



Фреймворк Django

разработка веб-приложений

For beginners



Урок 5. Маршрутизация

Принцип работы маршрутизатора

Процесс маршрутизации в Django представляет собой определение контроллера, который будет выполняться при поступлении клиентского запроса установленного формата. Подсистема Django, отвечающая за реализацию этого функционала и получила название маршрутизатора.

Перед тем, как передать управление контроллеру и выдать ответ в браузер, Django производит следующие действия:

1. Выделяет из запрашиваемого интернет-адреса путь, который передаёт маршрутизатору.
2. Из полученного URL удаляются: протокол, имя хоста / порта, список всех GET-параметров, в итоге получается только один путь. Например, мы сделали запрос по адресу <https://lms.com/list?page=10> и хотим получить список курсов на 10-ой странице. В результате обработки мы получим путь </list>, в свою очередь, протокол <https://>, имя хоста lms.com и GET-параметр [?page=10](https://lms.com/list?page=10) будут удалены.
3. Маршрутизатор производит поиск полученного пути в списках маршрутов всего проекта.
4. При совпадении маршрутизатор ищет в найденном маршруте контроллер и выполняет его. В том случае, если в маршруте имеется вложенный список маршрутов, то из пути удаляется префикс и инициируется разбор маршрутов из вложенного списка.
5. Если полученный путь не совпадает ни с одним шаблонным путём, то клиент получает в качестве ответа сгенерированную страницу с кодом ошибки [404](#) и сообщением «Страница не найдена».

Ошибка 404

Кажется что-то пошло не так! Страница, которую вы запрашиваете, не существует. Возможно она устарела, была удалена, или был введен неверный адрес в адресной строке.

[Перейти на главную](#)

Нужно отметить, что лучше создавать только уникальные маршруты, так как в случае установки нескольких одинаковых путей выполнится контроллер, принадлежащий самому первому указанному маршруту. Остальные будут проигнорированы.

Сопоставление URL-адресов с представлениями

Теперь необходимо разобраться как в Django решается проблема связи заданного URL с соответствующим ему представлением.

Знание того, как Django переходит от URL к представлению, важно для определения того, на что действительно способны представления.

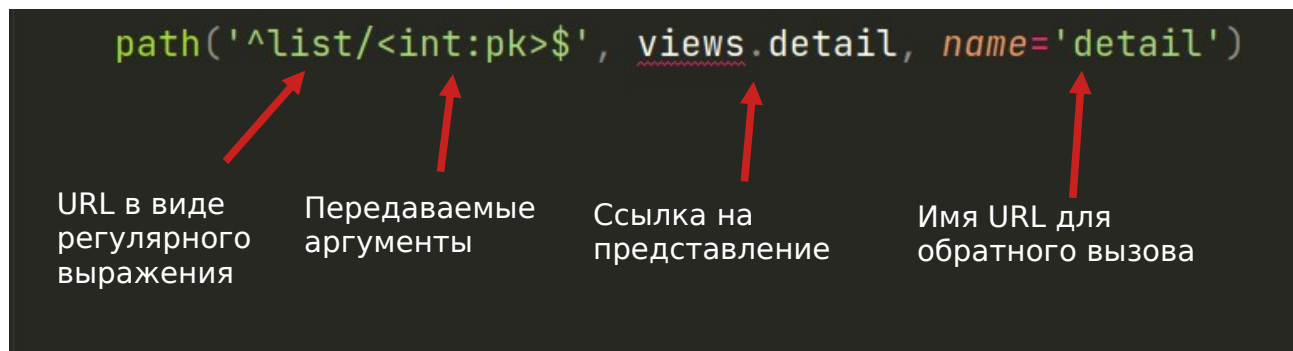
Сопоставление URL-адресов с представлениями — хорошо задокументированный процесс. Давайте же познакомимся с основными элементами, которые включает в себя любой маршрут.

