Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering

Bachelor Thesis

Development of bug-tracking system as web-application

Georgiy Vasyliev

BACHELOR THESIS ASSIGNMENT

Georgiy Vasyliev

Informatics

Thesis title

Development of bug-tracking system as web-application

Objectives of thesis

The main objective of the thesis is to design and implement a bug-tracking system in the form of a web application. The secondary objective is to describe all the necessary technologies and methods.

Methodology

The thesis consists of two parts – theoretical and practical. The theoretical part of the thesis is based on a study and review of various information sources. Based on the synthesis of the gained knowledge the groundwork for the practical part is defined.

The practical part of the thesis consists of design and implementation of a web application for bug-tracking purposes. The application will be deployed, tested and based on the experience from its development and testing feedback the conclusion will be formulated and possible future development option will be outlined. The standard means and tools of the software engineering will be utilized during the process.

The proposed extent of the thesis

35-40 pages

Keywords

Bug-Tracker, Web-Application, Python, Django

Recommended information sources

Anquetil, Roxane. Fundamental Concepts For Web Development: HTML5, CSS3, JavaScript and much more!. Webstreet Learning, 2019. ISBN 9781702250382

Martelli, Alex, Anna Ravenscroft, and David Ascher. Python cookbook. " O'Reilly Media, Inc.", 2005. ISBN 978-1-449-34037-7.

Melé, Antonio. Django 3 By Example: Build powerful and reliable Python web applications from scratch. Packt Publishing Ltd, 2020. ISBN 978-1-83898-195-2.

Expected date of thesis defence

2021/22 SS – FEM

The Bachelor Thesis Supervisor

Ing. Jiří Brožek, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 1. 11. 2021	Electronic approval: 23. 11. 2021
Ing. Martin Pelikán, Ph.D.	**Ing. Martin Pelikán, Ph.D.**
Head of department	Dean

Prague on 14. 03. 2023

Declaration

I declare that I have worked on my bachelor thesis titled "Development of bug-tracking system as web-application" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break any copyrights.

In Prague on 14.03.2023 ____________________________

Acknowledgement

I would like to thank Ing. Jiří Brožek, Ph.D., for advice and support during my work on this thesis.

Development of bug-tracking system as web-application

Abstract

The following thesis aims to develop a working bug-tracking project in the Django web framework. It intends to explore web development-related topics, including an overview of the web development field, the Django framework, website architecture design, and the bug-tracking process, and explain all steps of developing an app. This thesis goes through different stages of development, including setting up a start-up project, applying installations and configurations to the project, working in the app folder and making use of other folders and files, creating a database for a web server, and eventually deploying the project to a hosting platform. All of the above will be described and analyzed in the practical part of the thesis. The resulting web app will be working and hosted project that will be fully accessible for usage and open for future development.

Keywords: Bug-Tracker, Web-application, Python, Django, Programming, Backend, Frontend, Web-development

Vývoj bug-tracking systému jako webové aplikace

Abstrakt

Následující práce si klade za cíl vyvinout fungující projekt sledování chyb ve webovém frameworku Django. Má v úmyslu prozkoumat témata související s vývojem webu, včetně přehledu oblasti vývoje webu, rámce Django, návrhu architektury webu a procesu sledování chyb a vysvětlit všechny kroky vývoje aplikace. Tato práce prochází různými fázemi vývoje, včetně nastavení start-up projektu, aplikování instalací a konfigurací na projekt, práce ve složce app a využívání dalších složek a souborů, vytvoření databáze pro webový server, popř. nasazení projektu na hostingovou platformu. Vše výše uvedené bude popsáno a analyzováno v praktické části práce. Výsledná webová aplikace bude funkční a hostovaný projekt, který bude plně přístupný pro použití a otevřený pro budoucí vývoj.

Klíčová slova: Bug-Tracker, Web-Aplikace, Python, Django, Programování, Backend, Frontend, Web-development

Table of content

List of pictures

1 Introduction

Nowadays, there are a lot of professions in the IT sphere that have a different impact on building our bright future. But there are a lot of obstacles in our path to getting to the future we want. Because the world is not perfect and there are a lot of errors we have to fix.

These errors in computer programs are so-called "bugs," which were named after a moth trapped in a computing machine's relay. So, people invented such things as testing them. But for that, we need to somehow manage them systematically. For that reason, we have well-known bug-tracking systems. They serve as a testing tool to give information about a problem, prioritize it, and eliminate it.

Today, we can find many examples of bug-tracking or issue-tracking systems on the Internet. They all vary in different aspects like interface, functionality, and cost. But what if you do not want to pay or do not like the interface or functionality of any of them? In this situation, the whole team decides to create their own. But what if you want to do the first project that will show you all the nuances of web development and be helpful in the future? Then you are in the right place, because this thesis will show you what it is and how to build it.

2 Objectives and Methodology

2.1 Objectives

The main objective of the thesis is to design and implement a bug-tracking system in the form of a web application using the Django framework. The secondary objective is to:

- describe all necessary technologies and methods
- explain all necessary concepts of this project
- overview all necessary tools

