

Урок 4. Учим фреймворк работать с моделями сущностей

ОГЛАВЛЕНИЕ

[Что такое классы сущностей и для чего они нужны 3](#_Toc75779677)

[Как создать контроллеры для работы с моделями 7](#_Toc75779678)

[Добавляем привязку к маршрутам и Добавляем шаблоны 10](#_Toc75779679)

[запуск приложения 13](#_Toc75779680)

[Заключение 15](#_Toc75779681)

[КЛЮЧЕВЫЕ ВЫВОДЫ 15](#_Toc75779682)



# Что такое классы сущностей и для чего они нужны

Итак, мы завершили основную часть работы над Фреймворком и пришло время вплотную заняться веб-приложением, которое мы разрабатываем. Каждое веб-приложение посвящено какой-то предметной области и наша задача описать эту предметную область. В Django ORM и SQLAlchemy (ORM-библиотека, применяемая во Flask-проектах) мы используем специальные классы-модели, описывающие сущности нашей предметной области. Каждый такой класс является шаблоном таблицы БД и через него мы передаем данные в БД.

В нашем проекте также есть сущности. Опишем их с помощью ООП-подхода. Начнем с пользователей веб-приложения. Это сущности «Студент» и «Преподаватель». У них много общего, поэтому есть смысл создать некоторый базовый класс, от которого эти две сущности унаследуют свой интерфейс. Кроме того, нам пригодится еще один служебный класс, выполняющий роль фабрики пользователей и возвращающий объект нужной сущности, т.е. объект нужного класса – «Студента» или «Преподавателя».

**Листинг 1. step\_4/components/models.py**

|  |
| --- |
| *# Класс-Абстрактный пользователь* **class** User:  **pass** *# Класс-Преподаватель* **class** Teacher(User):  **pass** *# Класс-Студент* **class** Student(User):  **pass** *# Класс-Фабрика пользователей* **class** UserFactory:  types = {  **'student'**: Student,  **'teacher'**: Teacher  }   @classmethod  **def** create(cls, type\_):  **return** cls.types[type\_]() |

Продолжим описывать сущности, используемые в нашем проекте. Наше веб-приложение – это площадка для размещения курсов. Значит должны быть классы, описывающие такую сущность, как курс. У курса есть категория, в которую он входит, а также название самого курса. Сделаем класс «Курс» базовой сущностью, интерфейс которой унаследуют классы, представляющие конкретные виды курсов – «Интерактивный курс» и «Курс в записи». Кроме того, как и в предыдущем листинге, добавим класс, реализующий возможности фабрики курсов.

**Листинг 2. step\_4/components/models.py**

|  |
| --- |
| *# Класс-Курс* **class** Course:   **def** \_\_init\_\_(self, name, category):  self.name = name  self.category = category  *#self.category.courses.append(self)   # Класс-Интерактивный курс* **class** InteractiveCourse(Course):  **pass** *# Класс-Курс в записи* **class** RecordCourse(Course):  **pass** *# Класс-Фабрика курсов* **class** CourseFactory:  types = {  **'interactive'**: InteractiveCourse,  **'record'**: RecordCourse  }   @classmethod  **def** create(cls, type\_, name, category):  **return** cls.types[type\_](name, category) |

Т.к. каждый курс относится к какой-то категории, необходимо описать класс – «Категория курса».

**Листинг 3. step\_4/components/models.py**

|  |
| --- |
| *# Класс-Категория* **class** Category:   auto\_id = 0   **def** \_\_init\_\_(self, name, category):  self.id = Category.auto\_id  Category.auto\_id += 1  self.name = name  *self.category = category* self.courses = []   **def** course\_count(self):  result = len(self.courses)  *if self.category:  result += self.category.course\_count()* **return** result  **def** create(cls, type\_, name, category):  **return** cls.types[type\_](name, category) |

Теперь очень важный момент. Пока в нашем проекте нет базы данных, а также нет ORM. Создание новых пользователей, категорий, курсов потребует создания объектов соответствующих классов-сущностей. Но кто будет создавать эти объекты и куда их сохранять? Пока будем хранить объекты сущностей в оперативной памяти. Это значит, что если после создания нескольких сущностей мы остановим, а затем вновь запустим наш проект, то созданных объектов сущностей уже не будет. Пусть пока это будет некоторое ограничение нашего приложения.

Теперь касаемо того, кто будет обеспечивать создание объектов сущностей. Реализуем специальный класс-движок, который и будет отвечать за создание объектов сущностей и внесение их в списки.