Типичный шаблон URL состоит из нескольких отдельных элементов:

- **Регулярное выражение** для сопоставления с запрашиваемым входящим URL-адресом.
- **Ссылка на вызываемое представление**
- **Словарь аргументов**, который будет передаваться каждый раз при доступе к представлению.

- **Имя**, которое будет использоваться для ссылки на представление во время обратного поиска (значение термина «обратный поиск» мы разберём далее).

Приведём пример URL, содержащий все эти элементы:



Шаблон URL представляет собой регулярное выражение, которое может захватывать определенные части строки для дальнейшего использования.

Django использует это как стандартный способ извлечения аргументов из URL-адреса, чтобы их можно было передать в представление.

Есть два способа указания параметров, которые определяют, каким образом захваченные значения передаются в представление.

- Если параметры указаны без имён, они объединяются в кортеж **args**, который передаёт позиционные аргументы. Этот подход делает регулярное выражение немного меньше, но у него есть некоторые недостатки. Это не только делает регулярное выражение менее читабельным, но это также означает, что порядок аргументов в вашем представлении должен всегда совпадать с порядком групп в URL, потому что Django отправляет их как позиционные аргументы.
- Если параметрам даны имена, Django создаст словарь **kwargs**, сопоставляющий эти имена с извлеченными значениями из URL-адреса. Эта альтернатива помогает стимулировать более слабую связь между URL-адресами и представлениями за счёт передачи захваченных значений в представление в качестве аргументов ключевого слова. Обратите внимание, что

Django не позволяет использовать вместе именованные и неименованные группы в одном шаблоне URL.

Статические и динамические маршруты

Для более наглядного рассмотрения этих видов маршрутов приведём примеры их объявления.

```
urlpatterns = [  
    path('', views.index)  
]
```

В этом примере функция `path` связывает путь «» (являющийся пустой строкой) с представлением из файла `views.py` под именем `index`. Объявление пустой строки в виде маршрута означает, что для попадания на нужный нам сайт достаточно ввести только его домен. Любой же не пустой маршрут не совпадёт с таким шаблоном. Обычно данный путь соответствует главной странице сайта. Как можно уже догадаться такой тип маршрутов относится к статическим, поскольку он будет ожидать запроса ТОЛЬКО по этому пути.

Динамические же маршруты выглядят следующим образом:

```
urlpatterns = [  
    path('users/<int:pk>/favourites', views.index)  
]
```

Данный маршрут позволяет получить нам, в контексте нашего проекта, список понравившихся курсов. И он содержит динамическую часть `<int:pk>`, которая позволяет нам указать в пути id конкретного пользователя в таблице для получения списка курсов, которые он добавил в «Избранное». Более того, `int` говорит нам о том, что мы можем передать в данной части URL только целое число. Тем самым мы производим некоторую валидацию пути. Django предоставляет и другие преобразователи пути, о которых мы узнаем чуть

позже. В итоге с помощью объявления всего одного маршрута мы сможем получить информацию о списке понравившихся курсов у неограниченного числа пользователей. Данная динамическая часть легко передаётся в виде параметра функции-представления и в дальнейшем извлекается. Таким образом, динамические пути позволяют передавать в себе различные имена и идентификаторы, с помощью которых мы можем получить соответствующие данные.

