

---

# Table of Contents

Introduction	1.1
Syllabus	1.2
How to use it	1.3
Let's get it started	1.4
Hello Python	1.5
MVC and Django	1.6
REST API & DRF	1.7
Tools	1.8
Don't Repeat Yourself	1.9
Initial commit	1.10
Project detail	1.11
Basic steps	1.12
Application of Authentication	1.13
User profile and roles	1.14
Application of Projects	1.15
Application of Teams	1.16
Application of Scrum	1.17
Deployment with Docker	1.18
Single Page Application	1.19
Vue JS basics	1.20
REST API & DRF	1.21

# Introduction

Dear guest welcome to the '**Be Full-Stack**' tutorial. My name is Anuar and I want to share my experience by guiding you through whole development process.

## What is Full Stack Development?

To be or not to be Full-Stack, that is the question.

# FULL-STACK DEVELOPMENT



**Full stack developer** - we could precisely cover it with one Phrase “The captain of all ships.” A person who could handle all the required technologies to lead and turn the application development into astounding tailored solution. To the extent of becoming a full-stack developer, you must profound in both front-end and back-end technologies.

## What are we going to do?

Have you ever learned any technology or programming from tutorial? The problem is that after covering it you don't know how to put into practice things you have recently learned, but this one is an exception.

It's an exception because after completing it you can be one of the contributor for that project. Idea - is to guide you through whole development process explaining every step of the real project. Then if you have any idea, feature or find a bug you will easily realize and fix

it, since you know everything about it. It's also good for your resume to be a part of open-source project as a junior developer. At least you can deploy it on your server and use it as well.

## What about the project?

I decided to create something like [JIRA](#) and called it '**gitGile**'. It's a portal (consists of Back-end and Front-end) which helps for every member of software team to plan, track and release software. For the Back-end side I'm gonna use Python3 and [Django-Rest-Framework](#). For the Front-end side Vue.js is selected.

So I hope that this guidance will be interesting and useful for you. Have a nice trip and good luck.

## What if I have some ideas or complaints?

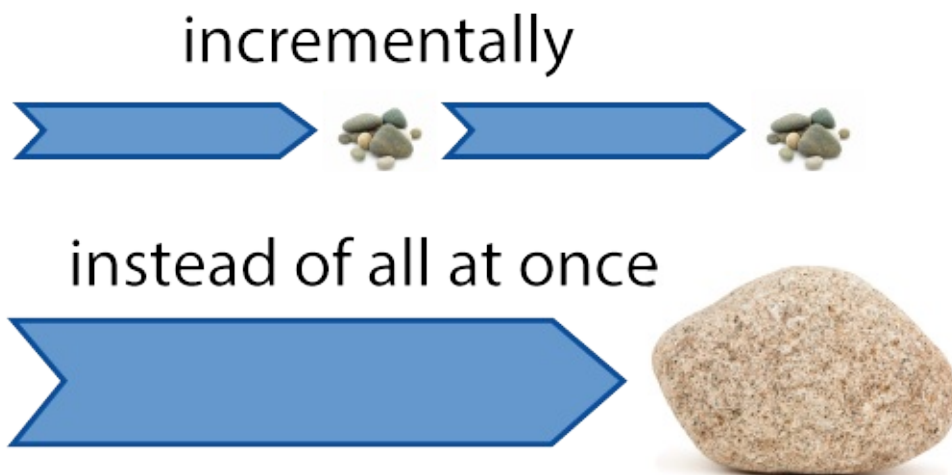
Just so everyone's on the same page I've decided to have public page where everyone can write anything (idea or complaint or question). Open the link [workflowy.com/s/E17I.aE8PQQOyYi](https://workflowy.com/s/E17I.aE8PQQOyYi)

## Course syllabus

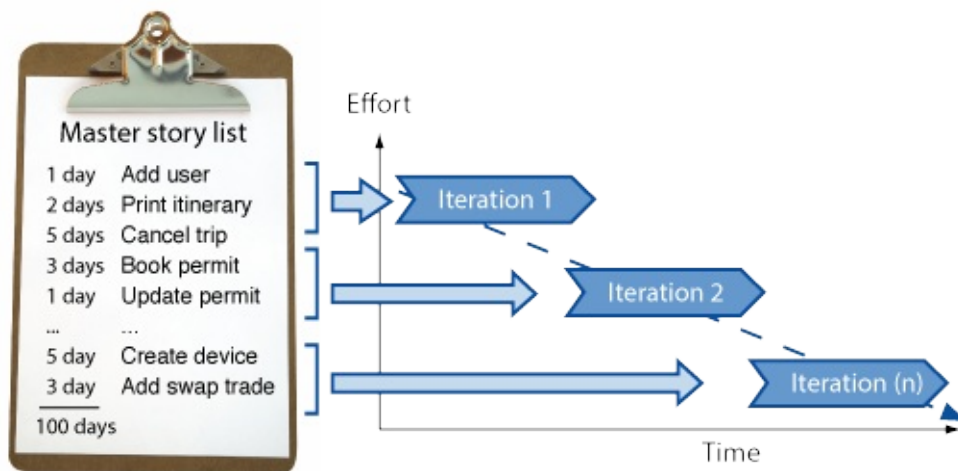
1. **Agile methodology, Scrum, Terminal commands, SSH, SVC Git.**
2. **Python programming language, TDD.**
3. **MVC and Django**
4. **REST API and Django Rest Framework**
5. **Start project, Postman**
6. **Applications: Authentication, User Profile, Teams, Scrum**
7. **Deployment, CI, Docker**
8. **Vue js (Frontend)**

# Agile methodology

**Agile** - is a time boxed, *iterative* approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver it *all at once* near the end.



It works by breaking projects down into little bits of user functionality called user stories, prioritizing them, and then continuously delivering them in short two week cycles called iterations.



## Scrum

Scrum is a subset of Agile. It is a lightweight process framework for agile development, and the most often used to manage complex software and product development, using iterative and incremental practices.

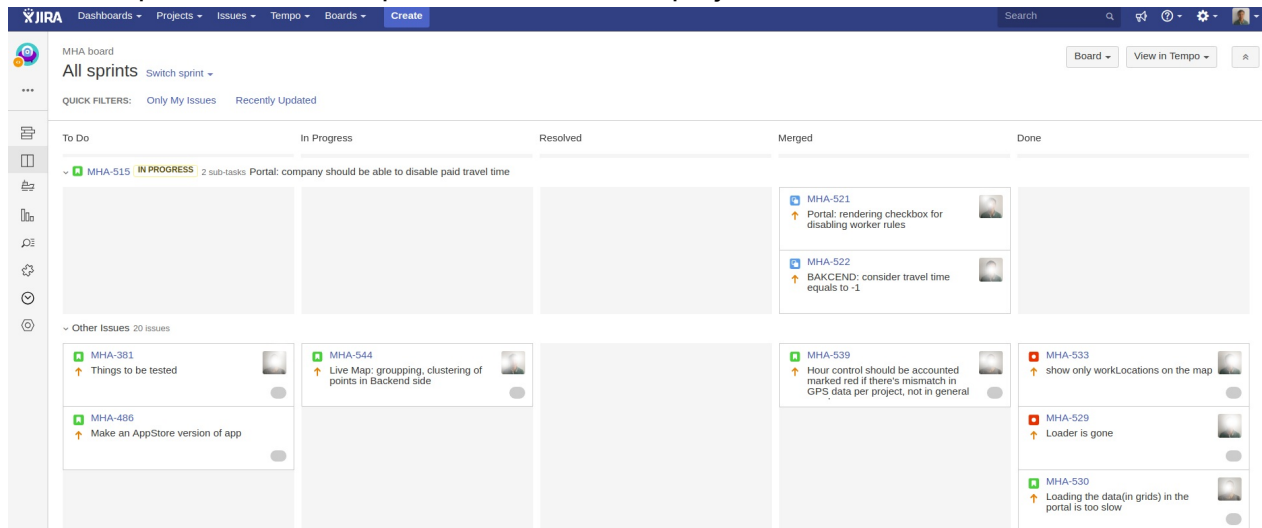
Scrum processes enable organizations to adjust smoothly to rapidly-changing requirements, and produce a product that meets evolving business goals. An agile Scrum process benefits the organization by helping it to

- Increase the quality of the deliverables
- Cope better with change (and expect the changes)
- Provide better estimates while spending less time creating them
- Be more in control of the project schedule and state

## SCRUM PROCESS



For example below is one sprint iteration for real project called MHA we used at work.