**Листинг 4. step\_4/components/models.py**

|  |
| --- |
| *# Класс-Основной интерфейс проекта* **class** Engine:  **def** \_\_init\_\_(self):  self.teachers = []  self.students = []  self.courses = []  self.categories = []   @staticmethod  **def** create\_user(type\_):  **return** UserFactory.create(type\_)   @staticmethod  **def** create\_category(name, category=**None**):  **return** Category(name, category)   **def** find\_category\_by\_id(self, id):  **for** item **in** self.categories:  print(**'item'**, item.id)  **if** item.id == id:  **return** item  **raise** Exception(**f'Нет категории с id = {**id**}'**)   @staticmethod  **def** create\_course(type\_, name, category):  **return** CourseFactory.create(type\_, name, category)   **def** get\_course(self, name):  **for** item **in** self.courses:  **if** item.name == name:  **return** item  **return None** @staticmethod  **def** decode\_value(val):  val\_b = bytes(val.replace(**'%'**, **'='**).replace(**"+"**, **" "**), **'UTF-8'**)  val\_decode\_str = quopri.decodestring(val\_b)  **return** val\_decode\_str.decode(**'UTF-8'**) |

Интерфейс представленного класса включает методы для создания новых пользователей, категорий, курсов, а также методы для поиска и получения объекта категории по идентификатору и получения объекта курса по его названию. Также в этом классе разместим уже знакомый нам метод для корректного декодирования данных.

Итак, нашим модели готовы. Класс-движок, обеспечивающий работу с моделями, также написан. Вспомним, где мы в Django-проектах использовали классы-модели? Мы импортировали их в модуль с **views.py** и использовали в контроллерах-обработчиках запросов.

# Как создать контроллеры для работы с моделями

Пришло время создать нашему веб-приложению новые контроллеры, которые и должны работать с моделями. Значит нужно импортировать классы-модели в файл **views.py**. Но у нас есть центральное звено, через которое мы работаем с моделями. Это наш класс-движок Engine. Импортируем его и подготовим исходный код классов-контроллеров, которые будут основными обработчиками запросов в нашем веб-приложении. Это контроллеры, обеспечивающие вывод списка курсов, создание нового курса.

**Листинг 5. step\_4/views.py**

|  |
| --- |
| **from** components.models **import** Engine  site = Engine()  *# Класс-контроллер - Страница "Список курсов"* **class** CoursesList:  **def** \_\_call\_\_(self, request):   **try**:  category = site.find\_category\_by\_id(  int(request[**'request\_params'**][**'id'**]))  **return '200 OK'**, render(**'course\_list.html'**,  objects\_list=category.courses,  name=category.name,  id=category.id)  **except** KeyError:  **return '200 OK'**, **'No courses have been added yet'** *# Класс-контроллер - Страница "Создать курс"* **class** CreateCourse:  category\_id = -1   **def** \_\_call\_\_(self, request):  **if** request[**'method'**] == **'POST'**:   data = request[**'data'**]   name = data[**'name'**]  name = site.decode\_value(name)   category = **None  if** self.category\_id != -1:  category = site.find\_category\_by\_id(int(self.category\_id))   course = site.create\_course(**'record'**, name, category)  site.courses.append(course)   **return '200 OK'**, render(**'course\_list.html'**,  objects\_list=category.courses,  name=category.name,  id=category.id)   **else**:  **try**:  self.category\_id = int(request[**'request\_params'**][**'id'**])  category = site.find\_category\_by\_id(int(self.category\_id))   **return '200 OK'**, render(**'create\_course.html'**,  name=category.name,  id=category.id)  **except** KeyError:  **return '200 OK'**, **'No categories have been added yet'** |

Каждый курс относится к какой-либо категории. Значит создадим контроллеры, обеспечивающие получение списка категорий и создание новой категории.

**Листинг 6. step\_4/views.py**

|  |
| --- |
| *# Класс-контроллер - Страница "Создать категорию"* **class** CreateCategory:  **def** \_\_call\_\_(self, request):   **if** request[**'method'**] == **'POST'**:   print(request)  data = request[**'data'**]   name = data[**'name'**]  name = site.decode\_value(name)   category\_id = data.get(**'category\_id'**)   category = **None  if** category\_id:  category = site.find\_category\_by\_id(int(category\_id))   new\_category = site.create\_category(name, category)   site.categories.append(new\_category)   **return '200 OK'**, render(**'index.html'**,  objects\_list=site.categories)  **else**:  categories = site.categories  **return '200 OK'**, render(**'create\_category.html'**,  categories=categories)   *# Класс-контроллер - Страница "Список категорий"* **class** CategoryList:  **def** \_\_call\_\_(self, request):  **return '200 OK'**, render(**'category\_list.html'**,  objects\_list=site.categories) |

Еще одним штрихом на текущем этапе будет добавление контроллеров для страниц «Расписание» и «404».

