**Лабораторная работа №2-4**

Разработка структуры проекта. Диаграмма модулей. Перечень артефактов и протоколов. Настройка работы системы контроля версий. Разработка проекта на основании шаблона.

**Цель**: создание модульной структуры проекта в виде диаграммы, разработка перечня артефактов (основных файлов) и протоколов, настройка Git

**Краткие теоретические сведения**

*Артефакт* — это любой элемент программной системы. К артефактам, могут относиться исполняемые файлы, исходные тексты, веб-страницы, справочные файлы, сопроводительные документы, файлы с данными, модели и многое другое, являющееся физическим носителем информации. Другими словами, артефактами являются те информационные элементы, которые тем или иным способом используются при работе программной системы и входят в ее состав.  
  
С понятием «*компонент*» часто ассоциируют компонентное или сборочного программирование, однако это не верно с точки зрения UML. В терминах UML компонентное или сборочное программирование манипулирует артефактами. Пример показан ниже.

*Компонент (в UML)* — это частью модели, описывающая логическую сущность, которая существует только во время проектирования (design time), хотя в дальнейшем ее можно связать с физической реализацией (артефактом) времени исполнения (run time).

**Шаблоны проекта веб-приложений Python**

Python в Visual Studio поддерживает разработку веб-проектов на платформах **Bottle**, **Django** и **Flask** с помощью шаблонов проектов и средства запуска отладки, которое можно настроить для работы с различными платформами. Эти шаблоны включают файл requirements.txt для объявления необходимых зависимостей. При создании проекта на основе одного из таких шаблонов в Visual Studio отобразится запрос на установку этих пакетов.

**Использование шаблона проекта**

Чтобы создать проект на основе шаблона, последовательно выберите **Файл** > **Создать** > **Проект**. Чтобы просмотреть шаблоны для веб-проектов, слева в диалоговом окне последовательно выберите **Python** > **Интернет**. Выберите шаблон по своему усмотрению, присвоив имена проекту и решению. Затем задайте параметры для каталога решения и репозитория Git и нажмите кнопку ОК.

Для других платформ, например, **Pyramid**, также можно использовать универсальный шаблон веб-проекта. В этом случае для шаблона не предусмотрена установка каких-либо платформ. Вместо этого в окружении проекта устанавливаются необходимые пакеты.

Универсальный шаблон **Веб-проект**, содержит пустой проект Visual Studio без кода (подразумевается только проект Python).

Все другие шаблоны основаны на веб-платформах Bottle, Flask или Django и делятся на три общие группы, как описано в следующих разделах. Приложения, созданные с помощью любого из этих шаблонов, содержат код, который обеспечивает локальное выполнение и отладку приложений. Кроме того, каждый из них предоставляет необходимый объект приложения WSGI (python.org) для использования с рабочими веб-серверами.

**Пустая группа**

С помощью всех шаблонов пустых веб-проектов для <framework> создаются проекты с использованием большего или меньшего объема стандартного кода и необходимых зависимостей, объявленных в файле **requirements.txt**.

|  |  |
| --- | --- |
| **Шаблон** | **Description** |
| Пустой веб-проект Bottle | В app.py создается минимальное приложение с домашней страницей для / и страницей /hello/<name>, на которой выводится <name>. Используется небольшой встроенный шаблон. |
| Пустой веб-проект Django | Создается проект Django с основной структурой веб-сайта Django, но без структуры приложений Django. |
| Пустой веб-проект Flask | Создается минимальное приложение для вывода одной страницы "Hello World!" для /. Это приложение похоже на результат выполнения инструкций в кратком руководстве по созданию первого веб-приложения Python с помощью Visual Studio. См. также шаг 1 руководства по работе с Flask. |

**Веб-группа**

С помощью всех шаблонов веб-проектов для <Framework> создаются начальные веб-приложения с идентичной структурой независимо от выбранной платформы. В приложении есть **домашняя страница**, **страницы со сведениями** и **контактными данными**, а также панель навигации и адаптивный макет, созданный на основе Bootstrap. Все приложения соответствующим образом настроены для предоставления статических файлов (CSS, JavaScript и шрифты). Используется механизм шаблона страницы, соответствующий конкретной платформе.