Объявление маршрутов

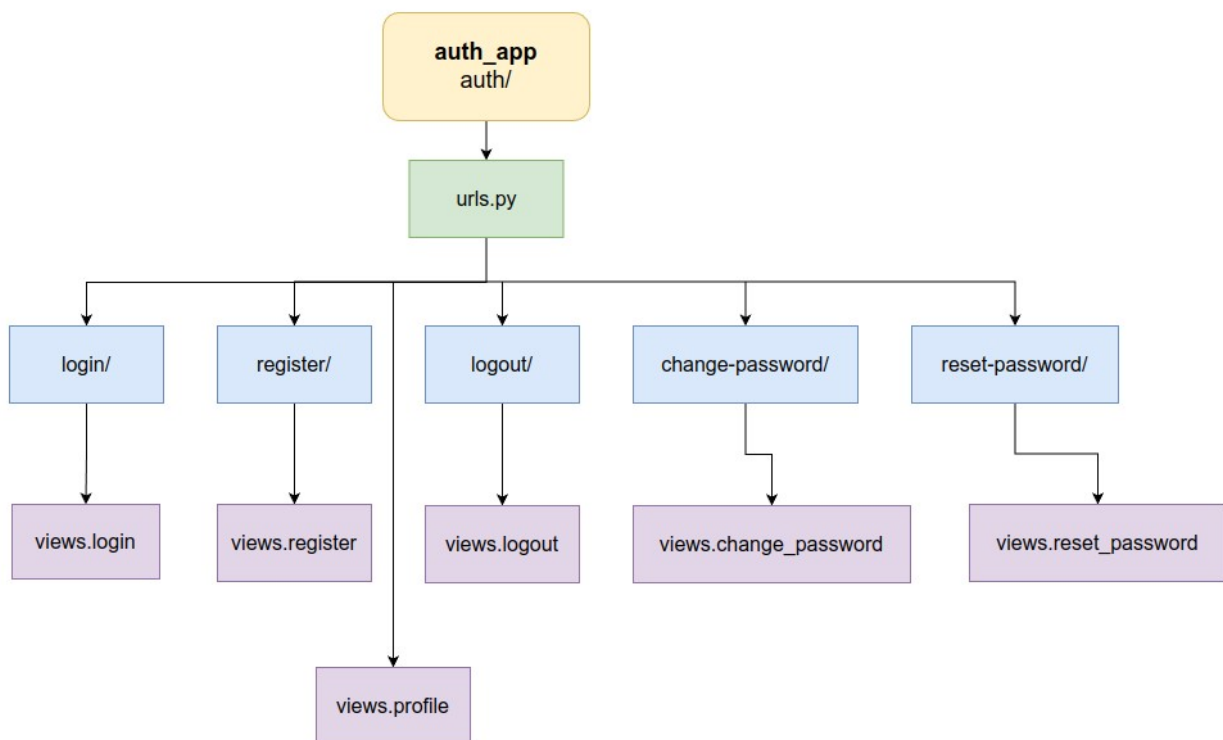
Пришло время перейти к описанию схемы маршрутов, которую будет иметь наша система онлайн-обучения.

Начнём с приложения аутентификации.

Приложение `auth_app` будет иметь следующий функционал:

- регистрация пользователей
- авторизация пользователей (вход на сайт, когда пользователь уже создал аккаунт)
- выход пользователя с сайта
- просмотр своего профиля
- изменение / сброс пароля

Полная схема будет выглядеть так:



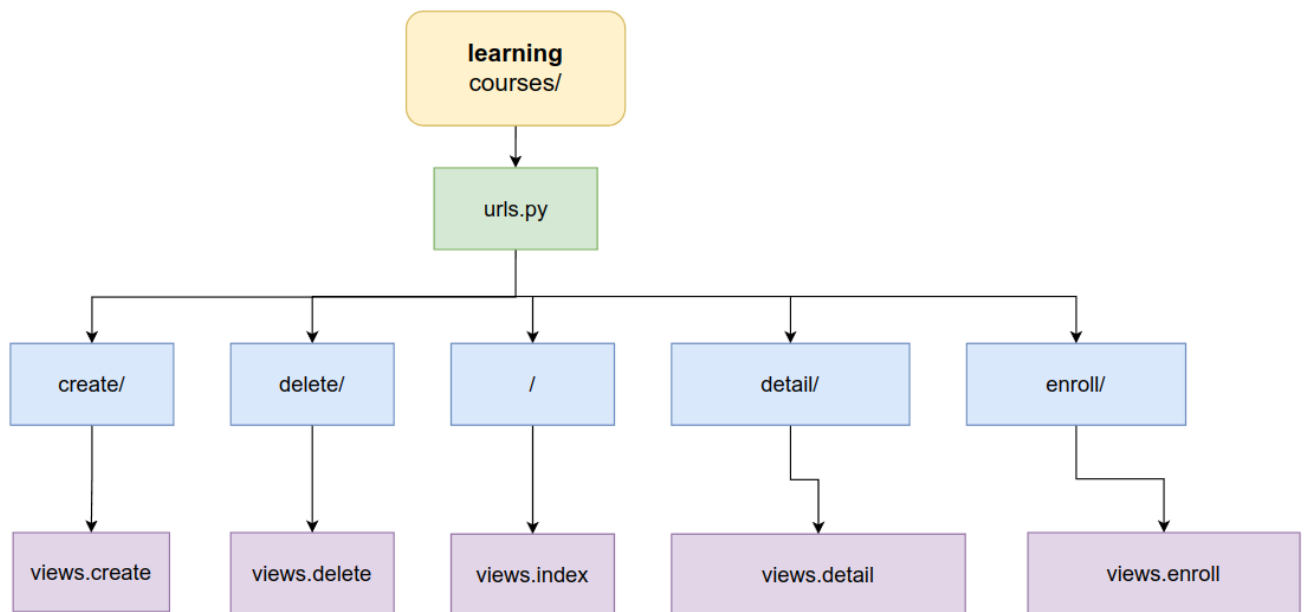
На схеме видно, что каждый шаблонный путь имеет соответствующее имя связываемого представления в файле **views.py**.

Теперь определим функционал для приложения **learning**.

Он будет иметь следующие опции:

- просмотр списка доступных курсов
- просмотр детальной информации по конкретному курсу
- возможность пользователя записаться на курс
- возможность создать свой курс
- возможность удалить свой курс

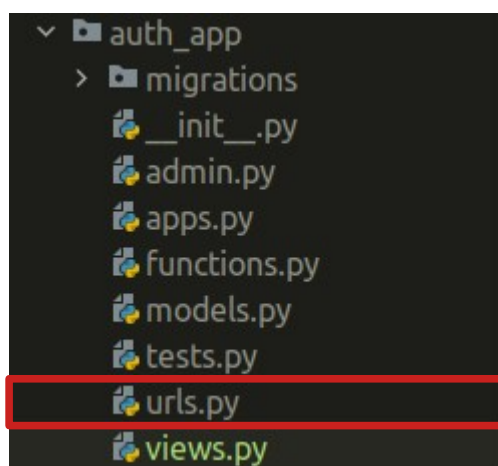
Полная структура связи маршрутов с представлениями представлена на картинке ниже:



Описание данной схемы аналогично описанию для приложения `auth_app`. Стоит добавить только то, что путь, обозначенный `</>`, будет главной страницей нашего сайта, которую будет получать пользователь при переходе по пути `courses/`, т. е. при попадании в приложение `learning`.

Конвертируем получившуюся схему в код. Начнём всё так же с приложения `auth_app`.

При создании приложения командой `startapp` по умолчанию не создаётся файл `urls.py`, поэтому создадим его.



Далее реализуем схему связи маршрутов с представлениями:


```
1 from django.urls import path
2 from .views import *
3
4
5 urlpatterns = [
6     path('login/', login),
7     path('register/', register),
8     path('logout/', logout),
9     path('change-password/', change_password),
10    path('reset-password/', reset_password),
11]
```

Список маршрутов представляет собой обычный список Python, имеющий имя `urlpatterns`. Этот список должен обязательно иметь такое имя, так как маршрутизатор будет производить поиск шаблонных путей именно там. Каждый элемент этого списка является результатом выполнения функции `path` из модуля `django.urls`, который мы и импортируем.

