

## ПЗ №3 Начало работы с FastAPI

FastAPI — это веб-фреймворк Python, основы использования которого необходимо осваивать как пример веб разработки на Python. Это быстрый и легкий современный API, который проще в освоении по сравнению с другими веб-фреймворками на основе Python, такими как Flask и Django. FastAPI относительно новый, но его сообщество растет. Он широко используется при создании веб-API и развертывании моделей машинного обучения.

Освоим настроить среду разработки и создать свое простейшее приложение FastAPI. В начале - изучение основ Git — системы управления версиями — чтобы дать знания о хранении, отслеживании и извлечении изменений файлов при создании приложения. Далее - работа с пакетами в Python с помощью `pip`, как создавать изолированные среды разработки с помощью `Virtualenv` и познакомитесь с основами `Docker`. Наконец, вы познакомитесь с основами FastAPI, создав простое приложение *Hello World*.

Понимание ранее упомянутых технологий требуется для создания полноценного приложения FastAPI. Это также служит дополнением к текущему набору навыков.

По завершении вы сможете настраивать и использовать Git, устанавливать пакеты и управлять ими с помощью `pip`, создавать изолированную среду разработки с помощью `Virtualenv`, использовать `Docker` и, что наиболее важно, создавать шаблоны для приложений FastAPI.

В теоретической части задания рассматриваются следующие темы:

- Основы Git
- Создание изолированных сред разработки с помощью `Virtualenv`
- Управление пакетами с помощью `pip`
- Настройка и изучение основ `Docker`
- Создание простого приложения Fast API

### 3.1. Основы Git

Git — это система контроля версий, которая позволяет разработчикам записывать, отслеживать и возвращаться к более ранним версиям файлов. Это децентрализованный и легкий инструмент, который можно установить в любой операционной системе.

Вы узнаете, как использовать Git для ведения учета. По мере создания каждого уровня приложения будут вноситься изменения, и важно, чтобы эти изменения сохранялись.

#### 3.1.1. Установка Git

Чтобы установить Git, посетите страницу загрузок по ссылке <https://git-scm.com/downloads> и выберите вариант загрузки для вашей текущей операционной системы. Вы будете перенаправлены на страницу с инструкциями по установке Git на

свой компьютер.

Также стоит отметить, что Git поставляется как с CLI, так и с GUI интерфейсом. Таким образом, вы можете скачать тот, который лучше всего подходит для вас.

### 3.1.2. Git операции

Как упоминалось ранее, Git можно использовать для записи, отслеживания и возврата к более ранним версиям файла. Однако в этой книге будут использоваться только основные операции Git, которые будут представлены в этом разделе.

Чтобы Git работал правильно, папки с файлами должны быть инициализированы. Инициализация папок позволяет Git отслеживать содержимое, за исключением исключений.

Чтобы инициализировать новый репозиторий Git в вашем проекте, вам нужно запустить следующую команду в своем терминале:

```
$ git init
```

Чтобы включить отслеживание файлов, файл необходимо сначала добавить и зафиксировать. Коммит Git позволяет отслеживать изменения файлов между временными рамками; например, коммит, сделанный час назад, и текущая версия файла.

Что такое коммит (фиксация)? Фиксация — это уникальный захват состояния файла или папки в определенное время, идентифицируемый уникальным кодом.

Теперь, когда мы знаем, что такое коммит, мы можем продолжить и зафиксировать файл следующим образом:

```
$ git add hello.txt $ git commit -m "Initial commit"
```

Вы можете отслеживать состояние ваших файлов после внесения изменений, выполнив следующую команду:

```
$ git status
```

Ваш терминал должен выглядеть примерно так:


A terminal window titled 'youngestdev@Abduls-MacBook-Air:~/Documents/FastAPI-Book'. It shows a sequence of Git commands and their outputs: 'git add hello.txt', 'git commit -m "Initial commit"' (resulting in commit eda7e6c), 'echo "This is a new addition to the file" > hello.txt', and 'git status'. The status shows 'On branch main' and 'Changes not staged for commit: modified: hello.txt'. It also shows 'no changes added to commit'. The prompt is 'FastAPI-Book git:(main) \*'.

