Сириус
IT-Колледж

FastAPI. Шаблоны Jinja2

HTML. CSS. Jinja2

Сириус IT-Колледж

HTML

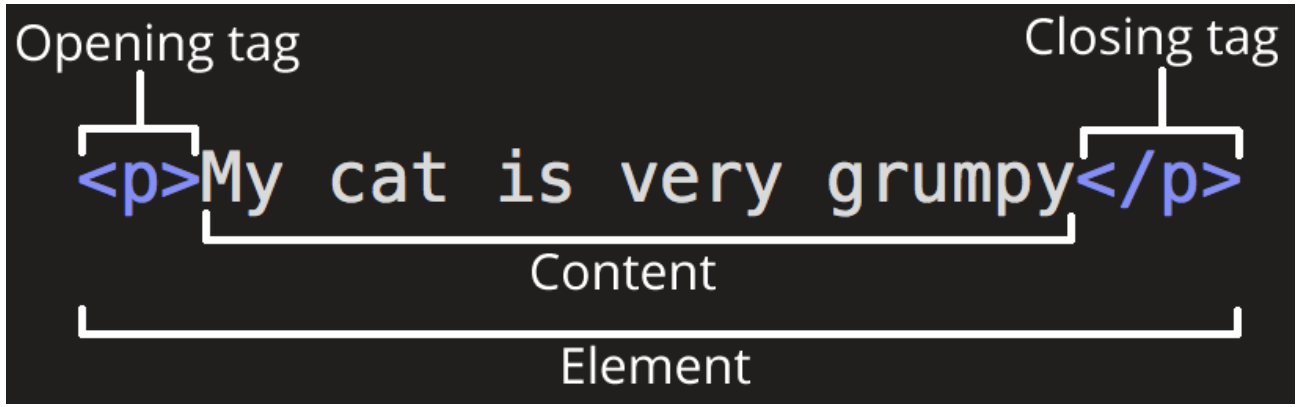
Hypertext Markup Language (HTML) - язык разметки для создания веб-страниц. Предназначен для создания документов разметки для просмотра на компьютере или для передачи по сети.

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Page</title>
    </head>
    <body>
        <h1>Hello, World!</h1>
    </body>
</html>
```

Сириус
IT-Колледж

HTML. Структура элемента

1. **Открывающий тег (Opening tag)**: Состоит из имени элемента (в данном случае, "p"), заключённого в открывающие и закрывающие угловые скобки. Открывающий тег указывает, где элемент начинается или начинает действовать, в данном случае — где начинается абзац.
2. **Закрывающий тег (Closing tag)**: Это то же самое, что и открывающий тег, за исключением того, что он включает в себя косую черту перед именем элемента. Закрывающий элемент указывает, где элемент заканчивается.
3. **Контент (Content)**
4. **Элемент(Element)**: Открывающий тег, закрывающий тег и контент вместе составляют элемент.



Сириус
IT-Колледж

HTML. Атрибуты элемента

Атрибуты содержат дополнительную информацию об элементе, которую вы не хотите показывать в фактическом контенте. В данном случае, `class` это имя атрибута, а `editor-note` это значение атрибута.

Атрибут всегда должен иметь:

1. Пробел между ним и именем элемента (или предыдущим атрибутом, если элемент уже имеет один или несколько атрибутов).
2. Имя атрибута, за которым следует знак равенства.
3. Значение атрибута, заключённое с двух сторон в кавычки.

Сириус IT-Колледж

HTML. Вложенные элементы

Вы также можете располагать элементы внутри других элементов — это называется вложением. Если мы хотим заявить, что наша кошка очень раздражена, мы можем заключить слово "очень" в элемент `<strong>`, который указывает, что слово должно быть сильно акцентированно:

```
<p>Моя кошка <strong>очень</strong> раздражена.</p>
```

Моя кошка **очень** раздражена.

Вы, однако, должны убедиться, что ваши элементы правильно вложены: в примере выше мы открыли первым элемент `<p>`, затем элемент `<strong>`, потом мы должны закрыть сначала элемент `<strong>`, затем `<p>`.

Сириус IT-Колледж

HTML. Пустые элементы

Некоторые элементы не имеют контента, и называются пустыми элементами. Возьмём элемент `<img>`:

```
<img src="images/firefox-icon.png" alt="Моё тестовое изображение" />
```

Он содержит два атрибута, но не имеет закрывающего тега `</img>`, и никакого внутреннего контента. Это потому, что элемент изображения не оборачивает контент для влияния на него. Его целью является вставка изображения в HTML страницу в нужном месте.

Сириус IT-Колледж

HTML. Структура документа

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Моя тестовая страница</title>
  </head>
  <body>
    <img src="images/firefox-icon.png" alt="Моё тестовое изображение" />
  </body>
</html>
```

- `<!DOCTYPE html>` — доктайп. На самом деле просто исторический артефакт, который должен быть включён для того, что бы все работало правильно.
- `<html></html>` — элемент `<html>`. Этот элемент оборачивает весь контент на всей странице, и иногда известен как корневой элемент.

Сириус
IT-Колледж

HTML. Структура документа

- `<head></head>` — элемент `<head>`. Этот элемент выступает в качестве контейнера для всего, что вы пожелаете включить на HTML страницу, но не являющегося контентом, который вы показываете пользователям вашей страницы. К ним относятся такие вещи, как ключевые слова и описание страницы, которые будут появляться в результатах поиска, CSS стили нашего контента, кодировка и многое другое.
- `<body></body>` — элемент `<body>`. В нем содержится весь контент, который вы хотите показывать пользователям, когда они посещают вашу страницу, будь то текст, изображения, видео, игры, проигрываемые аудиодорожки или что-то ещё.
- `<meta charset="utf-8">` — этот элемент устанавливает UTF-8 кодировку вашего документа.
- `<title></title>` — элемент `<title>`. Этот элемент устанавливает заголовок для вашей страницы, который является названием, появляющимся на вкладке браузера загружаемой страницы, и используется для описания страницы, когда вы добавляете её в закладки/избранное.

Сириус IT-Колледж

HTML. Изображения

```
<img src="images/firefox-icon.png" alt="Моё тестовое изображение" />
```

Как было сказано раньше, код встраивает изображение на нашу страницу в нужном месте. Это делается с помощью атрибута `src` (source, источник), который содержит путь к нашему файлу изображения.

Мы также включили атрибут `alt` (alternative, альтернатива). В этом атрибуте, вы указываете поясняющий текст для пользователей, которые не могут увидеть изображение по техническим или иным причинам.

Сириус
IT-Колледж

HTML. Разметка текста. Заголовки

Элементы заголовка позволяют вам указывать определённые части вашего контента в качестве заголовков или подзаголовков. Точно так же, как книга имеет название, названия глав и подзаголовков, HTML документ может содержать то же самое. HTML включает шесть уровней заголовков `<h1>` – `<h6>`, хотя обычно вы будете использовать не более 3-4 :

```
<h1>Мой главный заголовок</h1>
<h2>Мой заголовок верхнего уровня</h2>
<h3>Мой подзаголовок</h3>
<h4>Мой под-подзаголовок</h4>
```

Сириус
IT-Колледж

HTML. Разметка текста. Абзацы

Как было сказано раньше, элемент `<p>` предназначен для абзацев текста; вы будете использовать их регулярно при разметке текстового контента:

```
<p>Это одиночный абзац</p>
```

Сириус IT-Колледж

HTML. Разметка текста. Списки

Наиболее распространёнными типами списков являются нумерованные и ненумерованные списки:

1. **Ненумерованные списки** - это списки, где порядок пунктов не имеет значения. Они оборачиваются в элемент `<ul>`.
2. **Нумерованные списки** - это списки, где порядок пунктов имеет значение. Они оборачиваются в элемент `<ol>`.

Каждый пункт внутри списков располагается внутри элемента `<li>` (list item, элемент списка).

```
<p>Mozilla, мы являемся мировым сообществом</p>

<ul>
  <li>технологов</li>
  <li>мыслителей</li>
  <li>строителей</li>
</ul>

<p>работающих вместе ...</p>
```

Сириус
IT-Колледж

HTML. Ссылки

Чтобы добавить ссылку, нам нужно использовать простой элемент — `<a>` — а это сокращение от "anchor" ("якорь").

```
<a href="https://www.mozilla.org/ru/about/manifesto/details/">Манифест Mozilla</a>
```

Сириус
IT-Колледж

HTML. Базовый блочный элемент `<div>`

Элемент разделения контента HTML (`<div>`) является универсальным контейнером для потокового контента. Он не влияет на контент или макет до тех пор, пока не будет стилизован с помощью CSS.