2.2 Methodology

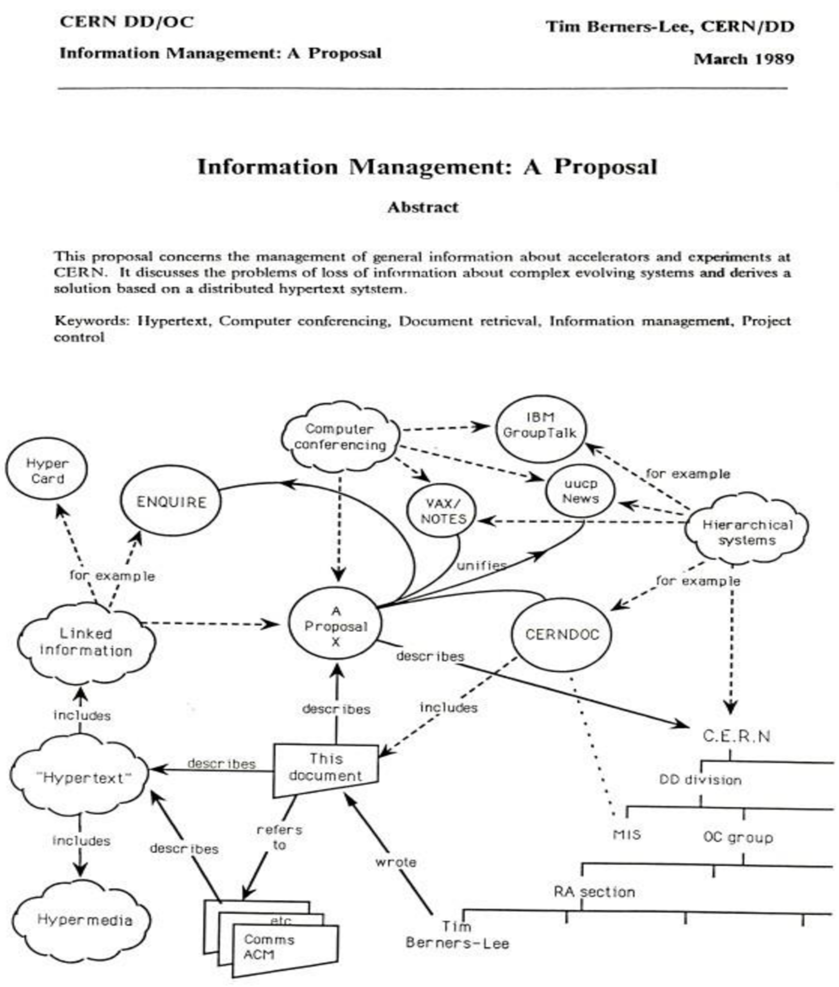
The thesis consists of two parts – theoretical and practical. The theoretical part of the thesis is based on a study and review of various information sources. Based on synthesizing the gained knowledge, the groundwork for the practical aspect is defined.

The thesis's practical part consists of designing and implementing a web application for bug-tracking purposes. The application will be deployed and tested using the bug-tracking method. Based on the experience from its development and testing feedback, a conclusion will be formulated, and possible future development options will be outlined. The standard means and technologies of software engineering, like back-end and front-end languages, web hosting services, version control software, software frameworks, developer tools, virtual environments, and an Integrated Development Environment, will be utilized during the process.

3 Literature Review

3.1 Brief history of web

The history of the web begins with the creation of the first network for sharing information by Tim Berners-Lee, a British scientist working at CERN. It was invented in 1989 and called the World Wide Web (WWW). To satisfy the need for automated information-sharing amongst scientists in universities and institutes around the world, the Web was initially designed and developed. The main idea was to create a global information system through a synthesis of evolving technologies. Tim Berners-Lee wrote the first proposal for the World Wide Web in March 1989 and his second proposal in May 1990 (A short history of the web, n.d.).



CERN DD/OC — **Tim Berners-Lee, CERN/DD**

Information Management: A Proposal — **March 1989**

Information Management: A Proposal

Abstract

This proposal concerns the management of general information about accelerators and experiments at CERN. It discusses the problems of loss of information about complex evolving systems and derives a solution based on a distributed hypertext sytstem.

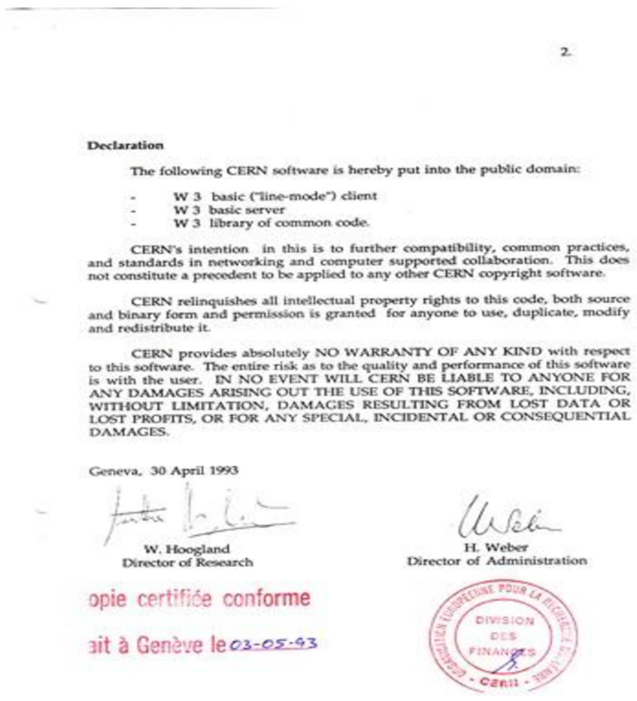
Keywords: Hypertext, Computer conferencing, Document retrieval, Information management, Project control

Figure 1 – The first proposal for WWW. Source: (A short history of the web, n.d.).

Robert Cailliau, a Belgian systems engineer, and Tim Berners-Lee formalized this as a management proposal in November 1990. The crucial concepts and terms regarding the Web were overviewed in this document. According to the paper, a "web" of "hypertext documents" may be read by "browsers" in the "WorldWideWeb" "hypertext project" (A short history of the web, n.d.).

The first Web server was coded on a NeXT computer by Tim Berners-Lee by the end of 1990. It was the first operational web server running at CERN. The computer carried the following handwritten label in red ink to prevent it from being unintentionally turned off: "This device serves as a server. DO NOT TURN IT OFF!" (A short history of the web, n.d.).

In 1991, the first web server outside of Europe was up at SLAC, the Stanford Linear Accelerator Center, in California. And in 1993, CERN management decided that the web should act as an open standard for all to use. Without this document, we would not have the web as we know it now.


2.

Declaration

The following CERN software is hereby put into the public domain:

- W 3 basic ("line-mode") client
- W 3 basic server
- W 3 library of common code.

CERN's intention in this is to further compatibility, common practices, and standards in networking and computer supported collaboration. This does not constitute a precedent to be applied to any other CERN copyright software.

CERN relinquishes all intellectual property rights to this code, both source and binary form and permission is granted for anyone to use, duplicate, modify and redistribute it.

CERN provides absolutely NO WARRANTY OF ANY KIND with respect to this software. The entire risk as to the quality and performance of this software is with the user. IN NO EVENT WILL CERN BE LIABLE TO ANYONE FOR ANY DAMAGES ARISING OUT THE USE OF THIS SOFTWARE, INCLUDING, WITHOUT LIMITATION, DAMAGES RESULTING FROM LOST DATA OR LOST PROFITS, OR FOR ANY SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES.