**Листинг 7. step\_4/views.py**

|  |
| --- |
| **from** datetime **import** date  *# Класс-контроллер - Страница "Расписания"* **class** StudyPrograms:  **def** \_\_call\_\_(self, request):  **return '200 OK'**, render(**'study-programs.html'**, data=date.today())   *# Класс-контроллер - Страница 404* **class** NotFound404:  **def** \_\_call\_\_(self, request):  **return '404 WHAT'**, **'404 PAGE Not Found'** |

Контроллеры-обработчики готовы. Теперь необходимо определить, как пути они будут обрабатывать. Нужно перейти к файлу **urls.py** и выполнить нужные привязки.

# Добавляем привязку к маршрутам и Добавляем шаблоны

**Листинг 8. step\_4/urls.py**

|  |
| --- |
| **from** views **import** Index, About, StudyPrograms, CoursesList, \  CreateCourse, CreateCategory, CategoryList  *# Набор привязок: путь-контроллер* routes = {  **'/'**: Index(),  **'/about/'**: About(),  **'/study\_programs/'**: StudyPrograms(),  **'/courses-list/'**: CoursesList(),  **'/create-course/'**: CreateCourse(),  **'/create-category/'**: CreateCategory(),  **'/category-list/'**: CategoryList() } |

Контроллеры-обработчики в том числе обеспечивают визуализацию страниц отображения списка категорий, курсов, страниц добавления записей в эти списки. Необходимо подготовить сами страницы, т.е. соответствующие HTML-шаблоны.

**Листинг 9. step\_4/templates/study-programs.html**

|  |
| --- |
| {% extends "base.html" %} {% block title %}  Обучение программированию {% endblock %} {% block style %}  {% include "inc-style.html" %} {% endblock %} {% block menu %}  {% include "inc-menu.html" %} {% endblock %} {% block body %}  {% include "inc-index-body.html" %} {% endblock %} |

**Листинг 10. step\_4/templates/category\_list.html**

|  |
| --- |
| {% extends "base.html" %} {% block title %} Список курсов {% endblock %} {% block style %}  {% include "inc-style.html" %} {% endblock %} {% block menu %}  {% include "inc-menu.html" %} {% endblock %} {% block body %}  <**a href="/"**>назад</**a**>   <**h1**>Список категорий</**h1**>  <**ul**>   {% for item in objects\_list %}  <**li**>  {{item.name}} <**span**>Количество курсов: {{item.course\_count()}}</**span**>  </**li**>  {% endfor %}  </**ul**>   <**div**>  <**li**><**a href="/courses-list/"**>Список курсов</**a**></**li**>  <**li**><**a href="/create-course/"**>Создать курс</**a**></**li**>  </**div**> {% endblock %} |

**Листинг 11. step\_4/templates/create\_category.html**

|  |
| --- |
| {% extends "base.html" %} {% block title %} Создание курса {% endblock %} {% block style %}  {% include "inc-style.html" %} {% endblock %} {% block menu %}  {% include "inc-menu.html" %} {% endblock %} {% block body %} <**h1**>Создание категории</**h1**> <**form method="post"**>  <**input type="text" name="name" placeholder="Название"**>  <**button type="submit"**>Сохранить и вернуться на главную</**button**> </**form**> {% endblock %} |

**Листинг 12. step\_4/templates/course\_list.html**

|  |
| --- |
| {% extends "base.html" %} {% block title %} Список курсов {% endblock %} {% block style %}  {% include "inc-style.html" %} {% endblock %} {% block menu %}  {% include "inc-menu.html" %} {% endblock %} {% block body %}  <**div**>  <**a href="/"**>назад на главную</**a**>  </**div**>   <**br**>  <**br**>   <**div**>  <**a href="/create-course/?id={{id}}"**>Создать новый курс</**a**>  <**br**>  <**h1**>Список курсов для категории "{{name}}"</**h1**>  {% for item in objects\_list %}  <**li**>  {{item.name}}  </**li**>  {% endfor %}  </**div**> {% endblock %} |