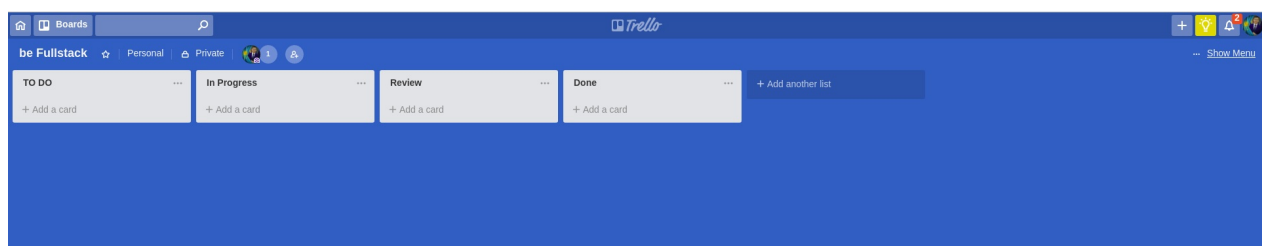


## Assignment

1. [Watch video on youtube.](#)
2. **Trello** - is a collaboration tool that organizes your projects into boards. In one glance, **Trello** tells you what's being worked on, who's working on what, and where something is in a process. So your task is:
  - i. Sign-up to **Trello**
  - ii. Create a board called "**be FullStak Developer**"
  - iii. Add list called "TO DO"
  - iv. Add list called "IN PROGRESS"
  - v. Add list called "REVIEW"
  - vi. Add list called "DONE"

Check that you have 4 list with titles: "TO DO", "IN PROGRESS", "REVIEW", "DONE"

It should look like this:



## How to use it

We are going to use Agile methodology & Scrum during learning this tutorial. One iteration (then call it '*sprint*') will take 2 weeks and this weeks have tasks initially put) you have to move all tasks (cards) from "**TO DO**" into "**DONE**".



## Operating System

I think that you have already known what is the [Linux](#) and I strongly recommend you to install [OS Ubuntu \(16.04, 18.04 LTS\)](#) or at least one of the Linux distribution. It's also easy to install Ubuntu alongside with Windows (see the picture below).

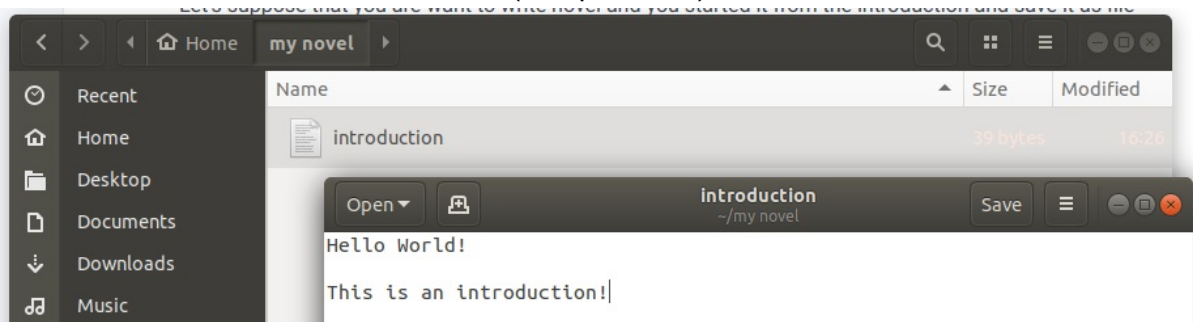


If you have Mac OS you have to learn how to work with terminal. Command are similar to Linux's commands.

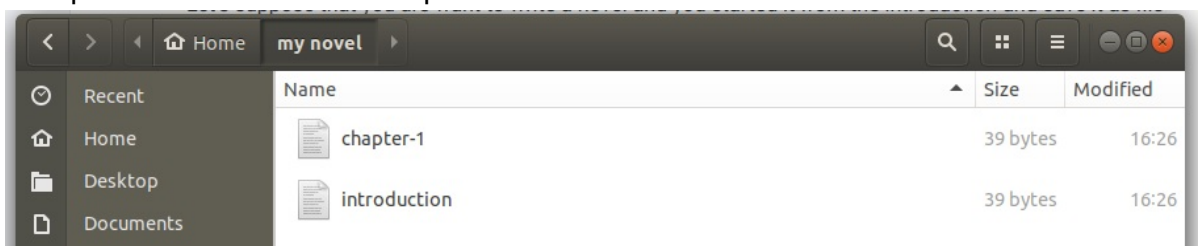
---

## Version Control System

1. Let's suppose that you are want to write a novel and you started it from the introduction and save it as file called introduction. (see picture 1)

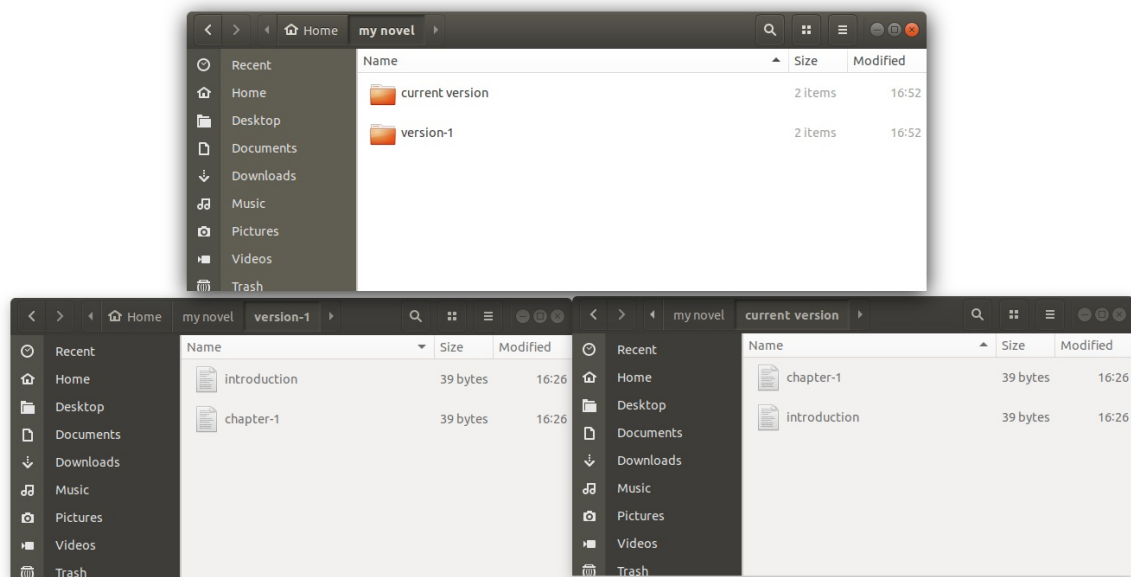


2. Then you decided to write the first chapter of your piece and after that the your workplace folder looks like the picture 2.



3. One day you decided to add something to the introduction and chapter-1, but don't want to loose the current version, because you are not sure that after changes it will be

better. To solve that problem you decided to save copy of the first version and start to do changes.