Geneva, 30 April 1993

[illegible]
W. Hoogland
Director of Research

[illegible]
H. Weber
Director of Administration

opie certifiée conforme

ait à Genève le 03-05-93

ORGANISATION EUROPEENNE POUR LA RECHERCHE NUCLEAIRE
DIVISION DES FINANCES
- CERN -

Figure 2 – Software release of WWW. Source: (Software release of WWW into public domain, 2009).

Today, with nearly 2 billion websites, we do not remember or can not even imagine the world without the World Wide Web.

3.2 Web development

Web development, in general, is the practice of developing websites and maintaining and hosting them via the Internet. The web development process consists of web design, web content development, client-side/server-side scripting, and network security configuration, among other tasks (What is web development? - definition from Techopedia, n.d.).

Web developers are people who create and maintain websites using a variety of coding languages. The languages they use depend on what part of the website they are working on. There are two main branches of web development. The first one is called front-end, and the second is back-end (Marionletendart, 2021).

3.2.1 Differences of web development

Front End Development is the set of web development practices that use front-end languages, which are languages that can be understood by the web browser. It implies that the developer should know web standards and the processes of the web browser.

Client-side uses, according to (Anquetil, 2019):

- Making structure and designing of web pages
- Making interactive web pages
- Creating dynamic visual components
- Interactivity with the data sent by the web server
- Sending requests to the server
- Interacting with temporary storage
- Interacting with local storage

Back End Development is the set of web development practices using back-end languages that the server interprets. It requires the knowledge of writing code that enables web browsers to communicate with databases and servers.

Server-side uses, according to (Anquetil, 2019):

- Processing the user input

- Displaying the requested pages
- Structuring web applications
- Interacting with servers/storages
- Interacting with databases
- Querying the database
- Encoding of data into HTML
- Operations over databases like deleting, updating etc.

Although they play somewhat different roles, they cannot exist without each other.

3.2.2 Instruments of web development

Web developers are constantly running into some troubles with their work, so they are always on the lookout for ways to improve their workflow, speed things up, and ultimately make their time more profitable. Various web tools easily handle this job.

We can distinguish such tools as:

- **IDE** - Integrated Development Environment allows programmers to combine the various components of building a computer program. By merging typical software development capabilities into a single place, such as editing source code, creating executables, and debugging, IDEs boost programmer productivity radically. (Codecademy, n.d.).

- **Virtual environment** – A virtual environment is a technology that helps to maintain the separation of dependencies needed by various projects by constructing separate virtual environments for them. Most developers utilize this as one of their most crucial tools. (Python Virtual Environment: Introduction, 2022).

- **GitHub** - GitHub, Inc. is a provider of Internet hosting for software development and version control using Git. It offers the distributed version control and source code management functionality of Git, plus its features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, continuous integration, and wikis for every project (Williams, 2012).

- **Software framework** - A software framework is a structural or conceptual platform that allows developers and users to selectively specialize or override common code with generic capabilities. Frameworks take the form of libraries, where a well-defined application program interface (API) is reusable across the project being developed (What is software framework? - definition from Techopedia, n.d.).

- **Developer Tools** - Developer Tools is a complete set of development tools that is integrated right into the browser. With the help of these tools, you can swiftly detect issues, change web pages in real-time, and create better websites (Prokopets, 2020).

- **Website hosting** - Web hosting is a service that allows others to browse the content of your website online. In order to keep all of the files and data for the website, you rent space on a real server when you buy a hosting service (G., 27).

- **Front-end languages:** HTML5, CSS3, JavaScript.
- **Back-end languages:** PHP, ASP.NET, C++, Java, Python, Ruby on Rails.

- **Web database** - A web database is known as an information storage system that allows for online access. For example, an online community may have a database containing its members' usernames, passwords, and other details (Alex Paul, 2023).

This project uses PyCharm IDE, Python virtual environment, GitHub, Chrome Developer Tools, Digital Ocean hosting service, HTML, CSS, JavaScript, Python, PostgreSQL, and Django framework.

3.3 Python

Figure 3 – Python logo. Source: (The python logo, n.d.).

Python is an object-oriented, high-level programming language that is the world's most popular programming language. It is also the most useful programming language, not just among software developers but also among data analysts, scientists, mathematicians, and so on. Python is a multi-disciplinary language that can perform a variety of different tasks, such as artificial intelligence and machine learning, data analysis, visualization, and so on. But the biggest profit of Python is that you can automate repetitive tasks. For example, when you work with folders and files and need to do a lot of copying, renaming, and uploading instead, you can easily write a Python script to automate all of those tasks and save your time. You can use Python in various ways, from making mobile, web, and desktop applications to even hacking systems.

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC programming language, which was inspired by SETL and capable of handling exceptions and interfacing with the Amoeba operating system. Its implementation began in December 1989 (General Python FAQ, n.d.).

The reasons why to use Python according to (Solutions, 2017):

- **Readable and Maintainable Code** – To make maintenance and updates easier, you must concentrate on the quality of the source code while creating a software program. Python's syntactic rules make it possible to convey ideas without adding new code. In addition, Python promotes code readability in contrast to other programming languages and permits the use of English terms in place of punctuation. As a result, Python allows you to create unique apps without having to write additional code. You can upgrade and maintain the product with minimal additional time and effort if you have a readable and clean code base.

- **Multiple Programming Paradigms** - Python offers a number of programming paradigms, much like other current programming languages. Both object-oriented and structured programming are fully supported. Moreover, the language's characteristics support a number of functional and aspect-oriented programming principles. Python also has an autonomous memory management mechanism and a dynamic type system. Python's capabilities and programming paradigms make it possible to create complex and expansive software programs.