|  |  |
| --- | --- |
| **Шаблон** | **Description** |
| Веб-проект Bottle | Создается приложение, статические файлы которого содержатся в папке static и обрабатываются с помощью кода в app.py. Данные маршрутизации для отдельных страниц содержатся в routes.py, а шаблоны страниц — в папке views. |
| Веб-проект Django | Создается проект и приложение Django с тремя страницами, поддержкой аутентификации и базой данных SQLite (без моделей данных). Дополнительные сведения см. в статье о шаблонах Django и шаге 4 руководства по изучению Django. |
| Веб-проект Flask | Создается приложение, статические файлы которого содержатся в папке static. Маршрутизация обрабатывается с помощью кода во views.py. Шаблоны страниц, созданные с использованием механизма Jinja, содержатся в папке templates. Код запуска помещен в файл runserver.py. См. шаг 4 руководства по работе с Flask. |
| Веб-проект Flask и Jade | Создается то же приложение, что и при использовании шаблона Веб-проект Flask, но с помощью расширения Jade для модуля создания шаблонов Jinja. |

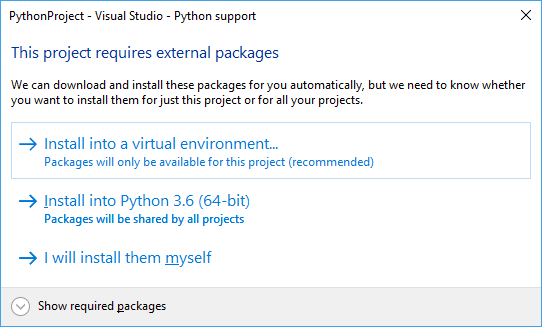
**Группа опроса**

С помощью шаблонов веб-проектов опроса для <framework> создаются начальные веб-приложения, в которых пользователи могут пройти опрос. Каждое приложение основано на структуре шаблона веб-проекта, которая позволяет использовать базу данных для управления опросами и ответами пользователей. Эти приложения включают соответствующие модели данных и специальную страницу приложения (/seed), с помощью которой из файла **samples.json** загружается опрос.

|  |  |
| --- | --- |
| **Шаблон** | **Description** |
| Веб-проект опроса Bottle | Создается приложение, которое можно запустить с выполняющейся в памяти базой данных, MongoDB или службой "Хранилище таблиц Azure", которая настраивается с помощью переменной среды REPOSITORY\_NAME. Модели данных и код хранилища данных содержатся в папке models. В файл settings.py помещен код, который определяет используемое хранилище данных. |
| Веб-проект опроса Django | Создаются проект и приложение Django с тремя страницами и базой данных SQLite. Шаблон включает настройки в интерфейсе администратора Django, позволяют прошедшему аутентификацию администратору создавать опросы и управлять ими. |
| Веб-проект опроса Flask | Создается приложение, которое можно запустить с выполняющейся в памяти базой данных, MongoDB или службой "Хранилище таблиц Azure", которая настраивается с помощью переменной среды REPOSITORY\_NAME. Модели данных и код хранилища данных содержатся в папке models. В файл settings.py помещен код, который определяет используемое хранилище данных. В приложении применяется модуль Jinja для создания шаблонов страниц. |
| Веб-проект опроса Flask и Jade | Создается то же приложение, что и при использовании шаблона Веб-проект опроса Flask, но с помощью расширения Jade для модуля создания шаблонов Jinja. |

**Установка требований проекта**

При создании проекта из шаблона для конкретной платформы отображается диалоговое окно для установки необходимых пакетов с помощью pip. Чтобы при публикации веб-сайта были настроены правильные зависимости, для веб-проектов рекомендуется использовать **виртуальную среду**.