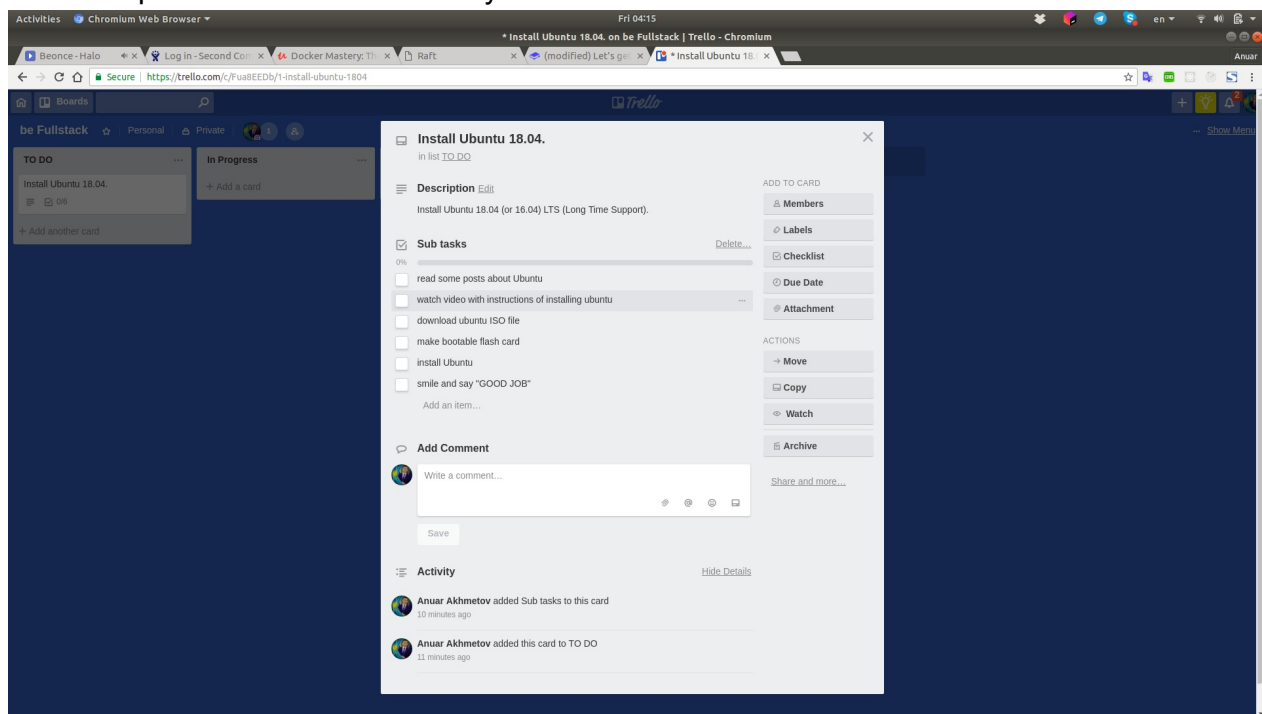
Then you add 20 chapters and for every changes save the copy of the previous version. It's really annoying to copy/paste files again and again, also the it take places in your storage. What if there is your partner who also helps you to write it? So as called it's name **version control system** is coming to help us where instead of copying files by hand you do it by one line command. It's also track only changed files and store differences, so it means that it save memory. Also by one command you can easily review the changes you've done. It's really great tool and integral part of the development. We will use most popular VCS called [GIT](#) which also was created by Linus [Linus Torvalds](#) (OS Linux).

---

## SPRINT 1

Assignment below has tasks for the first sprint (iteration 1). Read all of them and try to create cards (in Trello) inside **"TODO"** list. You can divide every task into small tasks. Just divide complex one into small parts and solve them one by one. Good luck!

For example this is a one of the way the first task can be divided:



## ASSIGNMENT

1. Install Ubuntu 18.04 (or 16.04) LTS (Long Time Support).
2. Read and play:
  - i. [Basic Linux command for beginners](#)
  - ii. [How to start using the Linux-terminal](#)
  - iii. Online game "[Bandit](#)"
3. Learn GIT
  - i. Cool course "[GIT REAL](#)" (50 minutes)
  - ii. Cool course -2 "[Learn GIT](#)"
  - iii. [Interactive game](#)
  - iv. read book "[PRO GIT -2](#)" (recommend)
4. Now you have a basic knowledge and some experience with git. Let's try to use it in the real world.
  - i. Create and verify account in [github](#).
  - ii. In github create repository with name "**test-repository**" (or anything you want).
  - iii. Open the terminal and clone your project to your PC.
  - iv. Add new any file and do commit
  - v. Push your changes into the repository
  - vi. Check your remote repository (github) and verify that last changes were pushed.

Cool. The first part is done and I have a question for you! Have you noticed that every time you want to do something with remote server (clone, push, pull) it asked you to enter the username & password? It's sucks that you have to do it every time, again and again. So to solve this problem SSH can help us.

```
1. Remove remote repository link from your local machine:
>>> git remote remove origin #it just removes url of your project from git2

2.Add new repository (origin) but now using ssh
git://github.com/your_username/repository_name # the first word is GIT instead of HTTP
S

3.Do any changes or create new file and commit it.
4.push your last changes to the remote repository.
```

You see it doesn't asked to enter username & password cause we used SSH. So great job!

---

## Expectation

After completing assignment I will expect that you:

1. Installed Ubuntu or bought Mac
  2. Can use basic terminal commands
  3. Know what is the "SSH"
  4. Can use GIT.
-

## Python is Popular

Python has been growing in popularity over the last few years. The 2018 [Stack Overflow Developer Survey](#) ranked Python as the 7th most popular and the number one most wanted technology of the year. [World-class software development countries around the globe use Python every single day.](#)

According to [research by Dice](#) Python is also one of the hottest skills to have and the most popular programming language in the world based on the [Popularity of Programming Language Index](#). Due to the popularity and widespread use of Python as a programming language, Python developers are sought after and paid well. If you'd like to dig deeper into [Python salary statistics and job opportunities](#), you can do so here.

---

## Python is Simple

Python code has a simple and clean structure that is easy to learn and easy to read. In fact, as you will see, the language definition enforces code structure that is easy to read. I want you have practice & understanding basic things from Python listed below before you go to the next section.

1. Basic syntax
2. Data types
3. Classes, methods.
4. Magic functions.

Looks very easy, but don't worry we will dive into the python during the development process.

---

## History

[Guido van Rossum](#) is best known as the author of the [Python programming language](#)



- "Benevolent Dictator For Life" for Python.
- Helped develop the [ABC programming language](#)
- In December 1989 had been looking for a "'hobby' programming project that would keep him occupied during the week around Christmas" as his office was closed.
- He attributes choosing the name "Python" to "being in a slightly irreverent mood (big fan of "Monty Python's Flying Circus)"

---

## Source of inspiration

Maybe you don't heard about the **ABC**, **Modula-3** programming languages, but python has borrowed some ideas from them and as we said before Van Rossum worked on developing ABC.

```
"ABC" programming language
HOW TO RETURN words document:
  PUT {} IN collection
  FOR line IN document:
    FOR word IN split line:
      IF word not.in collection:
        INSERT word IN collection
  RETURN collection

"MODULA-3" Programming langauge
TRY
  DO.SOMETHING()
EXCEPT
  IO.ERROR => IO.PUT("An I/O error occurred.")
END;
```

Wanted very simple, convenient, clear and useful language then we've got a Python. Look at the code below, even if you haven't worked with python you can easily understand what function does.