**Листинг 13. step\_4/templates/create\_course.html**

|  |
| --- |
| {% extends "base.html" %} {% block title %} Создание курса {% endblock %} {% block style %}  {% include "inc-style.html" %} {% endblock %} {% block menu %}  {% include "inc-menu.html" %} {% endblock %} {% block body %}  <**div**>  <**a href="/courses-list/?id={{id}}"**>назад</**a**>  </**div**>   <**br**>  <**br**>   <**div**>  <**h1**>Создание курса для категории "{{name}}"</**h1**>  <**form method="post"**>  <**input type="text" name="name" placeholder="Название"**>  <**button type="submit"**>Сохранить и вернуться к списку курсов</**button**>  </**form**>  </**div**> {% endblock %} |

**Листинг 6. step\_3/simba\_framework/templates/** inc-menu**.html**

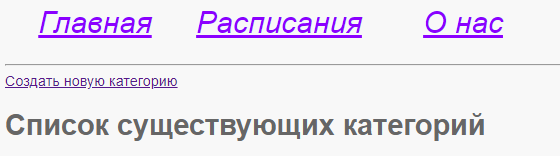
|  |
| --- |
| <**div class="menu"**>  <**li**><**a href="/"**>Главная</**a**></**li**>  <**li**><**a href="/about/"**>О нас</**a**></**li**> </**div**> <**hr**> |

# запуск приложения

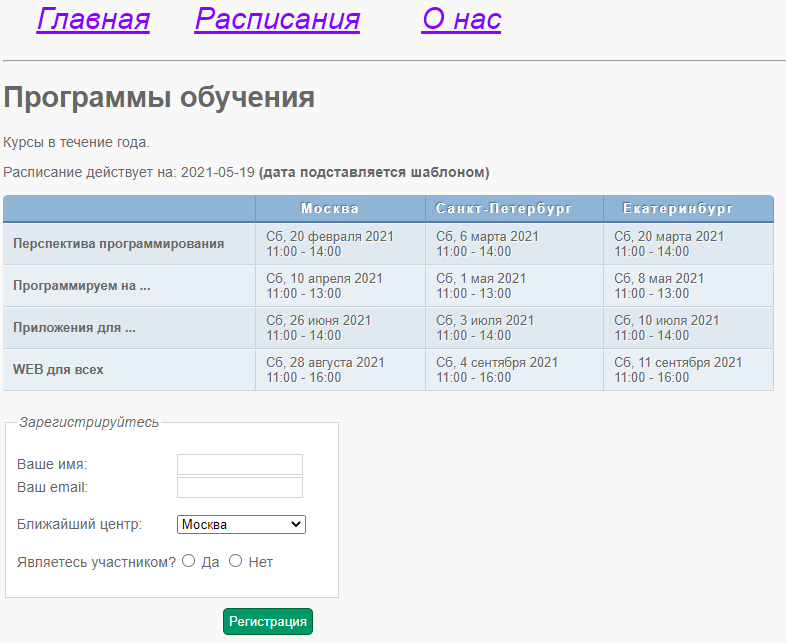
Проверим как работает наше приложение.

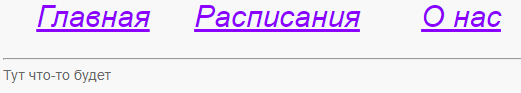
Перейдем на главную страницу веб-приложения:

<http://127.0.0.1:8080/>



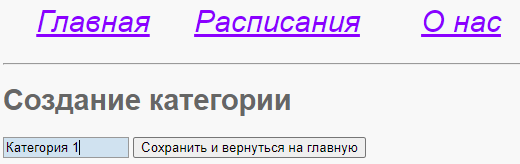
Проверим работу страниц «Расписания» и «О нас».

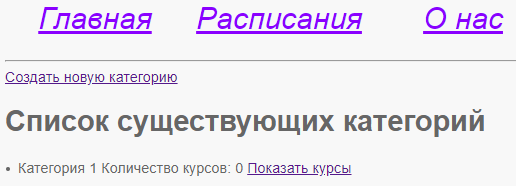




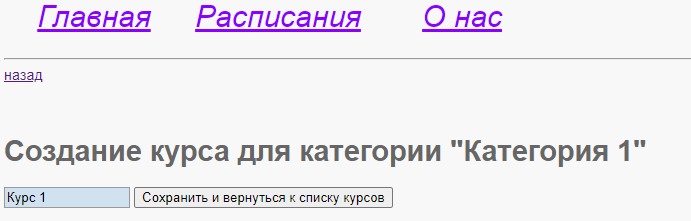
Текущая структура предложенного WSGI-фреймворка и создаваемого на его основе ве

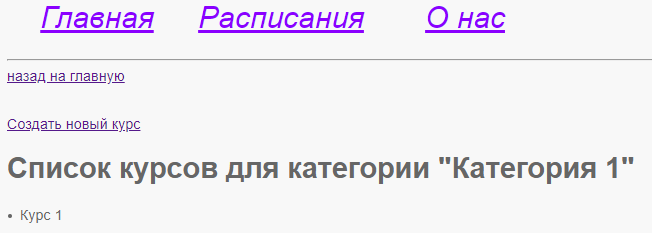
Текущая структура предложенного WSGI-фреймворка и создаваемого на его основщая структура предложенного WSGI-фреймворка и создаваемого на его основе веб-приложения приведена на рис.

