки (ограничиться годом 2016/2017) и список учебных групп для конкретного года обучения, каждая из которых включает название учебной группы и список обучающихся в ней студентов, для которых вместо должности указывается размер получаемой стипендии или null в случае ее отсутствия.

Заполните структуру данными по Университету ИТМО (идентификаторы могут быть придуманы самостоятельно). Приведите не менее 3 различных примеров вложенных объектов для каждого объекта-списка.

8. Создайте шаблон `universityInfo.html` и соответствующие ему ссылку `universityInfo` и обработчик запроса пользователя. Выведите на страницу данные по университету в указанном формате, подставив нужные значения, полученные из файла `json_data.json`. Формат вывода данных:

```
Университет: ..
Ректор: ..
Местоположение: индекс, город, адрес
Кол-во административных подразделений: ..
Кол-во научно-образовательных подразделений: ..
Кол-во мегафакультетов: ..
Кол-во факультетов: ..
Кол-во кафедр: ..
```

9. Создайте шаблон `disciplineInfo.html` и соответствующие ему ссылку `disciplineInfo` и обработчик запроса пользователя. Выведите на страницу данные по дисциплине в следующем формате, подставив вместо многоточий нужные значения, полученные из файла `json_data.json`:

```
Номер программы: ..
Название программы: ..
Дисциплина: ..
Год обучения: ..
Кол-во групп: ..
```

10. Создайте шаблон `groupsInfo.html` и соответствующие ему ссылку `groupsInfo` и обработчик запроса пользователя. Выведите на страницу данные по студентам всех групп в указанном формате, подставив нужные значения, полученные из файла `json_data.json`. Формат вывода:

Группа	Табельный номер	Студент	Стипендия
Q1234	120	Фамилия Имя Отчество	10 000
...	...	...	...

11. Создайте шаблон `departmentsInfo.html` и соответствующие ему ссылку `departmentsInfo` и обработчик запроса пользователя. Выведите на страницу данные по кафедрам в указанном формате, подставив нужные значения, полученные из файла `json_data.json`. Формат вывода данных:

```
ID: ..
Кафедра: ..
Заф. кафедры: Фамилия Имя Отчество (таб.номер)
Преподаватели:
```

1. Фамилия Имя Отчество (таб.номер, должность)
2. Фамилия Имя Отчество (таб.номер, должность)

…

```
Образовательные программы:
```

- Номер программы (Название, Дисциплина)

Между каждыми двумя кафедрами делайте пропуск двух строк при выводе информации.

12. Создайте шаблон `universityStructure.html` и соответствующие ему ссылку `universityStructure` и обработчик запроса пользователя. Выведите на страницу структуру университета с указанием идентификаторов подразделений и их названий, а также ФИО руководителей (Фамилия И.О.) в указанном формате, подставив нужные значения, полученные из файла `json_data.json`. Формат вывода данных:

```
Университет .. (Фамилия И.О. руководителя)
|_Административные подразделения:
| |_Ректорат (ID,..)
| |_Бухгалтерия (ID,..)
|
|_Научно-образовательные подразделения:
  |_Мегафакультет .. (ID,..)
  | |_Факультет .. (ID,..)
  | | |_Кафедра .. (ID,..)
  | | |_Кафедра .. (ID,..)
  | | |_ ...
  | | |_Кафедра .. (ID,..)
  | |_Факультет .. (ID,..)
  | | ...
  | |_Факультет .. (ID,..)
  | ...
  |_Мегафакультет .. (ID,..)
  | ...
  |_Мегафакультет .. (ID,..)
```