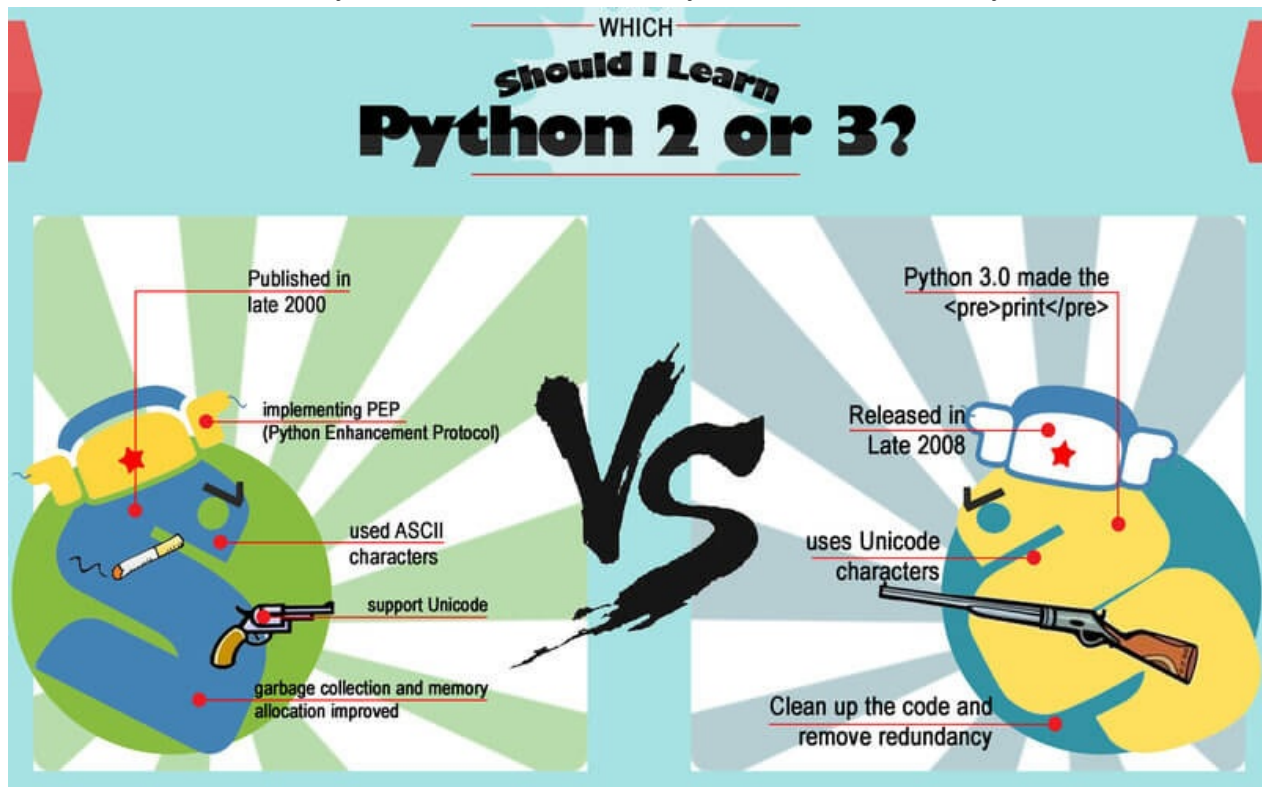
```
def magic(dir):
    acc = []
    for root, dirs, files in os.walk(dir):
        acc.extend(os.path.join(root, file) for file in files)
    return acc
```

Function ***magic*** get one argument which is a directory and get all ***root, directories, files*** on that directory. Build path absolute path for every file and add it to list. Easy! is not it?

---

## Versions

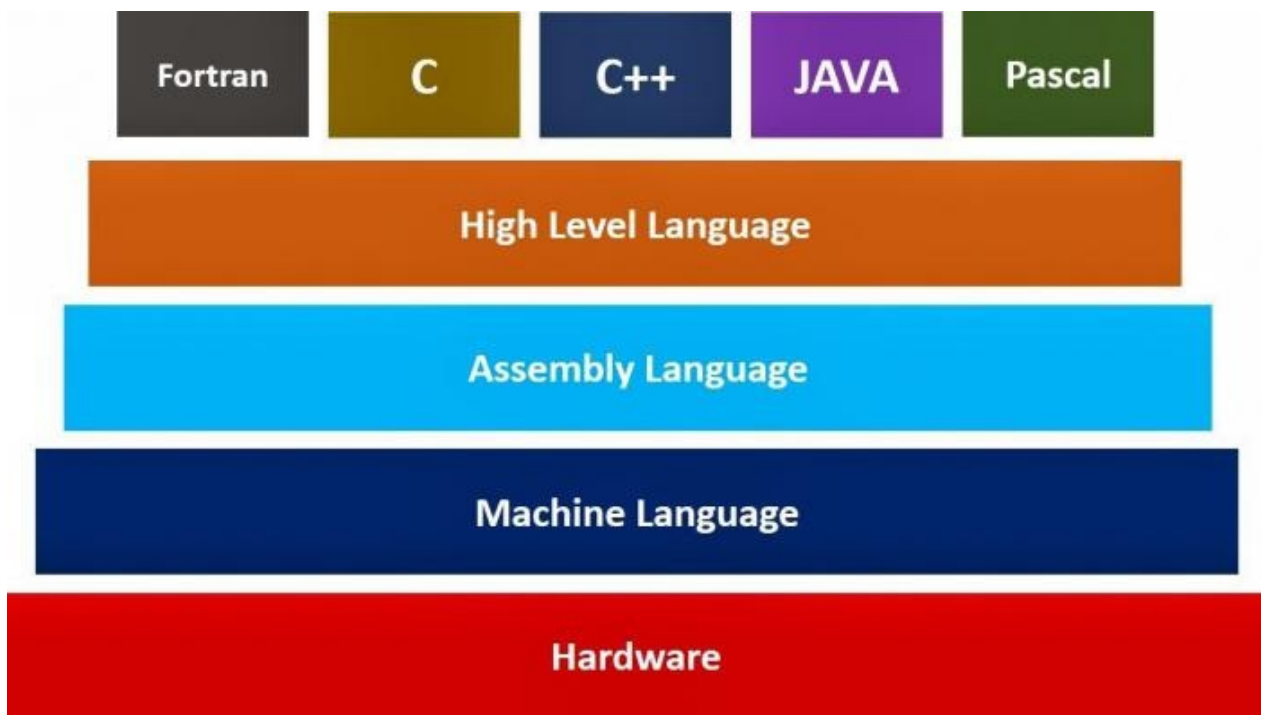
In python world there are two main version of Python (Python 2 and Python 3). Start use version 3 and don't worry about it, cause in our days it's stable and widely used version.



## Python - is a high level language

We don't care how does machine language code looks like also how does our program makes hardware work. We just write human readable, understandable code and run it. So that what is **High Level Programming Language** is. Python is a high level language that's why we can put it beside C++, Java, Pascal.