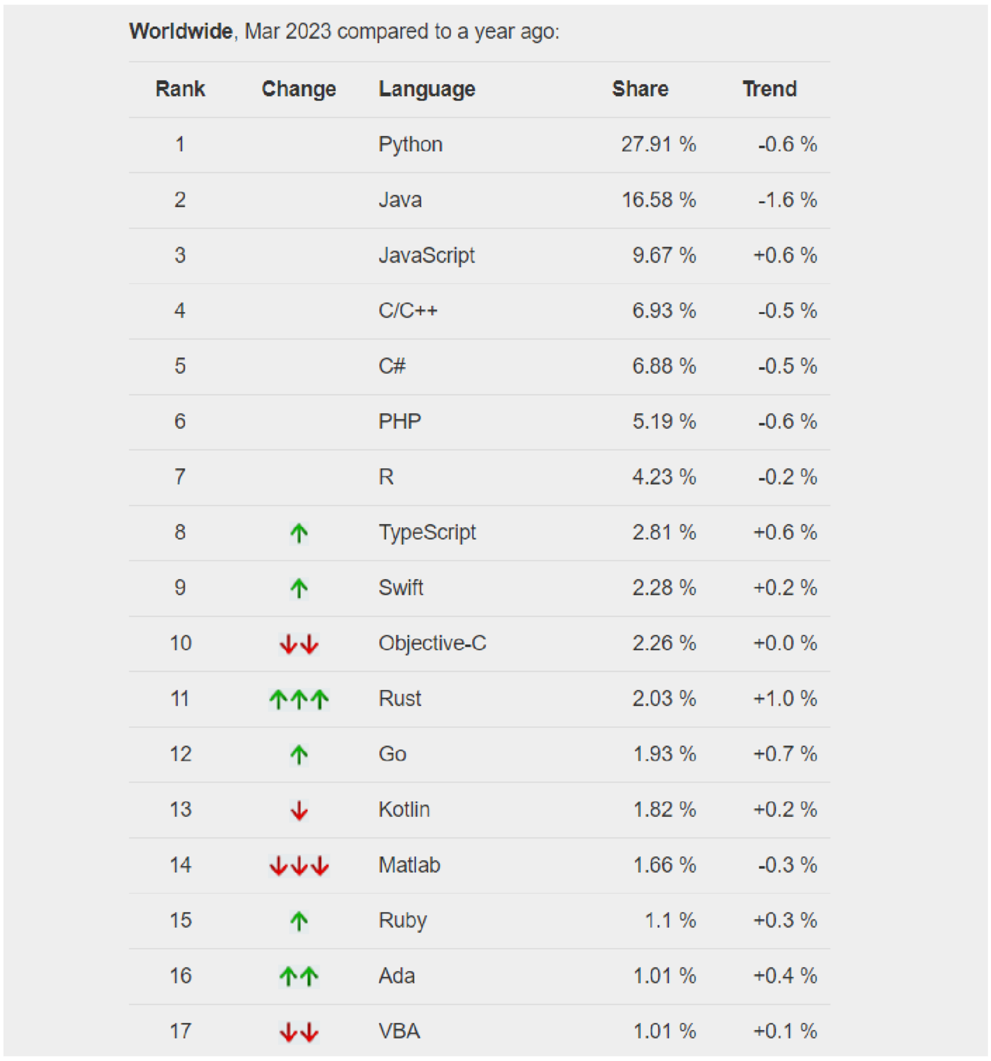
- **Compatible with Major Platforms and Systems** – Python presently supports a wide range of operating systems. Python interpreters can even be used to run the code on particular tools and platforms. The programming language Python is also an interpreted one. It enables you to run the same code on many platforms without having to recompile it. As a result, after making any changes, you are not necessary to recompile the code. Without having to recompile, you may execute the changed application code and see how the changes you made affected it right away. You may modify the code more easily with the help of the functionality without lengthening the development process.

- **Robust Standard Library** - Python succeeds above other programming languages due to its massive and robust standard library. With the standard library, you may select from a variety of modules based on your specific requirements. Each module also gives you the option to extend the functionality of the Python application without adding new code. For instance, while developing a web application in Python, you may utilize particular modules to handle operating system interfaces, construct web services, conduct string operations, and operate with internet protocols. By studying the documentation for the Python Standard Library, you may even learn more about other modules.

- **Many Open-Source Frameworks and Tools** - Python's status as an open source programming language enables you to drastically reduce the cost of software development. To save development time without raising costs, you may even employ a variety of free source Python frameworks, modules, and development tools. Even better, you may select from a variety of open source Python frameworks and development tools based on your unique requirements. For instance, you may use powerful Python web frameworks like Django, Flask, Pyramid, Bottle, and Cherrypy to simplify and accelerate the development of online applications. Similarly, by utilizing Python GUI frameworks and toolkits like PyQT, PyJs, PyGUI, Kivy, PyGTK, and WxPython, you may speed up the creation of desktop GUI applications.

- **Simplify Complex Software Development** - Python is a general-purpose programming language. Hence, you may create desktop and online apps using the programming language. Python may also be used to create sophisticated scientific and numerical applications. Python is built with tools that make data processing and visualization easier. Without spending additional time or effort, you may construct customized big data solutions by utilizing Python's data analysis tools. Moreover, Python's data visualization packages and APIs enable you to view and display data in a more attractive and useful manner. Several Python programmers now utilize Python to carry out tasks involving natural language processing and artificial intelligence (AI).

- **Adopt Test Driven Development** - Python may be used to quickly build a software application prototype. Also, by only reworking the Python code, the software application may be created immediately from the prototype. Python even makes it simpler for you to code and test at the same time by using the test driven development (TDD) methodology. Before developing any code, it is simple to construct the necessary tests, and you can use the tests to continually evaluate the application code. The tests can also be used to determine if the program complies with specifications based on its source code.



Worldwide, Mar 2023 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	27.91 %	-0.6 %
2		Java	16.58 %	-1.6 %
3		JavaScript	9.67 %	+0.6 %
4		C/C++	6.93 %	-0.5 %
5		C#	6.88 %	-0.5 %
6		PHP	5.19 %	-0.6 %
7		R	4.23 %	-0.2 %
8	↑	TypeScript	2.81 %	+0.6 %
9	↑	Swift	2.28 %	+0.2 %
10	↓↓	Objective-C	2.26 %	+0.0 %
11	↑↑↑	Rust	2.03 %	+1.0 %
12	↑	Go	1.93 %	+0.7 %
13	↓	Kotlin	1.82 %	+0.2 %
14	↓↓↓	Matlab	1.66 %	-0.3 %
15	↑	Ruby	1.1 %	+0.3 %
16	↑↑	Ada	1.01 %	+0.4 %
17	↓↓	VBA	1.01 %	+0.1 %

Figure 4 – PYPL table. Source: (PYPL popularity of Programming Language index, n.d.).