Являясь "чистым" контейнером, элемент `<div>`, по существу, не представляет ничего. Между тем, он используется для группировки контента, что позволяет легко его стилизовать, используя атрибуты class или id, помечать раздел документа, написанный на разных языках (используя атрибут lang), и так далее.

```
<div>
  <p>Любой тип контента. Например, &lt;p&gt;, &lt;table&gt;. Все что угодно!</p>
</div>
```

CSS - Cascading Style Sheets (Каскадные таблицы стилей)

CSS — это язык на основе правил: вы задаёте правила, определяющие группы стилей, которые должны применяться к определённым элементам или группам элементов на вашей веб-странице. Например:

```
h1 {
  color: red;
  font-size: 5em;
}
```

Правило открывается с помощью **селектора**. Этот селектор выбирает HTML-элемент, который мы собираемся стилизовать. В этом случае мы используем заголовки первого уровня — (`<h1>`).

Затем у нас есть набор фигурных скобок `{ }`. Внутри них будет один или несколько объявлений, которые принимают форму пары свойства и его значения. Каждая пара указывает свойство элемента(-ов), который(-е) мы выбираем, а затем значение, которое мы хотели бы присвоить свойству.

Перед двоеточием у нас есть свойство, а после двоеточия — значение. CSS-свойства имеют разные допустимые значения в зависимости от того, какое свойство указывается.

Таблица стилей CSS будет содержать много таких правил, написанных одно за другим.

```
h1 {
  color: red;
  font-size: 5em;
}

p {
  color: black;
}
```

Чтобы связать `styles.css` с `index.html`, добавьте следующую строку где-то внутри `<head>` HTML документа:

```
<link rel="stylesheet" href="styles.css" />
```

Элемент `<link>` сообщает браузеру, что у нас есть таблица стилей, используя атрибут `rel`, и местоположение этой таблицы стилей в качестве значения атрибута `href`.

Сириус IT-Колледж

CSS. Селектор

Чтобы нацелиться на все абзацы в документе, вы должны использовать селектор `p`. Чтобы сделать все абзацы зелёными, вы должны использовать:

```
p {
  color: green;
}
```

Вы можете выбрать несколько селекторов одновременно, разделив их запятыми.

```
p, li {
  color: green;
}
```

Сириус IT-Колледж

CSS. Добавление класса

Пока у нас есть стилизованные элементы, основанные на их именах HTML-элементов. Это работает до тех пор, пока вы хотите, чтобы все элементы этого типа в вашем документе выглядели одинаково. В большинстве случаев это не так, и вам нужно будет найти способ выбрать подмножество элементов, не меняя остальные. Самый распространённый способ сделать это — добавить класс к вашему HTML-элементу и нацелиться на этот класс.

```
<ul>
  <li>Элемент один</li>
  <li class="special">Элемент два</li>
  <li>Элемент <em>три</em></li>
</ul>
```

Сириус IT-Колледж

CSS. Добавление класса

В вашем CSS вы можете выбрать класс `special` к любому элементу на странице, чтобы он выглядел так же, как и этот элемент списка.

```
.special {
  color: orange;
  font-weight: bold;
}
```

Иногда вы увидите правила с селектором, который перечисляет селектор HTML-элемента вместе с классом:

```
li.special {
  color: orange;
  font-weight: bold;
}
```

Этот синтаксис означает **«предназначаться для любого элемента `li`, который имеет класс `special»`**.

Сириус IT-Колледж

CSS. Стилизация элементов на основе их расположения в документе

Например, чтобы выбрать только `<em>` который вложен в элемент `<li>`, можно использовать селектор под названием **descendant combinator (комбинатор-потомок)**, который просто принимает форму пробела между двумя другими селекторами.

```
li em {
  color: rebeccapurple;
}
```

Этот селектор выберет любой элемент `<em>`, который находится внутри (потомка) `<li>`.