Рисунок 3.1 - Git команды (терминал Mac)

Чтобы просмотреть изменения, внесенные в файл, которые могут быть дополнениями или вычитаниями из содержимого файла, выполните следующую команду:

```
$ git diff
```

Ваш терминал должен выглядеть примерно так:


A terminal window titled 'git diff hello.txt'. It shows the output of the 'git diff' command, comparing the index (e69de29..910b351) with the working directory (a/hello.txt). The output shows a single line added: '+This is a new addition to the file'. The prompt is 'FastAPI-Book git:(main) \*'.

Рисунок 3.2 - Вывод команды git diff (терминал Mac)

Рекомендуется включать файл .gitignore в каждую папку. The Файл .gitignore содержит имена файлов и папок, которые Git игнорирует. Таким образом, вы можете

добавить и зафиксировать все файлы в вашей папке, не опасаясь зафиксировать такие файлы, как `.env`.

Чтобы включить файл `.gitignore`, выполните в терминале следующую команду:

```
$ touch .gitignore
```

Чтобы освободить файл от отслеживания Git, добавьте его в файл `.gitignore` следующим образом:

```
$ echo ".env" >> .gitignore
```

Общие файлы, содержащиеся в файле `.gitignore`, включают следующее:

- Файлы окружения (`*.env`)
- Виртуальная папка (`env`, `venv`)
- Папки метаданных IDE (такие как `.vscode` и `.idea`)

### 3.1.3. Git ветки

Ветки — это важная функция, которая позволяет разработчикам легко работать над различными функциями приложения, ошибками и т. д. по отдельности, прежде чем объединиться с основной веткой. Система ветвления используется как в небольших, так и в крупных приложениях и продвигает культуру предварительного просмотра и совместной работы с помощью запросов на включение. Первичная ветвь называется основной ветвью, и это ветвь, из которой создаются другие ветки.

Чтобы создать новую ветку из существующей ветки, мы запускаем команду `git checkout -b newbranch`. Давайте создадим новую ветку, выполнив следующую команду:

```
$ git checkout -b hello-python-branch
```

Предыдущая команда создает новую ветвь из существующей, а затем устанавливает активную ветвь на вновь созданную ветвь. Чтобы вернуться к исходной основной ветке, мы запускаем `git checkout main` следующим образом:

```
$ git checkout main
```

Важная заметка. Запуск `git checkout main` делает `main` активной рабочей веткой, тогда как `git checkout -b newbranch` создает новую ветку из текущей рабочей ветки и устанавливает вновь созданную ветку в качестве активной.

Теперь, когда мы изучили основы Git, мы можем приступить к изучению того, как создавать изолированные среды с помощью `virtualenv`.

## 3.2. Создание изолированных сред разработки с помощью Virtualenv

Традиционный подход к разработке приложений на Python заключается в изоляции этих приложений в виртуальной среде. Это сделано для того, чтобы избежать глобальной установки пакетов и уменьшить количество конфликтов во время разработки приложений.

Виртуальная среда — это изолированная среда, в которой установленные зависимости приложений доступны только внутри нее. В результате приложение может получать доступ только к пакетам и взаимодействовать только внутри этой среды.

### 3.2.1. Создание виртуальной среды

По умолчанию в Python3 установлен модуль `venv` из стандартной библиотеки. Модуль `venv` отвечает за создание виртуальной среды. Давайте создадим папку `todos` и создадим в ней виртуальную среду, выполнив следующие команды:

```
$ mkdir todos && cd todos  
$ python3 -m venv venv
```

Модуль `venv` принимает в качестве аргумента имя папки, в которую следует установить виртуальную среду. В нашей только что созданной виртуальной среде копия интерпретатора Python установлена в папке `lib` а файлы, обеспечивающие взаимодействие внутри виртуальной среды, хранятся в папке `bin`.

В среде Windows установка и использование виртуального окружения выполняется разными способами. Например, можно использовать следующий.

```
$ pip install virtualenv  
$ pip install virtualenvwrapper-win
```