According to PYPL, which is an index of how popular the language is based on the number of tutorials searched on Google. So, on the picture, we can see that Python takes first place in the competition, occupying 27.91% of the market, which means that it is one of the most wanted programming languages of 2023. Nevertheless, it has over the year decrease in popularity, 0.6 %, it will stay at the peak for a long time.

In a nutshell, Python is a flexible language with a clear and easy-to-understand syntax. Mathematically, Python can achieve things that other programming languages can do, but Python's beauty in simplicity has made it grow far beyond other computer languages.

3.4 Django

django

Figure 5 – Django logo. Source: (Django overview, n.d.).

Django is an easy-to-use, open-source web framework built with Python by a team of experienced developers. This project is quite old. It has been maintained since 2005. Over that period, developers ironed out things that newbies or newcomers would make mistakes with and provided tools and configurations that help you automate the most common and most repetitive aspects of web development.

Advantages of Django according to (Django overview, n.d.):

- **Ridiculously fast** - Django was designed to help developers take applications from concept to completion as quickly as possible.
- **Fully loaded** - Django includes dozens of extras you can use to handle common web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds, and many more tasks - right out of the box.
- **Reassuringly secure** - Django takes security seriously and helps developers avoid common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery, and clickjacking. Its user authentication system provides a secure way to manage user accounts and passwords.
- **Exceedingly scalable** - Some of the busiest sites on the planet use Django’s ability to scale quickly and flexibly to meet the heaviest traffic demands.
- **Incredibly versatile** - Companies, organizations, and governments have used Django to build all sorts of things, from content management systems to social networks to scientific computing platforms.

3.5 MVC-MVT architecture

3.5.1 Traditional web application development model

Let's start by understanding the process model of traditional web applications. Whenever a client sends a request to the server, the request will be sent as an HTTP request to the web server, where the server accepts the request and identifies the requested web application. Then the requested page will be processed within the web server, and if there is any requirement to interact with the database, then the process interacts with the database. Consequently, it will generate a result that will be sent back as an HTTP response to the client.

3.5.2 Model View Controller (MVC)

MVC stands for Model-View-Controller, where:

The model is responsible for managing the data, directly handling the data logic and application rules, and retrieving data from the database when requested.

The view provides a user interface to present model data in a particular format.

The controller is responsible for complete orchestration. It is used to accept the request and respond to the client.

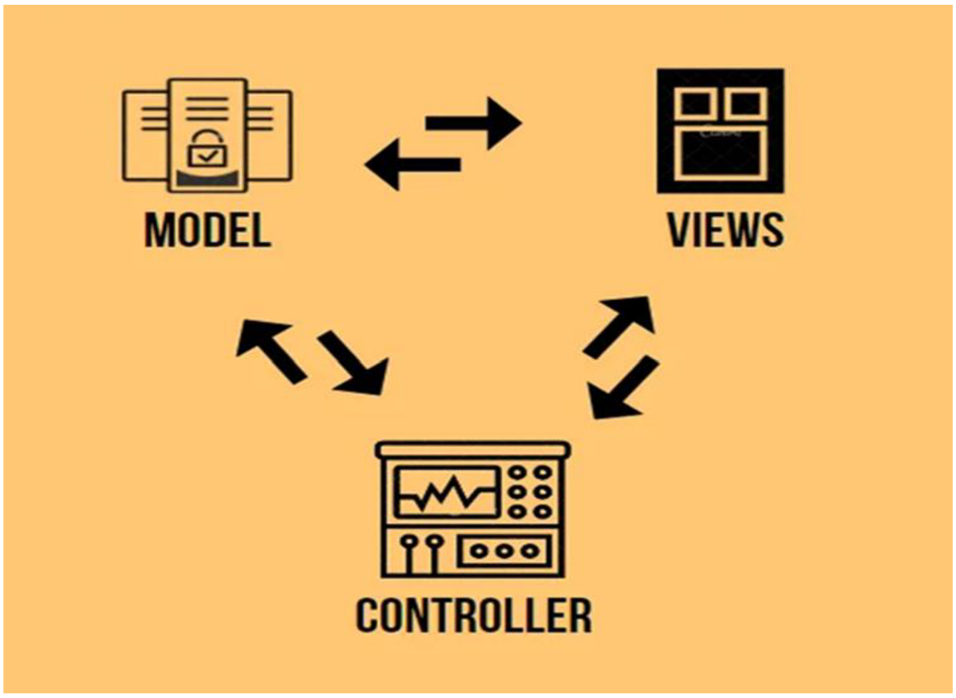


Figure 6 – MVC architecture. Source: (Nishant, 2020).

So, when a client sends a request to the web application, the controller will accept the request. If the client sends the request to update data present in the database, the controller accepts the request and manipulates the model that will manipulate the database. Then the controller will receive updated model details, from which model data goes to the view, where the view uses the model data, renders the UI as part of the requirement, and provides the result to the controller, which will then be returned to the client as an HTTP response.

3.5.3 Model View Template (MVT)

MVT or MTV stands for Model-Template-View, where:

Model, like in MVC, handles the data and logic part. The model defines the structure of stored data.

Templates are used to specify the structure of the output document. Templates are often used to create HTML content.

Views are used to receive HTTP requests from clients and return HTTP responses back to the clients. It is usually used to access databases, render templates, etc.