Создадим новую категорию.



Создадим новый курс и привяжем его к созданной категорией.







# Заключение

Мы завершили еще один урок, в ходе которого с помощью ООП описали шаблоны сущностей, используемых в нашем проекте. Каждая сущность получила свой класс. Кроме того мо создали класс-движок, выполняющий роль центрального звена проекта, где происходит создание объектов сущностей и заполнение ими списков. Пока в нашем проекте нет базы данных, вся иноформация будет храниться в оперативной памяти, пока проект находится в стадии запуска. Кроме того, мы дополнили наш проект новыми классами-контроллерами, которые «привязали» к соответствующим URL-адресам. Эти классы-контроллеры выполянют обработку поступающих запросов и выполняют операции с объектами сущностей нашего проекта.

## КЛЮЧЕВЫЕ ВЫВОДЫ

* Каждая сущность нашего веб-приложения получила свой класс. Например, в Django такие классы мы называем моделями.
* Для работы с моделями подготовлен центральный класс – Движок, который в свою очередь используют класс-контроллеры для работы с сущностями.
* Мы подготовили ряд контроллеров (обработчиков запросов) и «связали» их с соответствующими URL-адресами.

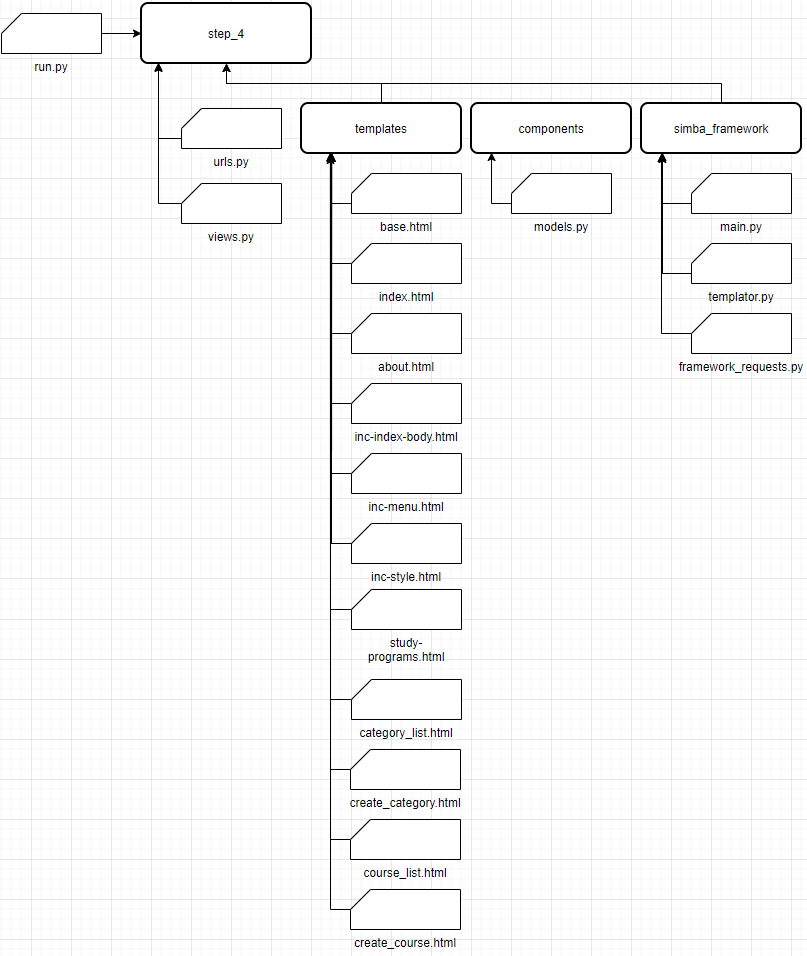
Текущая структура предложенного WSGI-фреймворка и создаваемого на его основе веб-приложения приведена на рис. 1.

Рис. 1. Структура проекта (фреймворк + веб-приложение)