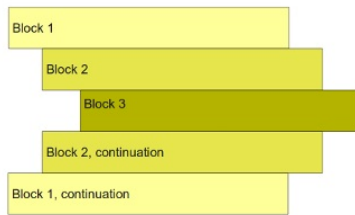


## Main ideas, Data types, conditional operators, loops.

**Module** - is the same as a code library. A file containing a set of functions you want to include in your application with extension **.py**. *For example let's suppose that we have `_hello.py` file then we can use it as a module (we import functions, objects which were declared inside that file)*

```
>>> import hello
>>> hello
<module 'hello' from '[...]/hello.py'>
>>> dir(hello)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
 '__name__', '__package__', '__spec__', 'message']
>>> hello.__name, hello.__file__
('hello', '[...]/hello.py')
>>> print(hello.message)
"Hello, world!"
```

**indentation** - Leading white space (spaces and tabs) at the beginning of a logical line is used to compute the indentation level of the line, which in turn is used to determine the grouping of statements. By conventions the indentation is equals to four white spaces.



```
>>> while True:
...     print(42)
      File "<stdin>", line 2
        print(42)
          ^
IndentationError: expected an indented block
```

**Object** - Everything is object.

```
>>> import hello
>>> type(hello)
<class 'module'>
>>> type(type(hello))
<class 'type'>
>>> type(type(type(hello)))
<class '<type'>
```

## Data types

### Strange

```
>>> None
>>> None == None
True
>>> None is None
True
```

### Logical

```
>>> to_be = False
>>> to_be or not to_be
True
>>> True or print(None) # lazy operator
True
>>> 42 + True
43
```

There are exist **Falsy & Truthy** values in python. All values which looks like False is Falsy value. For example

```
>>> my_string = ''          # empty string
>>> my_dict = {}            # empty dictionary (key/value pairs)
>>> my_list = []            # empty list
>>> my_set = set()          # empty set
>>> my_number = 0           # just zero
>>> my_tuple = ()           # empty tuple
>>> my_nothing = None       # null in other languages
>>> '''All values above will return False if we will check them as Boolean'''

>>> if my_string:
>>>     print("This will never prints because 'my_string' is Falsy value and return False")
>>>
```

## Operators

```
>>> 0 or 10 or 20 or 30
10
# or - is lazy operator, which means that it will return the first Truthy value.
# For above example: It checks 0 and found that it's Falsy value, then it goes
# to the next one and checks it. 10 is Truthy value, so it will return it.

>>> if 0 or 10 or 20 or 30:
>>>     print('This line will be printed, since the condition above is True')

# Question: What will be returned for expression below
>>> 0 or []

# Answer: it will return empty list, since it's the last checked element.
```

```
>>> 1 and 3 and [] and True
[]
# and - is also lazy operator, but it will return the first Falsy value.
# For above example: It checks 1, since it's Truthy value goes to the next one,
# checks 3, it's also Truthy value, so it goes to the [] and checks it.
# Found that empty list is Falsy value then return it

>>> if 1 and 3 and [] and True:
>>>     print('This line will never printed, since the condition above is False')

# Question: What will be returned for expression below
>>> 1 and 3 and 8

# Answer: it will return eight, the last checked element.
```

## Numbers

```
>>> 42      # integer
>>> 0.42     # float
>>> 42j      # complex

>>> 16/3
5.33333333333
>>> 16//3
5
>>> 16 % 3
1
```

## Strings & bytes

There is no difference with quotes to declaring string:

```
>>> single_quote_string = 'Hello world'
>>> double_quote_string = "Hello world 2"
>>> triple_quote_string = '''Hello world 3'''
>>> triple_doubled_quote_string = """Hello world 4"""
```

**bytes** - is a sequence consisted of 8-bit values

**string** - is a sequence consisted of `__**unicode\__` symbols.

```
>>> b'foo' # bytes
b'foo'
>>> b'foo'.decode("utf-8")
'foo'
>>> bar = "bar"
>>> len(bar)
3
>>> s = '.' * 10
'.....'
>>> "  foo bar  ".strip()
'foo bar'
```

## Lists

```
>>> [] # empty list
>>> my_list = list()
>>> [0] * 4
[0, 0, 0, 0]
>>> my_list = [1, 2, 3, 4]
>>> len(my_list)
4
>>> my_list[1]
2
>>> my_list[0] = 10
>>> my_list
[10, 2, 3, 4]
>>> my_list.append(42)
[10, 2, 3, 4, 42]
>>> del my_list[0] # or my_list.pop(0)
[2, 3, 4, 42]
```

## Concatenation and slices

```
>>> xs = [1, 2, 3, 4]
>>> xs + [5, 6]
[1, 2, 3, 4, 5, 6]

>>> ", ".join(['foo', 'bar'])
'foo, bar'

>>> s = 'foobar'
>>> xs[:2]
[1, 2]
>>> s[:2]
'fo'

>>> xs[1:3]
[2, 3]
>>> s[1:3]
'oo'

>>> xs[0:4:2]
[1, 3]
>>> s[0:4:2]
'fo'

>>> xs[:]
[1, 2, 3, 4]
>>> s[:]
'foobar'
```

## Tuple, Set, Dictionary

Tuple is almost the same data type as a list, but it's immutable and for its declaration we use '()

```
>>> tuple()
()
>>> date = ('year', 2018)
>>> date[1] = 2019
Traceback (most recent call last):
  File "stdin", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> del date[0]
Traceback (most recent call last):
  File "stdin", line 1, in <module>
TypeError: 'tuple' object does not support item deletion
```

Set stores unique values

```
>>> set() # hash-set
set()
>>> xs = {1, 2, 3, 4}
>>> 42 in xs
False
>>> 42 not in xs
True
>>> xs.add(42) # {1, 2, 3, 4, 42}
>>> xs.discard(42) # {1, 2, 3, 4}

# Operators IN, NOT IN works for all containers
>>> 42 in [1, 2, 3, 4]
False
>>> 'month' not in ('year', 2018)
True

>>> xs = {1, 2, 3, 4}
>>> ys = {4, 5}
>>> xs.intersection(ys) # or (xs & ys)
{4}
>>> xs.union(ys) # or (xs | ys)
{1, 2, 3, 4, 5}
>>> xs.difference(ys) # or (xs - ys)
{1, 2, 3}
```

Dictionary is a key/value table

```
>>> {} # or dict(), hash-table
>>> date = {'year': 2018, 'month': 'September'}
>>> date['year']
2018
>>> date.get('day', 7) # if there is no key 'day' return default value 7
7
>>> date.keys()
dict_keys(['month', 'year'])
>>> date.values()
dict_values(['September', 2018])
```

## Conditional operators & loops

```
>>> x = 42
>>> if x % 5 == 0:
>>>     print('fizz')
>>> elif x % 3 == 0:
>>>     print('buzz')
>>> else:
>>>     pass # do nothing
'buzz'

>>> "even" if x % 2 == 0 else 'odd' # ternary operator
'even'
```

```
>>> i = 0
>>> while i < 10:
>>>     i += 1
>>> i
10

>>> sum = 0
>>> for i in range(1, 5):
>>>     sum += i * i
>>> sum
30
```

## Sprint 2

Learn python.

---

## Assignment

1. "[Learn python](#)" - practice course
  2. "[solo learn python](#)" - online course
  3. Try to read [python in Habr](#)
  4. [Python and basic TDD task](#)
  5. Choose python in [Exercism](#) and try to solve the problem as much as possible
- 

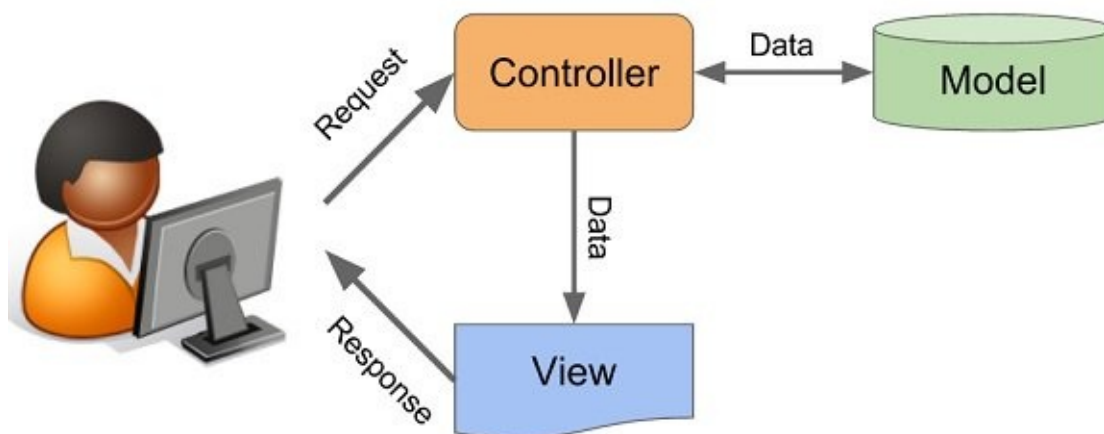
## Expectation

You are comfortable to work with python.