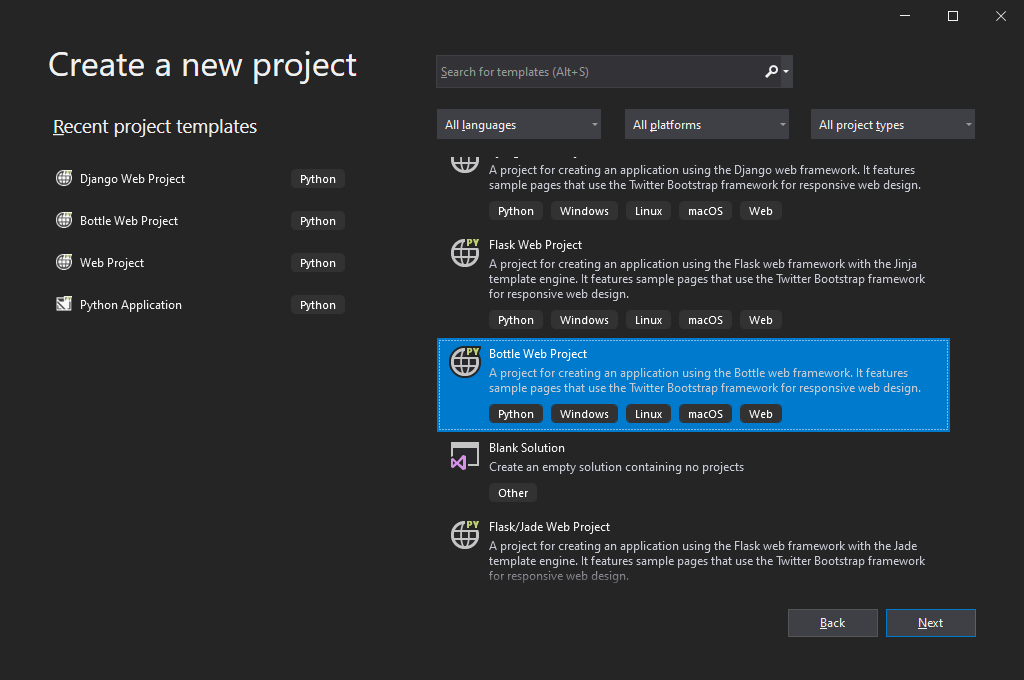


Если используется система управления версиями, папка виртуального окружения обычно опускается, так как виртуальное окружение можно воссоздать только с помощью **requirements.txt**. Чтобы исключить папку, рекомендуем сначала выбрать вариант самостоятельной установки, показанный выше в окне с запросом, а затем отключить автоматическую фиксацию перед созданием виртуального окружения.

Выполняя развертывание в службе приложений Microsoft Azure, нужно выбрать версию Python в качестве расширения сайта и установить пакеты вручную. Кроме того, поскольку при развертывании из Visual Studio служба приложений Azure не устанавливает пакеты из файла requirements.txt автоматически, используйте сведения о конфигурации, приведенные на странице aka.ms/PythonOnAppService.

**Bottle** — это мини-фреймворк для Python, позволяющий писать веб-приложения с высокой скоростью. Подробнее на <https://habr.com/ru/post/221659/>