Ещё можно попробовать стилизовать абзац, когда он идёт сразу после заголовка на том же уровне иерархии в HTML. Для этого поместите `+` **(соседний братский комбинатор)** между селекторами.

```
h1 + p {
  font-size: 200%;
}
```

Сириус IT-Колледж

CSS. Стилизация элементов на основе состояния

Когда мы создаём ссылку, мы должны нацелить элемент `<a>` (якорь). Он имеет различные состояния в зависимости от того, посещается ли он, посещается, находится над ним, фокусируется с помощью клавиатуры или в процессе нажатия (активации). Вы можете использовать CSS для нацеливания на эти разные состояния — CSS-код ниже отображает ссылки розового цвета и посещённые ссылки зелёного цвета.

```
a:link {
  color: pink;
}

a:visited {
  color: green;
}
```

Сириус
IT-Колледж

CSS. Стилизация элементов на основе состояния

Вы можете изменить внешний вид ссылки, когда пользователь наводит на неё курсор, например, удалив подчёркивание, что достигается с помощью следующего правила:

```
a:hover {
  text-decoration: none;
}
```

Сириус IT-Колледж

CSS. Сочетание селекторов и комбинаторов

Стоит отметить, что вы можете комбинировать несколько селекторов и комбинаторов вместе. Вот пример:

```
/* выбирает любой <span> внутри <p>, который находится внутри <article>  */
article p span { ... }
/* выбирает любой <p>, который идёт сразу после <ul>, который идёт сразу после <h1>  */
h1 + ul + p { ... }
```

Вы также можете комбинировать несколько типов вместе.

```
body h1 + p .special {
  color: yellow;
  background-color: black;
  padding: 5px;
}
```

Это будет стиль любого элемента с классом `special`, который находится внутри `<p>`, который приходит сразу после `<h1>`, который находится внутри `<body>`.

Сириус IT-Колледж

Jinja2

Модуль **Jinja2** - это современный и удобный язык шаблонов для Python, созданный по образцу шаблонов Django. Он быстр, т.к. компилируется в код Python, широко используется и безопасен благодаря дополнительной среде выполнения изолированных шаблонов.

Преимущества языка шаблонов Jinja2:

- Автоматическая система экранирования HTML для предотвращения XSS.
- Наследование шаблонов, поддержка макросов.
- Шаблоны компилируются до оптимального кода Python (можно отключить при отладке).
- При отладке, номера строк исключений точно указывают на неправильную строку в шаблоне.
- Настраиваемый синтаксис, много встроенный фильтров.
- Поддержка использования методов стандартных типов Python в шаблонах.
- Возможность вызова функций Python в шаблонах.

Сириус
IT-Колледж

Jinja2. Шаблоны

Шаблон Jinja2 - это просто текстовый файл. Модуль Jinja2 может генерировать любой текстовый формат (HTML, XML, CSV, LaTeX и т. д.). Шаблон Jinja не обязательно должен иметь конкретное расширение: вполне подойдет .html, .xml или любое другое расширение.

Шаблон содержит переменные и/или выражения, которые заменяются значениями при визуализации шаблона, также применяются теги, управляющие логикой шаблона. Синтаксис шаблона во многом вдохновлен Django и Python.

Jinja2. Шаблоны

Ниже приведен минимальный шаблон, который иллюстрирует некоторые основы использования конфигурации Jinja по умолчанию.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>My Webpage</title>
</head>
<body>
    <ul id="navigation">
    {% for item in navigation %}
        <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
    {% endfor %}
    </ul>