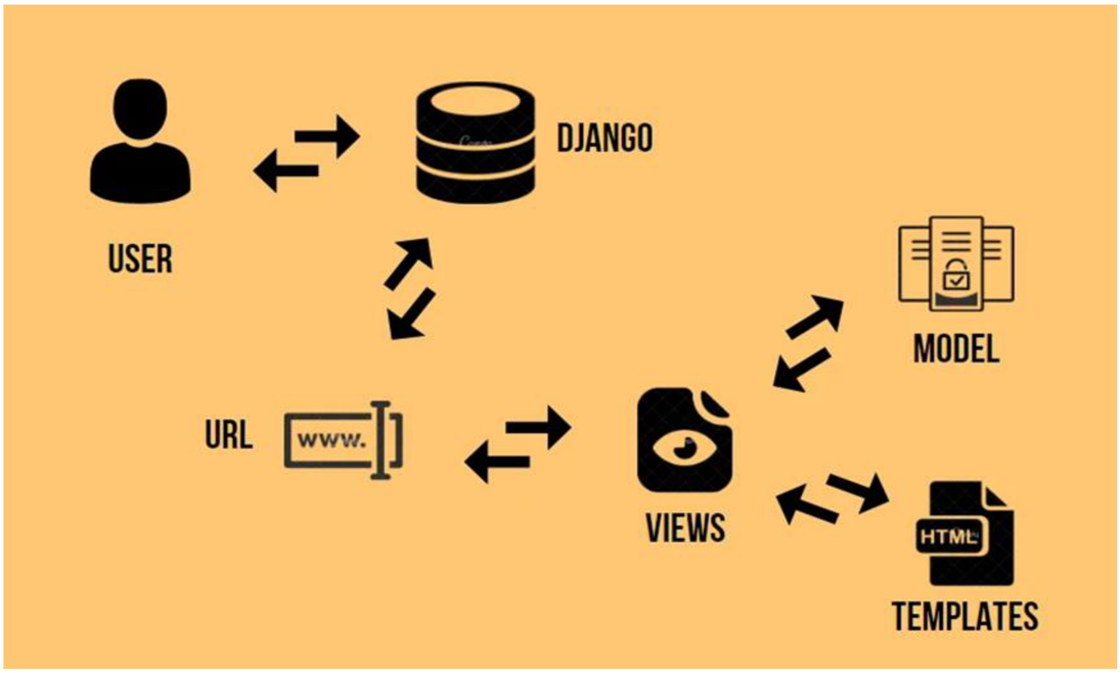


Figure 7 – MVT architecture. Source: (Nishant, 2020).

So, let’s understand how the Django web application works whenever a request has been submitted. Whenever the client sends a request to a Django web application, the Django framework accepts the request and submits the request to the URLs. The URLs act like mappers to redirect HTTP requests to the relevant view based on the requested URL. A view accepts the request, processes the instructions written within the appropriate view method, and provides the result as an HTTP response to the client. When a client sends an HTTP request for a Django web application for interaction with some database, the request will be given to the Django framework. Then it will be redirected to the URL. The request will be given to the relevant view based on the requested URL pattern. The view will identify the model to be used for interaction with a database, and it will read or write data to the model as required. It also uses a template to provide an efficient UI for the client. Once the page has been generated, the view will provide the result as an HTTP response to the client.

3.6 Bug-tracking system

A bug tracker or issue tracker is all about organizing and storing data from test projects. That data is presented in the issue tracker and stored in a database, which is the system's main component. Data from issues may include the reported time, the person who reported it, the deadline, the person assigned to this issue, and so on. The core principle of the bug tracking system is to track the progress of bugs at each stage of fixing them.

Within a bug tracker, all test results are presented in the form of a statistical diagram or chart. The system also should provide different functionalities for different roles in a team. According to the user's permission, the system should give access to perform tasks such as creating, updating, viewing, or deleting the bug. All the bugs in the system should be related to some test project.

The primary usage of a bug tracker is to give an overview of development requests and their statuses. Information from a categorized list of backlogs is useful for determining the future development of a project or simply a product's roadmap.

3.6.1 Bug-tracking

Bug tracking, or issue tracking, is the process of logging and monitoring errors during product testing, prioritizing, and fixing them. It is an essential process because there are probably more than a thousand issues in a large system. Every issue should be evaluated, monitored, and prioritized for fixing. This is necessary to provide a product that meets the user's needs and stays ahead of the competition (Nagappan, 2022).

Bug life cycle - the sequence of stages that a bug goes through on its way from the moment it was discovered to its final fix and closure.

Typical defect life cycle according to (What is bug tracking?, n.d.):

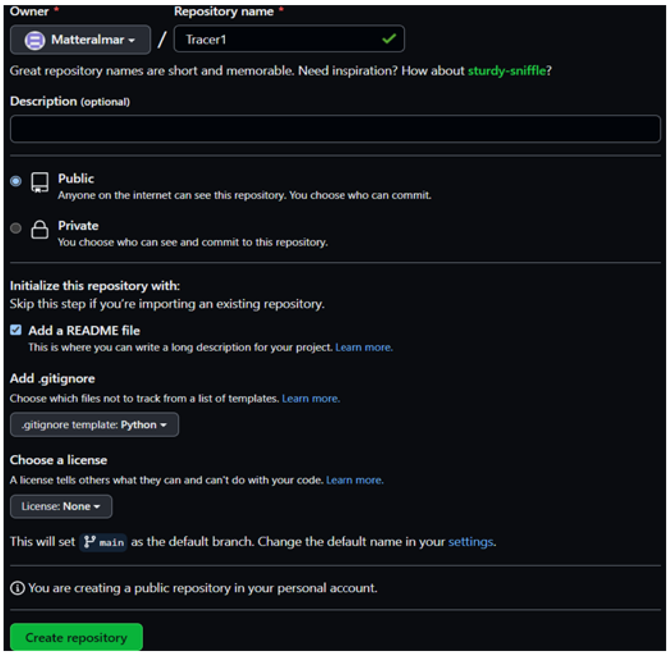
- Active: Investigation is underway
- Assigned: Responsible for fixing the defect is assigned
- Test: Fixed and ready for testing
    - fixed - fixes included in other versions.

    - duplicate - repeats a defect already in progress.
    - not fixed - works according to the specification, has too low a priority, the fix is delayed until the next version, etc.
    - irreproducible-request for additional information about the conditions under which the defect manifests itself.
- Closed: Can be closed after QA retesting or if it is not considered to be a defect
- Reopened: Not fixed and reactivated

4 Practical Part

4.1 Quick start

Before working on your project on your computer, you should create a repository on GitHub. You have to do so because you will interact with GitHub when you post your code and then deploy it from GitHub to your host server. So, you can open your GitHub account and find a button to create a new repository. You should specify the repository name, leave it as public, tick the option to add a "README" file, choose Python for the "gitignore" file, and then create the repository. After making your repository, press the "Code" button and copy the HTTP link from there.