Эта функция имеет следующие параметры:

1. Шаблонный путь в виде строки, в конце обязательно должен стоять символ прямого слэша «/»
2. Ссылка на функцию-контроллер ИЛИ вложенный список маршрутов

Первые 2 параметра являются обязательным для заполнения! Остальные же — нет.

3. Дополнительные параметры
4. Имя маршрута, являющееся параметром `name` функции `path`

Все маршруты, как мы можем заметить, будут статическими, т. е. предполагают получение конкретного функционала ТОЛЬКО по заданному URL.

Мы видим, что IDE нам подсказывает, что указанные контроллеры не существуют / не импортированы, поэтому откроем файл `views.py` приложения `auth_app` и создадим нужные обработчики:

```
1  from django.http import HttpResponse
2  from django.shortcuts import render
3
4
5  def login(request):
6      return HttpResponse('Эта страница для входа пользователя на сайт')
7
8
9  def register(request):
10     return HttpResponse('Эта страница для регистрации пользователя')
11
12
13  def logout(request):
14     return HttpResponse('Это представление выполняет выход и редирект на страницу входа')
15
16
17  def change_password(request):
18     return HttpResponse('Этот обработчик меняет пароль пользователя')
19
20
21  def reset_password(request):
22     return HttpResponse('В этом обработчике реализована логика сброса пароля пользователя')
```

В качестве первого параметра функции-обработчика всегда следует обязательно указывать параметр `request`, представленный объектом `HttpRequest`. Этот параметр представляет собой объект текущего запроса, с помощью которого мы можем получить данные о запросе, такие как: длина тела запроса, клиентский IP-адрес, клиентское имя хоста, метод запроса: GET / POST и многое другое.

В данном уроке перед нами стоит задача понять, как работает подсистема маршрутизатора и научиться связывать пути с представлениями, поэтому ограничимся возвратом из функции-представления стандартного HTTP-ответа, представленного объектом `HttpResponse` из модуля `django.http`, который отправляется клиенту в браузер. Содержимое этого ответа является обычным текстовым сообщением, в данном случае опишем в нём, что будет происходить при выполнении данного обработчика.

Для успешной связи маршрутов с представлениями также необходимо подключить созданный список маршрутов в файле `urls.py` директории настроек проекта:

```
16 from django.contrib import admin
17 from django.urls import path, include
18
19 urlpatterns = [
20     path('admin/', admin.site.urls),
21     path('auth/', include('auth_app.urls')),
22 ]
```

Этого можно добиться, импортировав функцию `include` всё из того же модуля `django.urls`.

Вместо того, чтобы представлять все шаблоны URL в одном файле, функция `include` позволяет разделить их между несколькими приложениями. Она принимает один аргумент: путь импорта, по которому можно найти другой модуль конфигурации URL.

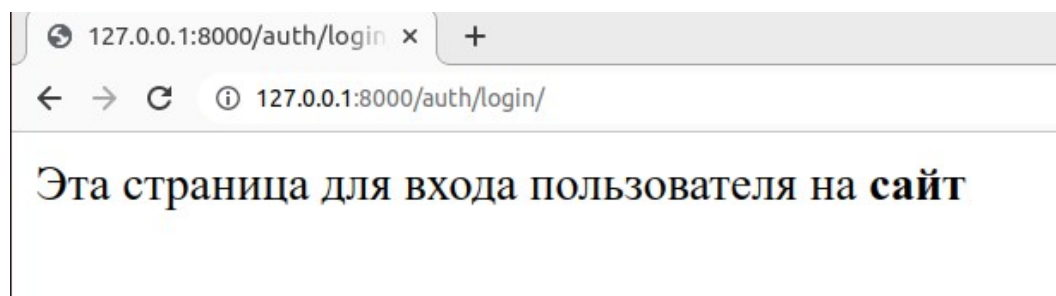
Первый параметр функции `path` указывает маршрут, ведущий на вложенные списки маршрутов.