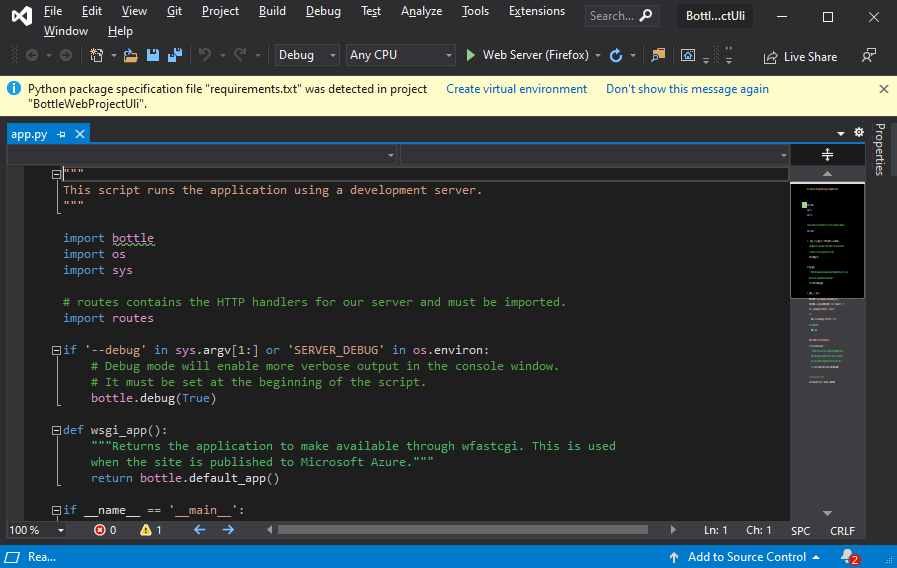
**Выполнение работы:**

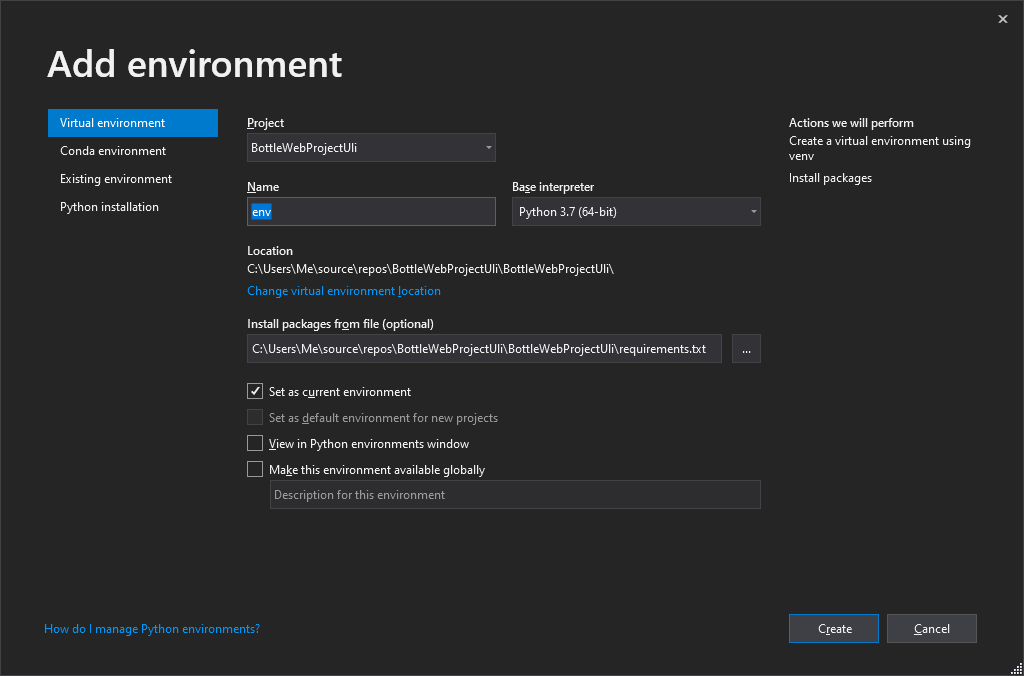
* 1. Требуется создать веб-приложение (сайт) на основе имеющегося шаблона. Выбрать **Файл** > **Создать** > **Проект (Create a new project)**. Далее в открывшемся окне выбрать **Bottle Web Project** и нажать **Next**:

Задать название проекта (в наименование должна быть включена фамилия студента) > **Create**.

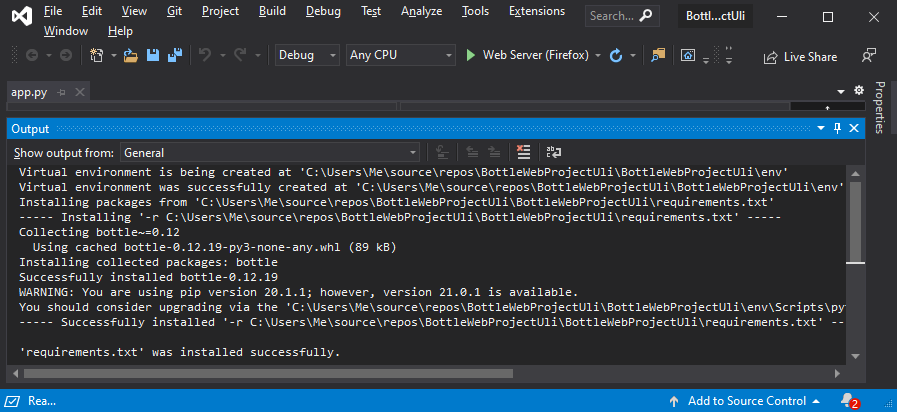
Процесс создания проекта займёт некоторое время.

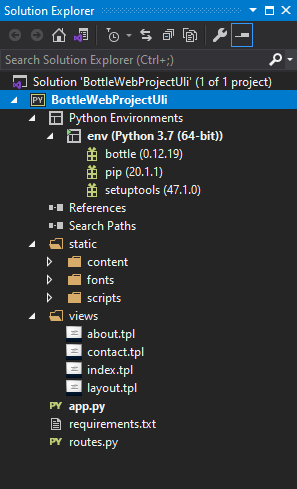
* 1. По умолчанию в редакторе кода откроется файл **app.py**. Для корректной интерпретации содержимого файла **requirements.txt** будет предложено создать **виртуальное окружение**:



Выбрать указанный пункт **Create virtual environment**. Имя можно оставить предлагаемое по умолчанию, нажать **Create**.