    <h1>My Webpage</h1>
    {{ a_variable }}

    {# a comment #}
</body>
</html>
```

Сириус IT-Колледж

Jinja2. Шаблоны

В примере показаны параметры конфигурации по умолчанию. Разработчик приложения может изменить конфигурацию синтаксиса с `{% foo %}` на `<% foo %>` или что-то подобное.

Есть несколько видов разделителей. Разделители Jinja по умолчанию настроены следующим образом:

- `{% ... %}` используется операторами шаблонов, такими как for/in, if/else, set, extend и т.д.
- `{{ ... }}` используется для написания выражений или переменными для печати на выходе шаблона.
- `{# ... #}` используется для написания комментариев, не включенных в выходные данные шаблона
- `# ... ##` используется для строчных операторов.

Сириус
IT-Колледж

Jinja2. Расширения

Как сказано выше, файл с любым расширением может быть загружен в качестве шаблона. Добавление расширения `.jinja`, такого как `user.html.jinja`, может облегчить работу некоторых IDE или плагинов, используемых редактором кода. Автоэкранирование HTML, может применяться на основе расширения файла, в этом случае необходимо будет учитывать дополнительный суффикс.

Еще одна хорошая эвристика для идентификации шаблонов заключается в том, что по умолчанию они находятся в папке templates, независимо от расширения. Это общий макет для проектов.

Сириус
IT-Колледж

Jinja2. Переменные и выражения в шаблонах Jinja.

Переменные шаблона определяются контекстным словарем, переданным в шаблон при помощи метода `Template.render()`.

Переменные могут иметь атрибуты или элементы, к которым можно получить доступ в шаблонах. Какие атрибуты имеют переменные, сильно зависит от приложения, предоставляющего эту переменную.

Сириус IT-Колледж

Jinja2. Переменные и выражения в шаблонах Jinja.

В шаблонах можно использовать точку (`.`) для доступа к атрибутам переменной в дополнение к стандартному синтаксису индексирования (`[]`) в Python `__getitem__()`.

Следующие строки делают одно и то же:

```
{{ foo.bar }}
{{ foo['bar'] }}
```

Важно понимать, что внешние двойные фигурные скобки `{{ ... }}` не являются частью переменной, а являются частью оператора шаблона для вывода переменной/выражения на печать. Если нужно обратится к переменным внутри тегов `{% ... %}`, например при использовании в цикле шаблона `{% for item in items %}`, то не заключайте переменные в фигурные скобки.

Сириус
IT-Колледж

Jinja2. Использование методов, определенных для типа Python.

В шаблонах Jinja2 можно использовать любой из методов, определенных для типа переменной. Значение, возвращаемое при вызове метода, используется как значение выражения. Вот пример, в котором используются методы, определенные для строк, где `page.title` - это строка:

```
{{ page.title.capitalize() }}
```

Это работает для методов с пользовательскими типами. Например, если для переменной foo типа Foo определен метод bar, то можно сделать следующее:

```
{{ foo.bar(value) }}
```

Сириус
IT-Колледж

Jinja2. Комментарии в шаблонах Jinja.

Чтобы закомментировать часть строки в шаблоне, используйте синтаксис комментария, который по умолчанию установлен на {# ... #}. Это полезно, чтобы закомментировать части шаблона для отладки или добавить информацию для других разработчиков шаблона, а также для себя:

```
{# note: commented-out template because we no longer use this
    {% for user in users %}
        ...
    {% endfor %}
#}
```

Jinja2. Наследование шаблонов jinja2 в Python

Самая мощная часть модуля jinja2 - это наследование шаблонов. Наследование шаблонов позволяет создать базовый "скелетный" шаблон, который содержит все общие элементы сайта и определяет блоки `{% block ... %}`, которые "дочерние" шаблоны могут переопределить.

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% block head %}
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}{% endblock %} - My Webpage</title>
    {% endblock %}
</head>
<body>
    <div id="content">{% block content %}{% endblock %}</div>
    <div id="footer">
        {% block footer %}
        &copy; Copyright 2008 by <a href="http://domain.invalid/">you</a>.
        {% endblock %}
    </div>
</body>
</html>
```

Дочерний шаблон Jinja2.

```
{% extends "base.html" %}
{% block title %}Index{% endblock %}
{% block head %}
    {{ super() }}
    <style type="text/css">
        .important { color: #336699; }
    </style>
{% endblock %}
{% block content %}
    <h1>Index</h1>
    <p class="important">
      Welcome to my awesome homepage.
    </p>
{% endblock %}
```

В дочернем шаблоне тег `{% extends ... %}` является ключевым. Он сообщает механизму шаблонов, что этот шаблон "расширяет" другой шаблон. Когда система шаблонов Jinja2 оценивает этот шаблон, она сначала находит родителя. Тег `extends` должен быть первым тегом в шаблоне.

Сириус
IT-Колледж

Jinja2. Именованные конечные теги блоков

Jinja позволяет ставить имя блока после закрывающего тега для лучшей читаемости:

```
{% block sidebar %}
    {% block inner_sidebar %}
        ...
    {% endblock inner_sidebar %}
{% endblock sidebar %}
```

При этом имя после слова `endblock` конечного блока обязательно должно совпадать с именем этого блока.