Owner * Repository name *

Matteralmar / Tracer1

Great repository names are short and memorable. Need inspiration? How about sturdy-sniffle?

Description (optional)

● Public
Anyone on the internet can see this repository. You choose who can commit.

○ Private
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☑ Add a README file
This is where you can write a long description for your project. Learn more.

Add .gitignore
Choose which files not to track from a list of templates. Learn more.

.gitignore template: Python

Choose a license
A license tells others what they can and can't do with your code. Learn more.

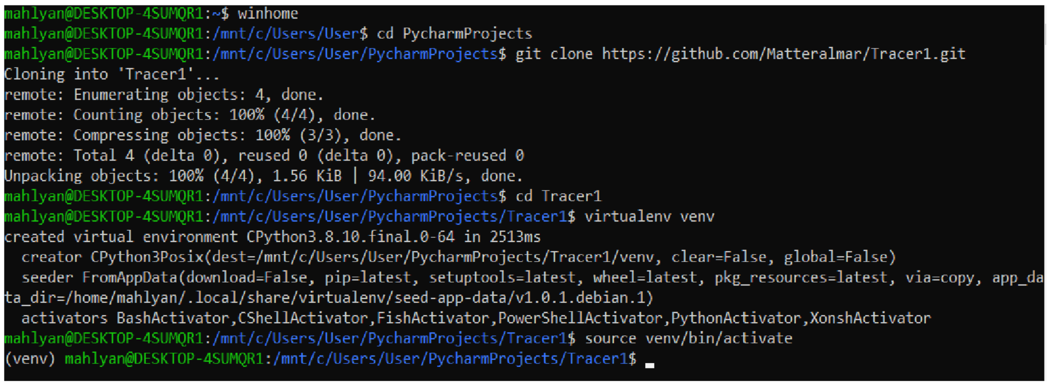
License: None

This will set main as the default branch. Change the default name in your settings.

You are creating a public repository in your personal account.

Figure 8 – GitHub form. Source: author.

So now we are going to start our project. Let's go to the terminal. In my case, I am using Ubuntu, which is a Unix-based terminal. Let's get to the IDE path. You clone the GitHub repository using a link you copied by writing "git clone" with a link at the end. After that, you can "cd" into this project. There, you are going to create a Python virtual environment for this project. After creating a virtual environment, you will activate it.

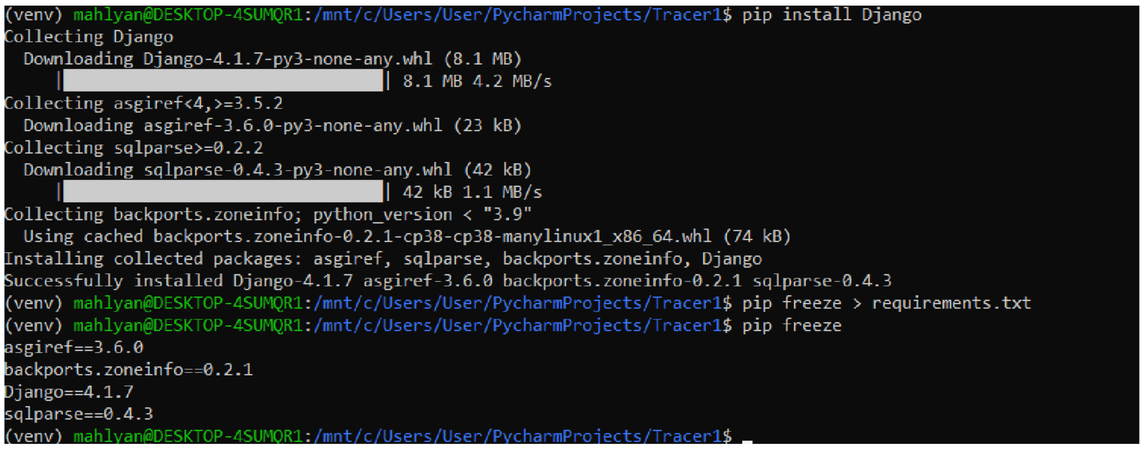


```
mahlyan@DESKTOP-4SUMQR1:~$ winhome
mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User$ cd PycharmProjects
mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects$ git clone https://github.com/Matteralmar/Tracer1.git
Cloning into 'Tracer1'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 1.56 KiB | 94.00 KiB/s, done.
mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects$ cd Tracer1
mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$ virtualenv venv
created virtual environment CPython3.8.10.final.0-64 in 2513ms
  creator CPython3Posix(dest=/mnt/c/Users/User/PycharmProjects/Tracer1/venv, clear=False, global=False)
  seeder FromAppData(download=False, pip=latest, setuptools=latest, wheel=latest, pkg_resources=latest, via=copy, app_da
ta_dir=/home/mahlyan/.local/share/virtualenv/seed-app-data/v1.0.1.debian.1)
  activators BashActivator,CShellActivator,FishActivator,PowerShellActivator,PythonActivator,XonshActivator
mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$ source venv/bin/activate
(venv) mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$
```

Figure 9 – Configuration of virtual environment. Source: author.

Now you can see the name of our virtual environment on the left side, which tells you that this terminal session is activated inside the virtual environment. So, if you are to install the Python package, it will get installed inside the "venv" folder and not inside your computer's root dependencies.

Once you have that displayed, then what you need to do is install Django. You can see in the terminal what dependencies are installed and check that all your dependencies are installed, and you can output all dependencies into a file called "requirements.txt".



```
(venv) mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$ pip install Django
Collecting Django
  Downloading Django-4.1.7-py3-none-any.whl (8.1 MB)
     |████████████████████████████████| 8.1 MB 4.2 MB/s
Collecting asgiref<4,>=3.5.2
  Downloading asgiref-3.6.0-py3-none-any.whl (23 kB)
Collecting sqlparse>=0.2.2
  Downloading sqlparse-0.4.3-py3-none-any.whl (42 kB)
     |████████████████████████████████| 42 kB 1.1 MB/s
Collecting backports.zoneinfo; python_version < "3.9"
  Using cached backports.zoneinfo-0.2.1-cp38-cp38-manylinux1_x86_64.whl (74 kB)
Installing collected packages: asgiref, sqlparse, backports.zoneinfo, Django
Successfully installed Django-4.1.7 asgiref-3.6.0 backports.zoneinfo-0.2.1 sqlparse-0.4.3
(venv) mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$ pip freeze > requirements.txt
(venv) mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$ pip freeze
asgiref==3.6.0
backports.zoneinfo==0.2.1
Django==4.1.7
sqlparse==0.4.3
(venv) mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$
```

Figure 10 – Installation of Django. Source: author.