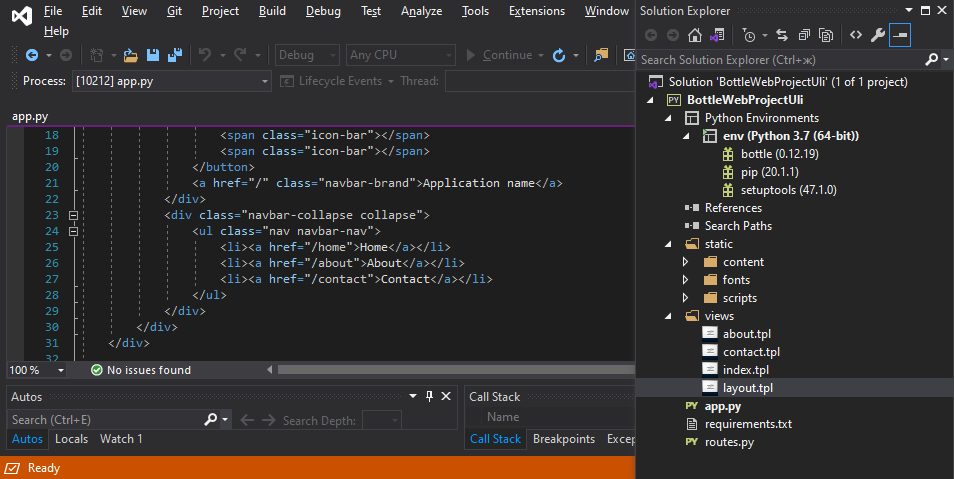
Если все операции по созданию виртуального окружения выполнены успешно, в окне вывода должно отобразиться соответствующее подтверждающее сообщение:

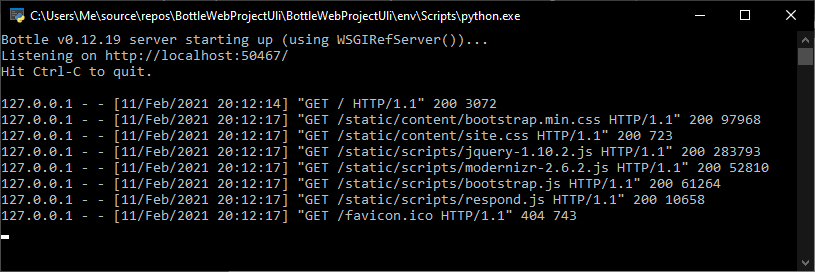


* 1. Ознакомиться со структурой проекта, открыв **Обозреватель решений** (**Solution Explorer**):

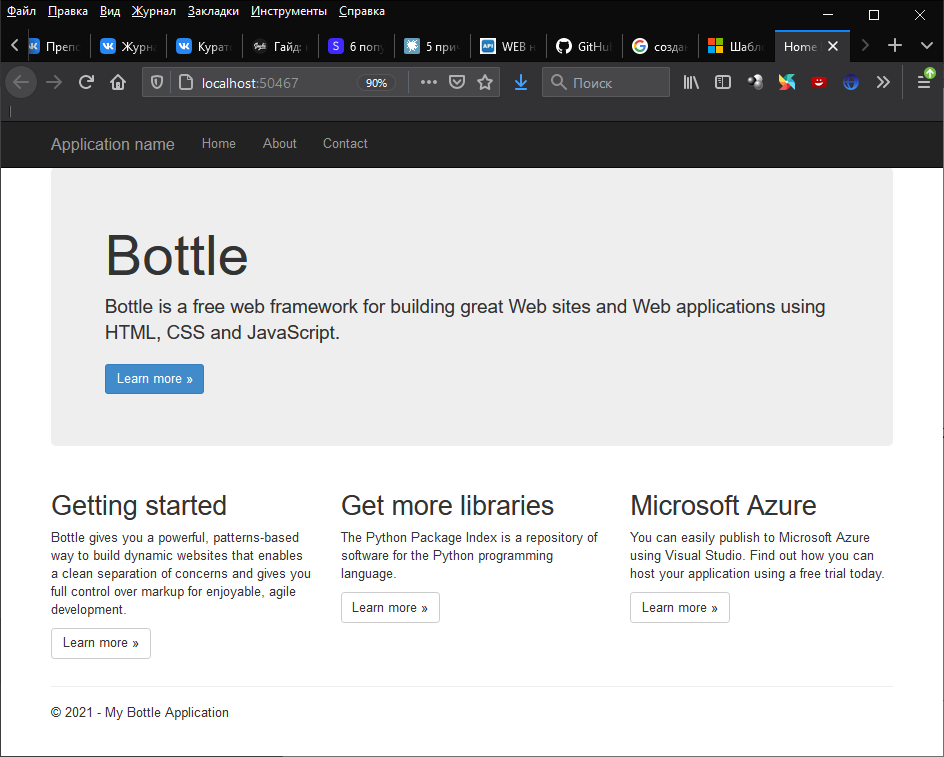
В папке **static** содержатся стилевые css-файлы для оформления внешнего вида веб-страниц, шрифтовые наборы и js-скрипты.