Совершим перезапуск сервера и перейдём по первому пути приложения `auth_app` в браузере. Мы увидим, что ответ был успешно возвращён.

Также можно обернуть текст в какие-нибудь HTML-теги для выделения части контента.

```
5 def login(request):
6     return HttpResponse('Эта страница для входа пользователя на <b>сайт</b>')
```

В результате обёрнутый текст станет жирным:



Проверяя следующие по списку интернет-адреса мы будем видеть ответ, специфичный для каждого обработчика. Если же мы ошибёмся в URL хотя бы на 1 символ, то маршрутизатор выдаст вполне ожидаемую ошибку с кодом 404. При этом он покажет все зарегистрированные адреса, чтобы нам стало понятно, что введённого адреса не существует.



Теперь объявим маршруты для приложения **learning**. Для этого тем же образом создадим файл **urls.py**, привяжем маршруты к обработчикам, создадим обработчики и зарегистрируем список маршрутов в файле **urls.py** в директории настроек проекта.

Файл `urls.py` приложения `learning`:

```
1 from django.urls import path
2 from .views import *
3
4
5 urlpatterns = [
6     path('', index, name='index'),
7     path('create/', create, name='create'),
8     path('delete/', delete, name='delete'),
9     path('detail/', detail, name='detail'),
10    path('enroll/', enroll, name='enroll'),
11 ]
```

В этом приложении мы использовали параметр `name` при объявлении маршрута. Данный параметр позволяет применять обратный поиск адресов. Это может говорить нам о том, что по заданному имени маршрута автоматически будет формироваться соответствующий ему готовый адрес. Данная возможность может использоваться как в шаблонах, так и в контроллерах. Она позволяет при замене определённого маршрута не потерять связь с соответствующим контроллером.

Например, это может быть полезно при наличии в коде проекта большого числа перенаправлений на заданные маршруты. Т.е. в данном случае нам нужно будет изменить маршрут только в файле `urls.py`, остальное же за нас сделает возможность обратного разрешения адресов.

Файл views.py:

```
1 from django.http import HttpResponse
2 from django.shortcuts import render
3
4
5 def index(request):
6     return HttpResponse('Здесь будет список всех доступных курсов')
7
8
9 def create(request):
10    return HttpResponse('Здесь будет форма для создания собственного курса')
11
12
13 def delete(request):
14    return HttpResponse('Здесь будет производиться удаление указанного курса')
15
16
17 def detail(request):
18    return HttpResponse('Здесь мы узнаем детальную информацию о курсе')
19
20
21 def enroll(request):
22    return HttpResponse('Здесь мы сможем записаться на выбранный курс')
```

Файл urls.py:

```
19 urlpatterns = [
20     path('admin/', admin.site.urls),
21     path('auth/', include('auth_app.urls')),
22     path('courses/', include('learning.urls')),
23 ]
```

Здесь всё аналогично приложению `auth_app`.

Передача данных в контроллеры

Для передачи URL-параметра в контроллер-функцию необходимо в шаблоне пути заключить передаваемый параметр в угловые скобки «<>». Далее нужно указать тип параметра, например, `int`, и наконец в конце дать имя этому параметру. При этом тип параметра и имя разделяются двоеточием «:». Имя,

данное в URL, и станет именем аргумента при объявлении функции и оно обязательно к заполнению.

В итоге получим следующий пример: «`detail/<int:course_id>`».

Отредактируем шаблоны URL в приложении `learning` с учётом вышесказанного.