# MVC & DJANGO

MVC has been around as a concept for a long time, but has seen exponential growth since the advent of the Internet because it is the best way to design client-server applications. All of the best web frameworks are built around the MVC concept. As a concept, the MVC design pattern is really simple to understand:



- The **model(M)** is a model or representation of your data. The model allows you to pull data from your database without knowing the intricacies of the underlying database.
- The **view(V)** is what you see. It's the presentation layer for your model (in the browser for a Web app, or the UI for a desktop app).
- The **controller(C)** controls the flow of information between the model and the view. It uses programmed logic to decide what information is pulled from the database via the model and what information is passed to the view.

---

## Sprint 3

It's one of the difficult sprint for you, cause you have to cover two tutorials. The first one is very useful and described very well and I hope that you will get some experience from that.

---

## Assignment

1. There is already exists a great tutorial called "[Django Girls](#)". So I strongly recommend you to follow it and try to understand.

2. [Django Official tutorial](#).
- 

## Expectation

1. You now how to django application and understand the flow
2. Have known:
  - i. Models
  - ii. Migrations
  - iii. URL resolvers
  - iv. Views
  - v. Forms

# API & JSON

When you use an application on your mobile phone, the application connects to the Internet and sends data to a server. The server then retrieves that data, interprets it, performs the necessary actions and sends it back to your phone. The application then interprets that data and presents you with the information you wanted in a readable way. This is what an API is - all of this happens via [API](#).

To explain this better, let us take a familiar example.

Imagine you're sitting at a table in a restaurant with a menu of choices to order from. The kitchen is the part of the "system" that will prepare your order. What is missing is the critical link to communicate your order to the kitchen and deliver your food back to your table. That's where the waiter or API comes in. The waiter is the messenger – or API – that takes your request or order and tells the kitchen – the system – what to do. Then the waiter delivers the response back to you; in this case, it is the food.

[JSON](#) - (**JavaScript Object Notation**) is a way to store information in an organized, easy-to-access manner. It gives us a human-readable collection of data that we can access in a really logical manner.

For example:

```
1  {  
2  "project": "gitGile",  
3  "description": "Here is a description",  
4  "something": "some data"  
5  }  
6
```

---

## REST API & Django Rest Framework

REST ([Representational state transfer](#)) which essentially refers to a style of web architecture that has many underlying characteristics and governs the behavior of clients and servers. It lets you interact with Parse from anything that can send an HTTP request. See [Parse REST Definition](#) for examples of this.

Traditionally, Django is known to many developers as a MVC Web Framework, but it can also be used to build a backend, which in this case is an API [Django REST framework](#) is a powerful and flexible toolkit for building Web API.

---

## Sprint 4

It's one of the difficult sprint for you, cause you have to cover two tutorials. The first one is very useful and described very well and I hope that you will get some experience from that.

---

## Assignment

1. Django Rest Framework [official tutorial](#).
- 

## Expectation

1. You now how to DRF application and understand the flow
2. Have known:
  - i. API & REST API
  - ii. Models
  - iii. Migrataion
  - iv. Serializers
  - v. Routers

## JetBrains Toolbox

[Toolbox](#). “All Products Pack” gives you access to over a dozen developer tools included under the JetBrains Toolbox. You decide what to put in your toolbox for your next project. If you are student or teacher and have educational email account (name@sdu.edu.kz) then you can use educational version which is also include a lot of feature related to the Community edition. Just search how to get educational licence key and do it.

---

## IDEA

[PyCharm](#) - is one of the best IDEA for python developer created by [JetBrains](#).

---

## Terminal

- For Ubuntu users recommend install **Guake**
  - For Mac OS users recommend install **Itterm2**
  - For Windows users recommend install Ubuntu or buy Mac.
-

## Never reinvent the wheel

I hope you have already heard this phrase and can guess what does it mean. So let's try to do it, [just do it](#). Suppose you have a task to add new feature to the project. Write down your steps of actions to complete that task.

Here is mine:

1. Think about new feature and image it's workflow
  2. Googling it, maybe other developers have done and delivered it
  3. Try to find libraries and play with that (maybe it's suitable for you)
  4. Make decision (use library? ) and start to code
  5. Finish task and be happy
- 

## Don't repeat yourself

[DRY]([https://en.wikipedia.org/wiki/Don't\\_repeat\\_yourself](https://en.wikipedia.org/wiki/Don't_repeat_yourself)) - principle of software development.

There are a thousands of Django project developed around the world by developers. Almost every project has been developed fundamentals in the common way. Create django project, create django application for authentication etc. What do you think maybe there is already exists boilerplate for django proeject?

Try to find it.

---

## How to search

[Django Packages](#) - is a directory of reusable apps, sites, tools, and more for your Django projects. If you have found some library before using it try to find it in django-packages and analyze it. (how many starts, commits, tests status etc.) After that you will answer the question "Is this library suitable for my project?".

---

## Cookie-cutter

Recommend boilerplate for creating django project: [django-cookiecutter](#).

```
>>> if you_have_found_it: # django-cookiecutter
>>>     good_job = True
>>> else:
>>>     good_job = False
```

Please before creating the project read the documentation. I recommend you always read the documentation cause by this way you can save your time. Don't underestimate it.

---

## cookiecutter-django-rest

Recommend boilerplate for creating django rest framework project: [cookiecutter-django-rest](#)

```
>>> if django_project:
>>>     use_cookiecutter_django = True
>>> else:
>>>     use_cookiecutter_django_rest = True
```

## Prepare

Since we will have REST API we are gonna use [cookiecutter-django-rest](#). Repository and you will find that for creating django-project you have to run two commands:

```
# install cookiecutter
$ pip3 install cookiecutter
# run cookiecutter-django-rest
$ cookiecutter gh:agconti/cookiecutter-django-rest
```

Answer the prompts with your own desired options. Here is my example: Then cookiecutter will generate project (new directory with project name)

```
an4lk@an4lk-laptop:~/Study/be-Fullstack$ cookiecutter gh:agconti/cookiecutter-django-rest
You've downloaded /home/an4lk/.cookiecutters/cookiecutter-django-rest before. Is it okay to delete and re-download it? [yes]: yes
github_repository_name [piedpiper-web]: gitgile
app_name [piedpiper]: gitGile
email [richard.hendriks@piedpiper.com]: akhmetov.anuar09@gmail.com
description [Its all about a Weissman score > 5.0]:
github_username [The owner of the repository. Either your github username or organization name.]: An4lk
an4lk@an4lk-laptop:~/Study/be-Fullstack$
```

---

## Database

We will use postgresql as database and the first thing you have to do is install it.

```
$ sudo apt-get update
$ sudo apt-get install postgresql postgresql-contrib
```

Since you have installed it then create database and role.



```
# login as a super user postgres
$ sudo su - postgres
# enter psql promtp
$ psql

postgres=# create database gitgile;
CREATE DATABASE

postgres=# create role gitgile with password 'gitgile';
CREATE ROLE

postgres=# grant all privileges on database gitgile to gitgile;
GRANT

# Enable your user to login
postgres=# ALTER ROLE "gitgile" WITH LOGIN;

# exit psql promt
postgres=# \q

# logout
postgres@an4ik-laptop:~$ exit
```

It's recommend to have the same name for database, role, password. It's not a vulnerability since it's local deployment.