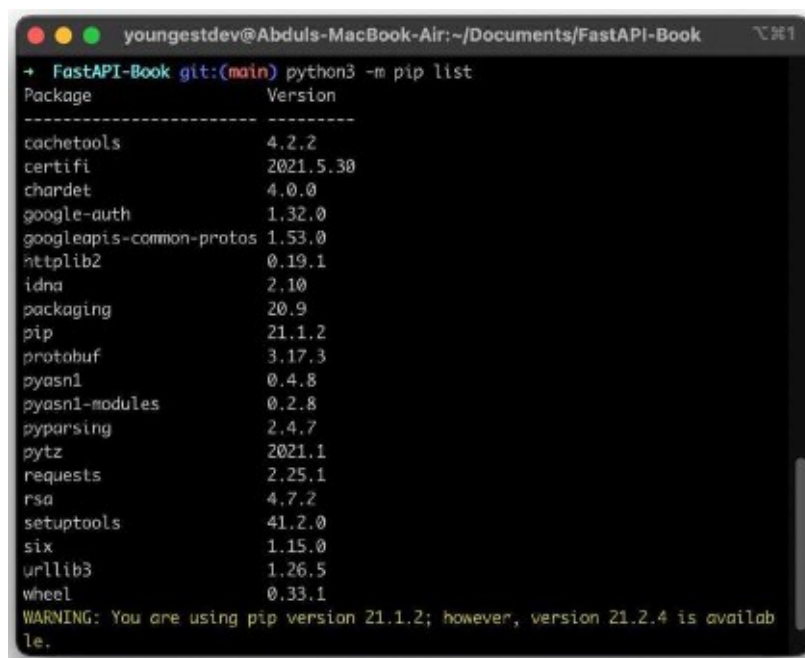
### 3.2.2. Активация и деактивация виртуальной среды

Чтобы активировать виртуальную среду, мы запускаем следующую команду:

```
$ source venv/bin/activate
```

Предыдущая команда указывает вашей оболочке использовать интерпретатор и пакеты виртуальной среды по умолчанию. После активации виртуальной среды

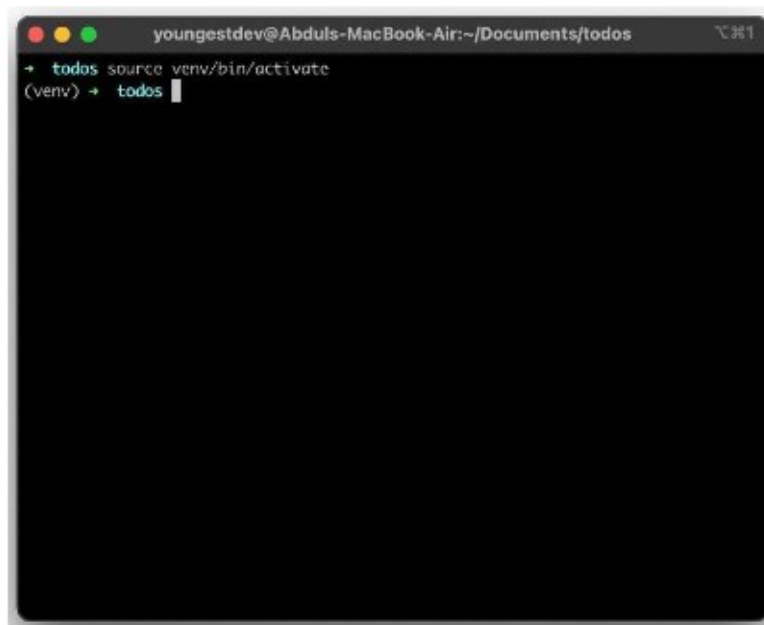
префикс папки виртуальной среды `venv` добавляется перед приглашением следующим образом:



```
youngestdev@Abduls-MacBook-Air:~/Documents/FastAPI-Book
+ FastAPI-Book git:(main) python3 -m pip list
Package            Version
-----
cachetools         4.2.2
certifi            2021.5.30
chardet            4.0.0
google-auth       1.32.0
googleapis-common-protos 1.53.0
httplib2           0.19.1
idna               2.10
packaging          20.9
pip               21.1.2
protobuf          3.17.3
pyasn1            0.4.8
pyasn1-modules    0.2.8
pyparsing         2.4.7
pytz              2021.1
requests          2.25.1
rsa               4.7.2
setuptools        41.2.0
six               1.15.0
urllib3           1.26.5
wheel             0.33.1
WARNING: You are using pip version 21.1.2; however, version 21.2.4 is available.
```

Рисунок 1.3 – Подсказка с префиксом

Чтобы деактивировать виртуальную среду, в командной строке запускается команда `deactivate`. При выполнении команды происходит немедленный выход из изолированной среды, а префикс удаляется следующим образом:



```
youngestdev@Abduls-MacBook-Air:~/Documents/todos
+ todos source venv/bin/activate
(venv) + todos
```

Рисунок 1.4 - Деактивация виртуальной среды

Важная заметка. Вы также можете создать виртуальную среду и управлять зависимостями приложений, используя *Pipenv* и *Poetry*.