Выбрав для просмотра файлы из **папки** views, можно убедиться, что они представляют собой макеты веб-страниц: **главной** (index.tpl), **о нас** (about.tpl), **контакты** (contact.tpl), а файл layout.tpl – не что иное, как шаблон обёртки с панелью навигации, содержащей четыре гиперссылки: Application name, Home, About, Contact в «шапке» сайта и небольшим «подвалом».

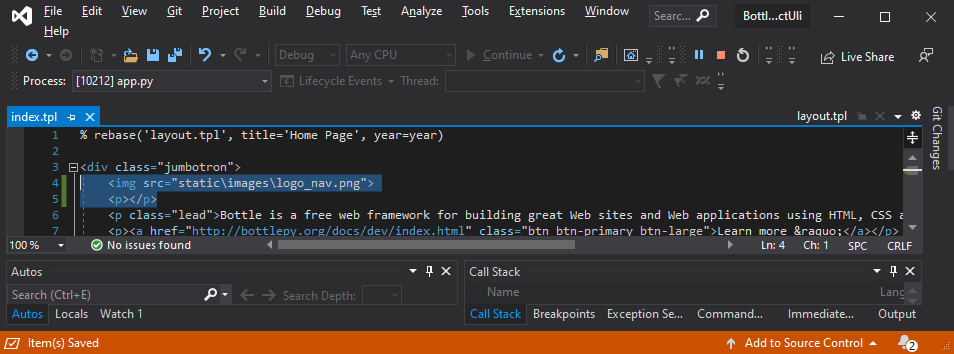


* 1. Выполнить запуск проекта (вместо **Start** рядом c зелёной стрелкой должна быть надпись **Web Server**). Т.к. в Python встроен свой локальный сервер с поддержкой стандарта взаимодействия WSGI (Web Server Gateway Interface), то автоматически к нему будут отправлены запросы на отображение главной страницы созданного приложения. В окне интерпретатора можно увидеть соответствующие заголовки запросов и ответов сервера:

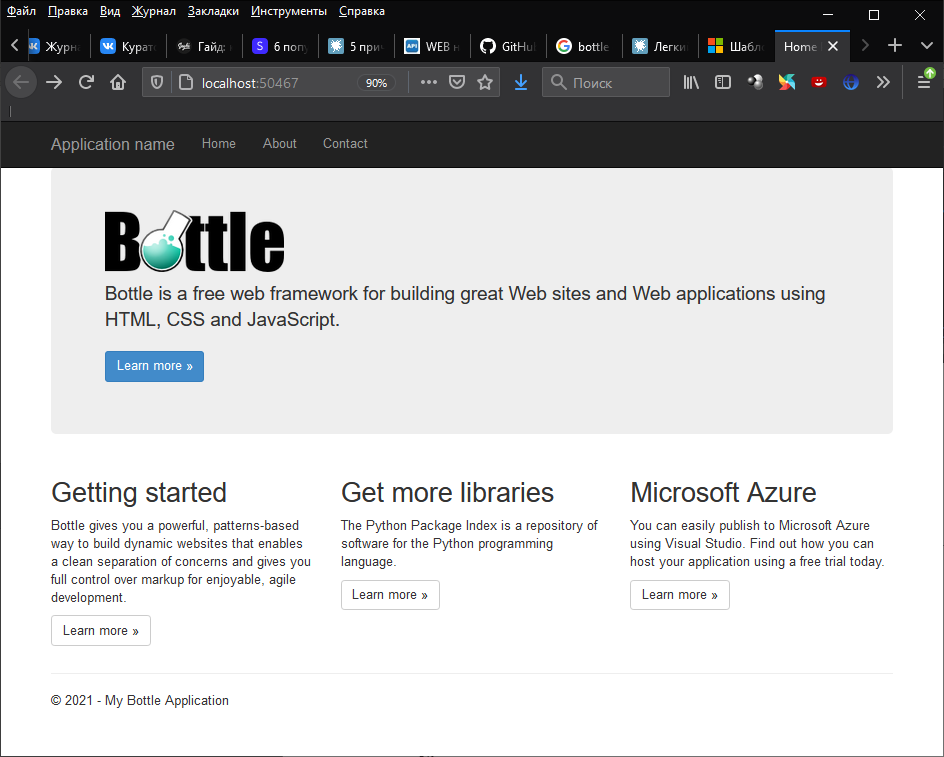
А в новой вкладке браузера (в данном случае Firefox) с localhost будет открыта главная страница сайта index.html, формируемая на основе шаблона index.tpl. На странице два основных блока: первый – с заголовком, коротким абзацем и кнопкой, второй – с перечнем из трех вложенных подразделов похожей структуры:



* 1. Приложение, созданное по шаблону, может быть наполнено нужным текстовым и графическим содержимым. Для изменения названия пунктов панели навигации нужно отредактировать layout.tpl, переписав наименования внутри тэгов <a href …> **Имя пункта** </a>. Для изменения заголовков и подзаголовков на страницах index.tpl и др. редактируются данные внутри тэгов соответственно <h1> **Заголовок** </h1>, <h2> **Подзаголовок** </h2>, а для изменения содержимого абзацев – внутри <p> **Текст абзаца** </p>. Для вставки изображения в нужный блок следует добавить тэг <img src="static\images\**Файл изображения**">.

Например, вместо текста заголовка «Bottle» на главной (домашней) странице требуется разместить картинку. Для этого сначала в структуре каталога **static** создаётся папка **images**, в которую копируется файл изображения **logo\_nav.png**. Затем в коде **index.tpl** удалить строку <h1> Bottle </h1>, добавить **<img src="static\images\logo\_nav.png">** (файл изображения можно скачать с сайта http://bottlepy.org/docs/dev/index.html) и пустой абзац **<p> </p>**:

Сохранив изменения в VS и обновив страницу сайта в браузере должен получиться результат:



* 1. По аналогии с ЛР1 выполнить настройку Git для созданного приложения.

***Примечание***: следующие задания (п.7) выполняются в команде из 2-3 человек.

* 1. Требуется создать веб-приложение (тематический сайт) на основе шаблона фреймворка Bottle. Придерживаться следующих этапов:
  2. Выбрать тематику приложения, согласовать с преподавателем (темы не повторяются!).
  3. Подобрать текстовые и графические (иллюстрации, иконки, карты) материалы для сайта.
  4. Построить UML-диаграмму компонентов приложения (css-файлы считать одним укрупнённым структурным компонентом, аналогично js-скрипты, их подключение осуществляется в файле layout.tpl).
  5. Настроить Git. Хранение проекта осуществлять в репозитории на Github одного из участников, остальные клонируют проект в свой локальный репозиторий.
  6. Распределить задачи редактирования файлов шаблонов между членами команды. Изменить содержимое этих файлов согласно выбранной тематике (ссылки на кнопках поставить на нужные страницы похожих по тематике сайтов).
  7. Вести контроль версий. Синхронизировать все локальные изменения с удалённым репозиторием на Github.
  8. Заполнить файл README.md. Примерный вариант описания можно найти в шаблоне <https://gist.github.com/PurpleBooth/109311bb0361f32d87a2>

или по ссылке <https://github.com/Sinclear/default_readme>

**Контрольные вопросы:**

1. Каково основное назначение UML-диаграмм и их роль в разработке приложений?
2. Можно ли считать артефактами веб-страницы сайта? Обоснуйте ответ.
3. Предпочтительнее ли протокол HTTPS протоколу HTTP? Почему (да / нет)?
4. Что значит код ответа сервера «200»? Какие ещё группы и коды ответов вы знаете (привести по 1-2 примера из каждой группы)?
5. Какую систему защиты и сертификации данных использует Git по умолчанию?
6. Что значит «клонировать» репозиторий в терминах Git?