Сириус IT-Колледж

Jinja2. Циклы `for/in` в шаблонах jinja2 в Python

Цикл `for/in` в шаблонах jinja2 необходимо располагать внутри блоков `{% ... %}`. Например, чтобы отобразить список пользователей, указанный в переменной с именем `users`:

```
<h1>Members</h1>
<ul>
{% for user in users %}
  {% фильтр `|e` - экранирует HTML %}
  <li>{{ user.username|e }}</li>
{% endfor %}
</ul>
```

Сириус IT-Колледж

Jinja2. Циклы `for/in` в шаблонах jinja2 в Python

Так как переменные в шаблонах сохраняют свои свойства объектов, можно перебирать контейнеры, такие как словари Python dict:

```
<dl>
{% for key, value in my_dict.items() %}
    {% фильтр `|e` - экранирует HTML %}
    <dt>{{ key|e }}</dt>
    <dd>{{ value|e }}</dd>
{% endfor %}
</dl>
```

Обратите внимание, что словари Python по умолчанию не отсортированы, для их сортировки, в шаблон, можно либо передать отсортированный список кортежей, либо `collections.OrderedDict()`, либо использовать фильтр `dictsort()`.

Сириус
IT-Колледж

Jinja2. Оператор ветвления `if/elif/else` в шаблонах jinja2

Оператор `if/elif/else` в шаблонах Jinja2 сравним с оператором `if/else` в Python. В простейшей форме, можно использовать его, чтобы проверить, определена ли переменная, не пуста ли она или не ложна:

```
{% if users %}
<ul>
{% for user in users %}
    <li>{{ user.username|e }}</li>
{% endfor %}
</ul>
{% endif %}
```

Сириус
IT-Колледж

Jinja2. Оператор ветвления `if/elif/else` в шаблонах jinja2

Для нескольких веток можно использовать `{% elif ... %}` и `{% else %}`, как в Python. Также можно использовать более сложные выражения:

```
{% if kenny.sick %}
    Kenny is sick.
{% elif kenny.dead %}
    You killed Kenny!  You bastard!!!
{% else %}
    Kenny looks okay --- so far
{% endif %}
```

Сириус IT-Колледж

Jinja2. Оператор ветвления `if/elif/else` в шаблонах jinja2

В шаблонах Jinja2 доступно встроенные выражения `{% ... if ... else ... %}`, что полезно в некоторых ситуациях. Например, выражения `if/else` можно использовать для расширения шаблона из другого шаблона, если переменная определена, в противном случае из шаблона макета по умолчанию:

```
{% extends layout_template if layout_template is defined else 'master.html' %}
```

Общий синтаксис: <сделать что-то>, if <что-то истинно>, else <сделать что-то еще>.

Сириус IT-Колледж

Jinja2. Оператор ветвления `if/elif/else` в шаблонах jinja2

Часть выражения `else` в шаблоне необязательна. Если не предусмотрено другое поведение, то по умолчанию блок `else` неявно вычисляется в неопределенный объект `Undefined`:

```
{{ "[{}]".format(page.title) if page.title }}
```

Так же оператор шаблона `if` можно использовать в циклах шаблонов, для фильтрации последовательности в момент итерации, что позволяет пропускать ненужные элементы. В следующем примере пропускаются все скрытые пользователи:

```
{% for user in users if not user.hidden %}
    <li>{{ user.username|e }}</li>
{% endfor %}
```

Сириус IT-Колледж

FastAPI + Jinja2

Файл `main.py`:

```python
from fastapi import FastAPI, Request
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates

app = FastAPI()

app.mount("/static", StaticFiles(directory="static"), name="static")


templates = Jinja2Templates(directory="templates")


@app.get("/items/{id}", response_class=HTMLResponse)
async def read_item(request: Request, id: str):
    return templates.TemplateResponse(
        request=request, name="item.html", context={"id": id}
    )
```

Сириус
IT-Колледж

Далее, необходимо добавить файл шаблона `templates/item.html`, например, с таким содержимым:

```
<html>
<head>
    <title>Item Details</title>
    <link href="{{ url_for('static', path='/styles.css') }}" rel="stylesheet">
</head>
<body>
    <h1><a href="{{ url_for('read_item', id=id) }}">Item ID: {{ id }}</a></h1>
</body>
</html>
```