В среде Windows соответствующие команды представлены таблице.

Команда	Описание
<code>mkvirtualenv env-name</code>	Создаем новое окружение
<code>workon</code>	Смотрим список окружений
<code>workon env-name</code>	Меняем окружение
<code>deactivate</code>	Выходим из окружения
<code>rmvirtualenv env-name</code>	Удаляем окружение

Теперь, когда мы создали виртуальную среду, мы можем перейти к пониманию того, как работает управление пакетами с помощью `pip`.

### 3.3. Управление пакетами с помощью `pip`

Приложение FastAPI представляет собой пакеты, поэтому вы познакомитесь с методами управления пакетами, такими как установка пакетов, удаление пакетов и обновление пакетов для вашего приложения.

Установка пакетов из исходного кода может оказаться сложной задачей, поскольку в большинстве случаев она включает в себя загрузку и распаковку файлов. `tar.gz` перед установкой вручную. В сценарии, где необходимо установить сто пакетов, этот метод становится неэффективным. Тогда как автоматизировать этот процесс?

`Pip` — это менеджер пакетов Python, подобный JavaScript's `yarn`; он позволяет автоматизировать процесс установки пакетов Python как глобально, так и локально.

#### 3.3.1. Установка `pip`

`Pip` автоматически устанавливается во время установки Python. Вы можете проверить, установлен ли `pip`, выполнив следующую команду в своем терминале:

```
$ python3 -m pip list
```

Предыдущая команда должна вернуть список установленных пакетов. Результат должен быть похож на следующий рисунок:

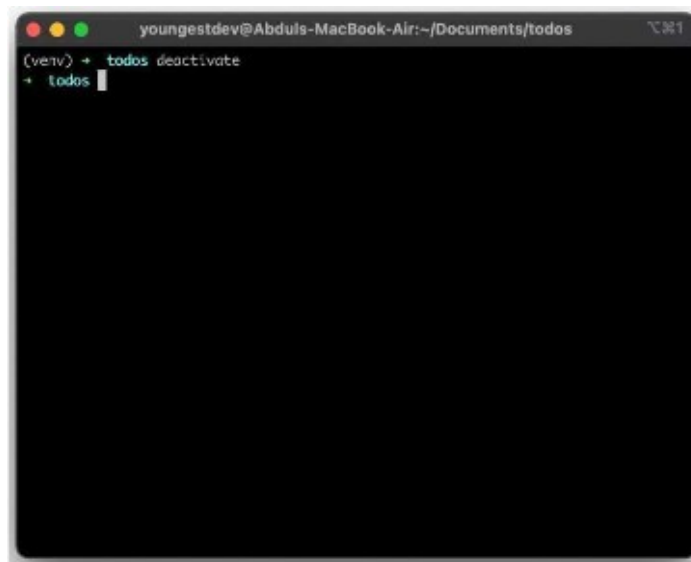


Рисунок 1.5 - Список установленных пакетов Python

Если команда возвращает ошибку, следуйте инструкциям на странице <https://pip.pypa.io/en/stable/installation/>, чтобы установить pip.

### 3.3.2. Основные команды

Установив pip, давайте изучим его основные команды. Чтобы установить пакет FastAPI с помощью pip, мы запускаем следующую команду:

```
$ pip install fastapi
```

В операционной системе Unix, такой как Mac или Linux, в некоторых случаях ключевое слово `sudo` добавляется перед установкой глобальных пакетов.

Для удаления пакета используется следующая команда:

```
$ pip uninstall fastapi
```

Чтобы собрать текущие пакеты, установленные в проекте, в файл, мы используем следующую команду `freeze`:

```
$ pip freeze > requirements.txt
```

Оператор `>` указывает `bash` сохранить вывод команды в файл `requirements.txt`. Это означает, что запуск `pip freeze` возвращает все установленные в данный момент пакеты.

Чтобы установить пакеты из файла, такого как файл `requirements.txt`, используется следующая команда:

```
$ pip install -r requirements.txt
```



Предыдущая команда в основном используется при развертывании.

Теперь, когда вы изучили основы `pip` и ознакомились с некоторыми основными командами, давайте изучим основы Docker.