---

## Virtual environment

You have already known about virtual environment (you had to use it in django tutorials). There is convention about naming virtual environment **.venv** or **venv** (observation from virtual environment). So let's create virtual environment.

```
$ python3 -m virtualenv .venv
```

```
an4ik@an4ik-laptop:~/Study/be-Fullstack/gitgile$ python3 -m virtualenv .venv
Using base prefix '/usr'
New python executable in /home/an4ik/Study/be-Fullstack/gitgile/.venv/bin/python3
Also creating executable in /home/an4ik/Study/be-Fullstack/gitgile/.venv/bin/python
Installing setuptools, pip, wheel...done.
```

Then activate it and install required libraries

```
# activate your virtual environment
$ source .venv/bin/activate
# after activating virtual environment you can see it in the left side, shell starts with (.venv)

# install required libraries to your virtual environment
$ pip3 install -r requirements.txt
```

```
an4ik@an4ik-laptop:~/Study/be-Fullstack/gitgile$ source .venv/bin/activate
(.venv) an4ik@an4ik-laptop:~/Study/be-Fullstack/gitgile$ pip3 install -r requirements.txt
Collecting pytz==2018.5 (from -r requirements.txt (line 2))
```

## Configuration

We have created database named '**gitgile**', role '**gitgile**' with password '**gitgile**', so let's set them in project configuration. Look at the file gitGile/config/common.py. You have to change it.

```
53 # Postgres
54 DATABASES = {
55     'default': dj_database_url.config(
56         default='postgres://role:password@localhost:5432/database',
57         conn_max_age=int(os.getenv('POSTGRES_CONN_MAX_AGE', 600))
58     )
59 }
```

Our used boilerplate cookicutter-django-rest created the project without **SECRET\_KEY**, so do it now. Here is my example

```
40
41 ALLOWED_HOSTS = ["*"]
42 ROOT_URLCONF = 'gitGile.urls'
43 SECRET_KEY = os.getenv('DJANGO_SECRET_KEY', 'just_any_secret_key')
44 WSGI_APPLICATION = 'gitGile.wsgi.application'
45
```

## Migration

What is a migration? Read the [django documentation](#) and after that do migration

```
(.venv) an4ik@an4ik-laptop:~/Study/be-Fullstack/gitgile$ ./manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, authtoken, contenttypes, sessions, users
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0001_initial... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying users.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying authtoken.0001_initial... OK
  Applying authtoken.0002_auto_20160226_1747... OK
  Applying sessions.0001_initial... OK
  Applying users.0002_auto_20171227_2246... OK
(.venv) an4ik@an4ik-laptop:~/Study/be-Fullstack/gitgile$
```

Again, it's applied all models changes to our database. Now we have all required schemas, tables etc.

---

## Run application

```
# Run our application locally
$ ./manage.py runserver
```

```
(.venv) an4ik@an4ik-laptop:~/Study/be-Fullstack/gitgile$ ./manage.py runserver
django-configurations version 2.1, using configuration 'Local'
Performing system checks...

System check identified no issues (0 silenced).
September 15, 2018 - 06:00:55
Django version 2.1.1, using settings 'gitGile.config'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Now you can see it via your browser using localhost:8000.

# Initial commit and pushing to the repository

It's time to create repository in your github account, do initial commit and push it.

1. Create github repository
2. Commit and push your local changes

This is my [initial commit](#)





---

## Configuring Virtualenv Environment in PyCharm

Set virtual environment to your IDEA. Follow instructions from [JetBrains documentation](#).

PyCharm makes it possible to use the [virtualenv](#) tool to create a project-specific isolated *virtual environment*. The main purpose of virtual environments is to manage settings and dependencies of a particular project regardless of other Python projects. **virtualenv** tool comes bundled with PyCharm, so the user doesn't need to install it.

### To create a virtual environment

1. In the **Settings/Preferences** dialog (`Ctrl+Alt+S`), select **Project: <project name> | Project Interpreter**.
2. In the [Project Interpreter](#) page, click  and select **Add**.
3. In the left-hand pane of the **Add Python Interpreter** dialog box, select **Virtualenv Environment**. The following actions depend on whether the virtual environment existed before.  
If **New environment** is selected:
  1. Specify the location of the new virtual environment in the text field, or click  and find location in your file system. Note that the directory where the new virtual environment should be located, must be empty!
  2. Choose the base interpreter from the drop-down list, or click  and find the base interpreter in the your file system.
  3. Select the **Inherit global site-packages** check-box if you want to inherit your global site-packages directory. This check-box corresponds to the `--system-site-packages` option of the [virtualenv](#) tool.
  4. Select the **Make available to all projects** check-box, if needed.  
If **Existing environment** is selected:
  1. Specify the required interpreter: use the drop-down list, or click  and find one in your file system.
  2. Select the check-box **Make available to all projects**, if needed.
4. Click **OK** to complete the task.

# Project

**GitGile** - is a system which helps for every member of software team to plan, track and release software using Agile methodology. I also planning to integrate with [gitLab](#).

## Functionalities

1. **Users.** Description
2. **Projects.** Description
3. **Tasks.** Description
4. **Sprints.** Description
5. **Integrating with GitLab.** Description

# JWT authentication

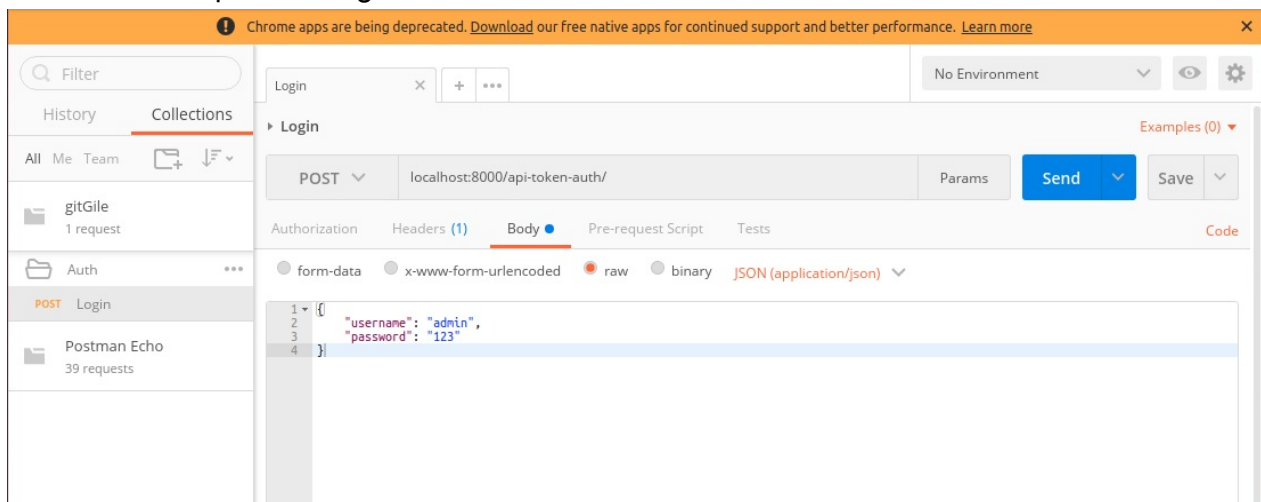
Our cookicutter-django-rest used default Token Based authentication, but in our days JWT is more secure and popular that's why we are gonna use it. There is a library called [django-rest-framework-jwt](#). So try to use it. Steps you have to do:

1. Read the [documentation](#) for that library
2. Use and check it.
3. Compare with my [commit #2](#)

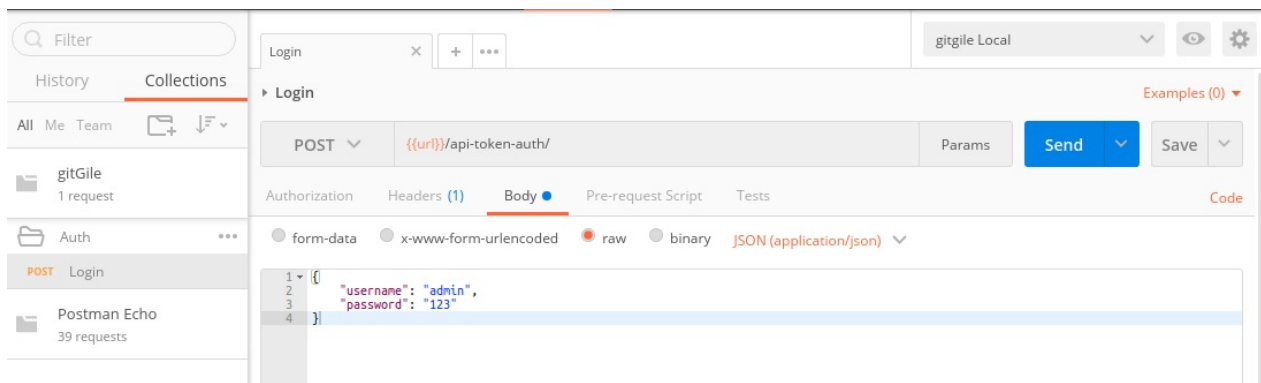
## POSTMAN helps us

PostMan is great tool which helps us to build API.

Recommend you to create folder for every application. For now create Auth folder and save there POST request for login.



Postman also has own environment, it's very helpful to check between local, test server. As you see I've stored **url=localhost:8000** in gitGile local environment.



## Reponse structure

We are building REST API for web and mobile applications, so that's why our response structure should be simple and comfortable for all of them. Our Response JSON structure looks like this:

```
{
  "success": false,
  "data": {},
  "errors": {
    "detail": "Given data is not correct"
  }
}

{
  "success": true,
  "data": "given data",
  "errors": {}
}
```

## Let's do it

The first thing you have to is research. Look at the documentaion and try to find something about exception.

If you found [this one](#) good job. Read it carefully.

In order to alter the style of the response, you could write the following custom exception handler:

```
from rest_framework.views import exception_handler

def custom_exception_handler(exc, context):
    # Call REST framework's default exception handler first,
    # to get the standard error response.
    response = exception_handler(exc, context)

    # Now add the HTTP status code to the response.
    if response is not None:
        response.data['status_code'] = response.status_code

    return response
```

Try do it. Your response structure should be like below. Then you can check it with my [commit #3](#)

```
{
  "data": {},
  "success": false,
  "errors": {
    "username": [
      "A user with that username already exists."
    ]
  }
}
```

Now let's try to create user with any username and password via postman. The response is like this:

```
{
  "id": "3281dd3b-da56-42ab-8b22-1b6c7a39b321",
  "username": "user4",
  "first_name": "",
  "last_name": "",
  "email": "",
  "auth_token": "5c81ecdc1c1b697c1f6902032f929d52a24ea50e"
}
```

This isn't what we want. Change it. But before let's analyze ***UserCreateViewSet*** \_which is return response. We know that it is inherited from ***mixins.CreateModelMixin*** and calls it's ***create()***\_ function. Look at it:



```

class CreateModelMixin(object):
    """
    Create a model instance.
    """
    def create(self, request, *args, **kwargs):
        serializer = self.get_serializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        self.perform_create(serializer)
        headers = self.get_success_headers(serializer.data)
        return Response(serializer.data, status=status.HTTP_201_CREATED, headers=headers)

    def perform_create(self, serializer):
        serializer.save()

    def get_success_headers(self, data):
        try:
            return {'Location': str(data[api_settings.URL_FIELD_NAME])}
        except (TypeError, KeyError):
            return {}

```

The important thing is the last line of create function. it returns ***serializer.data*** which means that we have to create custom ModelSerializer class and override data property. All model related serializers will inheritate from that class. Good luck!

It's [commit #4](#)

```

{
  "errors": {},
  "success": true,
  "data": {
    "id": "955a718b-f5c0-4750-9984-235484c9f7dd",
    "username": "user5",
    "first_name": "",
    "last_name": "",
    "email": "",
    "auth_token": "8bed729e6a1eeebaca93036670e774441d29e775"
  }
}

```

But wait what about other serializers? I think we will also use other serializers (not only ModelSerializer) so let it be more dynamic ([commit #5](#))

```

class DefaultModelSerializer(BaseSerializer, ModelSerializer):
    pass

```

Look at the code. It says that **DefaultModelSerializer** inherited from (first) **BaseSerializer** then (second) **ModelSerializer**. It's very useful to know the workflow of inheritance in python. So read about it, hope you will find and learn about [C3 linearization](#).

---

## Fix authentication response

If you do POST request to localhost:8000/api-token-auth/ the response will be like this:

```
{
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2V..."
}
```

It's wrong cause we said that our responses will have other structure. So let's fix it. Read the [documentation](#) again and try to find something useful.

Read about **\_JWT\_RESPONSE\_PAYLOAD\_HANDLER\_** in additional-settings section. Then try to do fix authentication response by yourself. Then check with my version of [commit #6](#).

There is another problem when you try to login with invalid data, since response will be like this:

```
{
  "non_field_errors": [
    "Unable to log in with provided credentials."
  ]
}
```

So to solve this problem look at the **urls.py** file find which view is handles **/api-token-auth** POST request. You will find **ObtainJSONWebToken** inherited from **JSONWebTokenAPIView** and **JSONWebTokenAPIView** has post function, which is called for our POST request. Try to find something can help you:

```
def post(self, request, *args, **kwargs):
    serializer = self.get_serializer(data=request.data)

    if serializer.is_valid():      # THIS IS WHAT WE ARE LOOKING FOR
        user = serializer.object.get('user') or request.user
        token = serializer.object.get('token')
        response_data = jwt_response_payload_handler(token, user, request)
        response = Response(response_data)
        if api_settings.JWT_AUTH_COOKIE:
            expiration = (datetime.utcnow() +
                          api_settings.JWT_EXPIRATION_DELTA)
            response.set_cookie(api_settings.JWT_AUTH_COOKIE,
                              token,
                              expires=expiration,
                              httponly=True)

        return response

    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

We have found that it calls **serializer.is\_valid()** function. So let's change it. Here is my [commit #7](#).

But wait look at the response carefully

```
{
  "data": {},
  "success": false,
  "errors": {
    "non_field_errors": [
      "Unable to log in with provided credentials."
    ]
  }
}
```

Hope that you have remarked that for some invalid data error message put into **\_non\_field\_errors** field. It's not following our convention, so try to fix it. (hint read documentation for DRF).

Hope you have found [this](#). So just adding one line of code fixes our problem: [commit #8](#)

---

## Authentication problem

Now think about a login POST request. What does it return for valid username & password ?

The answer is token, but I think it should be better if it returns user data (username, email, last\_name etc.) So let's do it. From previous experience you have to know that the first thing is read documentation and try to find something [useful to solve problem](#).

(JWT\_RESPONSE\_PAYLOAD\_HANDLER). Good luck! ([commit #9](#))

---

## Authentication required

Since we use Token authentication every request has to include token in it's header. But our users request allows to do everything without authentication. So let's fix it. The first thing is reading documentation and try to find [something usefull](#).

You have found that DEFAULT\_PERMISSION\_CLASSES has set correct. So what's the problem?

```
REST_FRAMEWORK = {  
    # ...  
    'DEFAULT_PERMISSION_CLASSES': [  
        'rest_framework.permissions.IsAuthenticated',  
    ],  
    # ...  
}
```

Then look at the views.py file in users application. You see that every views has own permission classes.

```
permission_classes = (IsUserOrReadOnly,)  
permission_classes = (AllowAny,)
```

To solve it just remove them. [commit #10](#)

---

## Avoid copy/paste in POSTMAN

Our application will requires authentication token for every request. So it's not easy to copy/paste everytime you change remove or change current user. Fortunately we can fix it in POSTMAN.