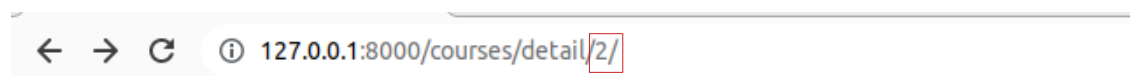
Для выполнения процесса удаления, записи на курс и просмотра детальной информации о нём нам нужно указать в URL, над каким курсом мы хотим произвести действие. В качестве параметра будем использовать идентификатор, представляющий автоинкрементное поле таблицы `learning_course`. Он относится к типу `int`, поэтому его и укажем. И после двоеточия назовём передаваемый параметр именем `course_id`. Продублируем объявление динамической части в остальных URL.

```
8      path('delete/<int:course_id>', delete, name='delete'),
9      path('detail/<int:course_id>', detail, name='detail'),
10     path('enroll/<int:course_id>', enroll, name='enroll'),
```

Перейдём в файл `views.py` и добавим к аргументам функций-обработчиков соответствующие переменные, и немного скорректируем содержимое ответа.

```
13     def delete(request, course_id):
14         return HttpResponse(f'Здесь будет производиться удаление {course_id} курса')
15
16
17     def detail(request, course_id):
18         return HttpResponse(f'Здесь мы узнаем детальную информацию о {course_id} курсе')
19
20
21     def enroll(request, course_id):
22         return HttpResponse(f'Здесь мы сможем записаться на {course_id} курс')
```

Перейдём в браузер и проверим корректность выполнения запросов.



Здесь мы узнаем детальную информацию о 2 курсе

Запросы выполнены успешно. Если ввести в части, где нужно передать параметр, значение, отличное от типа `int`, то появится всё та же страница с ошибкой 404.

Помимо параметра URL типа `int` существуют и другие типы, которые мы можем передать:

- `str` — любая непустая строка, кроме символа прямого слэша `</>`.
- `path` — то же самое, что и `str`, но он включает в себя символ прямого слэша `</>`.
- `uuid` — правильно отформатированный универсальный уникальный идентификатор.
- `slug` — слог в виде строки. Используется в случаях, когда нужно в качестве параметра URL передать, например, название статьи, написанное транслитом. При этом слова разделяются дефисами или символами подчёркивания и содержат только строчные символы. Например, `web-development-course_2`.

Регулярные выражения в URL

Указание шаблонных путей в формате регулярных выражений позволяет нам самим определить, каким образом наше приложение будет производить обработку входящего запроса.

Функция `re_path` из модуля `django.urls` позволяет указывать в шаблонных путях регулярные выражения. Только этим она и отличается от функции `path`. В наборе передаваемых аргументов она полностью идентична ей. Формат ввода аналогичен использованию регулярных выражений с помощью модуля `re`.

В качестве наглядного примера преобразуем путь `detail/` приложения `learning` в регулярное выражение. Импортируем функцию `re_path` и немного изменим ранее объявленный маршрут.

```
1 from django.urls import path, re_path
```


9

```
re_path('^detail/(?P<course_id>[1-9]*)/$', detail, name='detail'),
```

Символ «`^`» означает, что объявленный шаблон должен присутствовать в начале пути. Символ «`$`» говорит нам о том, что полученный путь должен полностью совпадать с шаблонным. В куске пути, заключенном в скобки, мы объявляем именованную группу под именем `course_id` и указываем, что клиент сможет получить только первые 9 курсов.

A screenshot of a web browser's address bar. It shows navigation icons (back, forward, refresh) on the left, followed by the IP address and port '127.0.0.1:8000', and the path '/courses/detail/7/'.

Здесь мы узнаем детальную информацию о 7 курсе

Как видим обработка запроса прошла успешно, но если мы попытаемся ввести число, выходящее за границы указанного предела, то мы получим ошибку.

Page not found (404)

Request Method: GET

Request URL: http://127.0.0.1:8000/courses/detail/10/

🚩 Домашнее задание

- ✓ Определить схему запросов, которую будет иметь ваше приложение
- ✓ Реализовать полученную схему в виде объявленных путей и связанных с ними функций-представлений
- ✓ Реализовать хотя бы один шаблонный путь в виде регулярного выражения