## 3.4.Настройка Docker

По мере того, как наше приложение становится многоуровневым, например, база данных, объединение приложения в единый элемент позволяет нам развертывать наше приложение. Мы будем использовать Docker для контейнеризации уровней наших приложений в единый образ, который затем можно будет легко развернуть локально или в облаке.

Кроме того, использование Dockerfile и файла `docker-compose` избавляет от необходимости загружать образы наших приложений и делиться ими. Новые версии наших приложений можно создавать из файла Dockerfile и развертывать с помощью файла `docker-compose`. Образы приложений также можно хранить и извлекать из Docker Hub. Это известно, как операция толкания и вытягивания.

Чтобы начать настройку, загрузите и установите Docker с <https://docs.docker.com/install>.

### 3.4.1. Dockerfile

Dockerfile содержит инструкции о том, как должен быть создан образ нашего приложения. Ниже приведен пример Dockerfile:

```
FROM PYTHON:3.8
# Set working directory to /usr/src/app WORKDIR /usr/src/app
# Copy the contents of the current local directory into the container's working directory
ADD . /usr/src/app
# Run a command
CMD ["python", "hello.py"]
```

Далее мы создадим образ контейнера приложения и назовем `getting_started` следующим образом:

```
$ docker build -t getting_started .
```

Если Dockerfile отсутствует в каталоге, где запускается команда, путь к Dockerfile должен быть правильно добавлен следующим образом:

```
$ docker build -t api api/Dockerfile
```

Образ контейнера можно запустить с помощью следующей команды:

```
$ docker run getting-started
```

Docker — эффективный инструмент для контейнеризации. Мы рассмотрели только основные операции.

### 3.5.Создание простого приложения FastAPI

Наконец, теперь мы можем перейти к нашему первому проекту Fast API. Наша цель в этом разделе — представить FastAPI, создав простое приложение. Мы подробно рассмотрим операции в последующих главах.

Мы начнем с установки зависимостей, необходимых для нашего приложения, в папку todos которую мы создали ранее. Зависимости следующие:

- fastapi: Фреймворк, на котором мы будем строить наше приложение.
- uvicorn: Модуль Asynchronous Server Gateway Interface для запуска приложения.

Сначала активируйте среду разработки, выполнив следующую команду в каталоге вашего проекта:

```
$ source venv/bin/activate
```

Затем установите зависимости следующим образом:

```
(venv)$ pip install fastapi uvicorn
```

А пока мы создадим новый файл api.py и создадим новый экземпляр FastAPI следующим образом:

```
from fastapi import FastAPI app = FastAPI()
```

Создав экземпляр FastAPI в переменной приложения, мы можем приступить к созданию маршрутов. Создадим приветственный маршрут.

Маршрут создается, сначала определяя декоратор для указания типа операции, а затем функцию, содержащую операцию, которая будет выполняться при вызове этого маршрута. В следующем примере мы создадим маршрут "/", который принимает только запросы GET requests и возвращает приветственное сообщение при посещении:

```
@app.get("/")
async def welcome() -> dict:
    return { "message": "Hello World"}
```

Следующим шагом будет запуск нашего приложения с помощью uvicorn. В терминале выполните следующую команду:

```
(venv)$ uvicorn api:app --port 8000 --reload
```

В предыдущей команде, `uvicorn` принимает следующие аргументы:

- `file: instance:` Файл, содержащий экземпляр FastAPI и переменную имени, содержащую экземпляр FastAPI..
- `--port PORT:` Порт, на котором будет обслуживаться приложение.
- `--reload:` Необязательный аргумент, включенный для перезапуска приложения при каждом изменении файла.

Команда возвращает следующий вывод:

```
(venv) -- todos uvicorn api:app --port 8080 --reload
INFO: Will watch for changes in these directories: ['/'
Users/youngestdev/Documents/todos']
INFO: Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C
to quit)
INFO: Startedreloader process [3982] using statreload
INFO: Startedserver process [3984]
INFO: Waitingfor application startup.
INFO: Application startup complete.
```

Следующий шаг — протестировать приложение, отправив запрос GET в API. В новом терминале отправьте запрос GET с помощью `curl` следующим образом:

```
$ curl http://0.0.0.0:8080/
```

Ответ от приложения в консоле будет следующим:

```
{"message": "Hello World"}
```

## Задание

1. Изучить теоретический материал.
2. Провести установку рассмотренных компонентов. Для каждого компонента сохранить копии экрана .
3. Продемонстрировать запуск приложения и его тест.
4. Подготовить и сдать отчет.