When you have Django installed, you can create your project and run different sorts of commands using Django. Your first Django command will be the creation of a project folder. Write the command "django-admin start project bugTrack . ". This command created the folder "bugTrack" inside your project, and I used space with a dot to create the folder inside the folder I am currently working in, which helped to lay out the project a little bit easier.

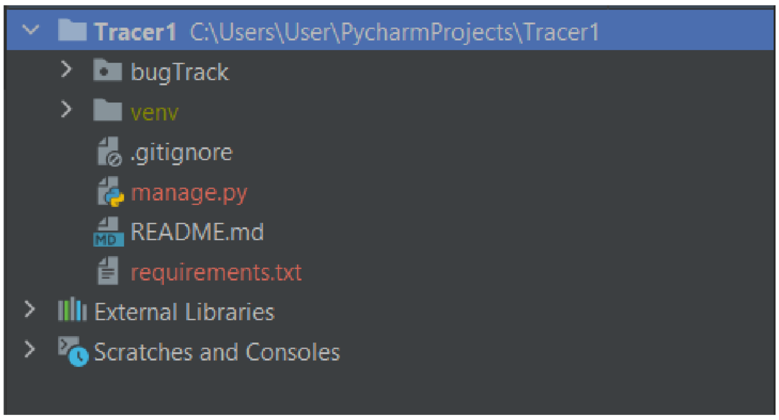
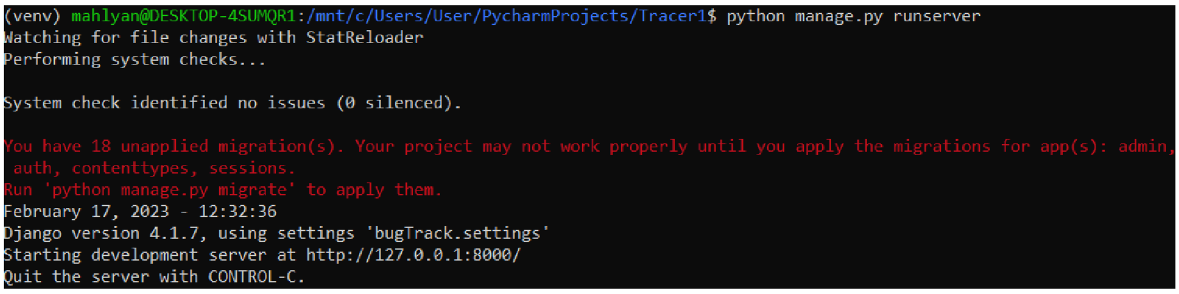


Figure 11 – Main folder content. Source: author.

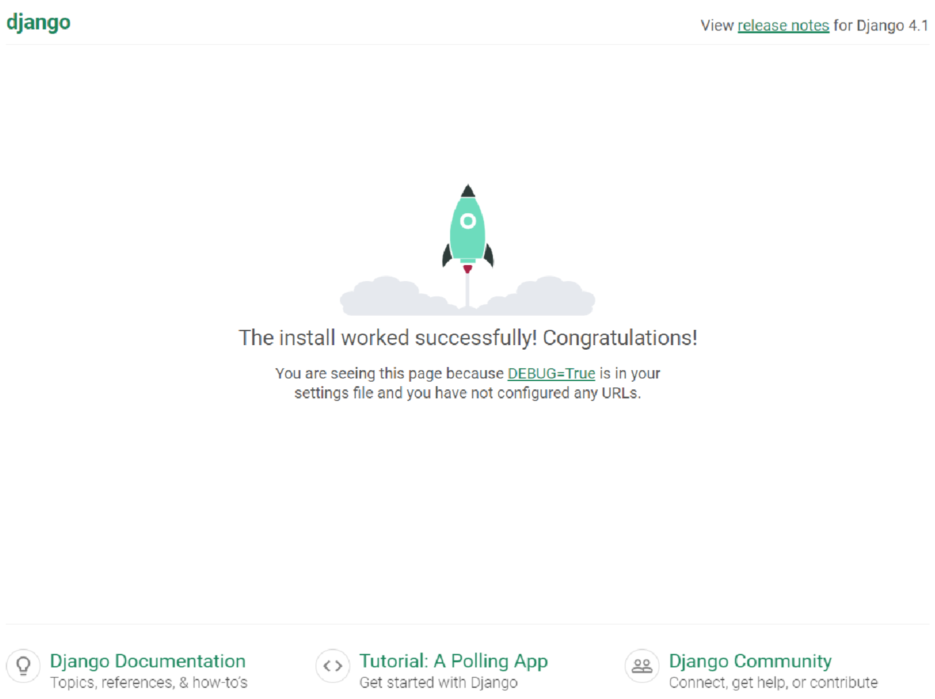
You have everything in place now to run the server. Inside the terminal, make sure that you have the virtual environment activated. Then you will call Python and make use of the manage.py file, which allows you to run a whole bunch of different commands. So, run "python manage.py runserver" in your terminal.



```
(venv) mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
 auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
February 17, 2023 - 12:32:36
Django version 4.1.7, using settings 'bugTrack.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Figure 12 – Unapplied migrations. Source: author.

As you can see, there are unapplied migrations that we will deal with later. Still, underneath that, you can see your version of Django using "settings.py" within the folder you created recently, your development server link, and the quit command to stop the server. Paste the link from a terminal into your web browser.


django

View release notes for Django 4.1

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Django Documentation
Topics, references, & how-to's

Tutorial: A Polling App
Get started with Django

Django Community
Connect, get help, or contribute

Figure 13 – The rocket is loaded up. Source: author.

Here you can see the page with rocket loaded up, which is an identifier that your server is running. You can also see valuable links to Django documentation, tutorials, and the Django community underneath the page. This is how to run the server, so whenever you want to test things, you need to have your server running so you can play around and navigate your site. You can go back to your terminal to see the output.

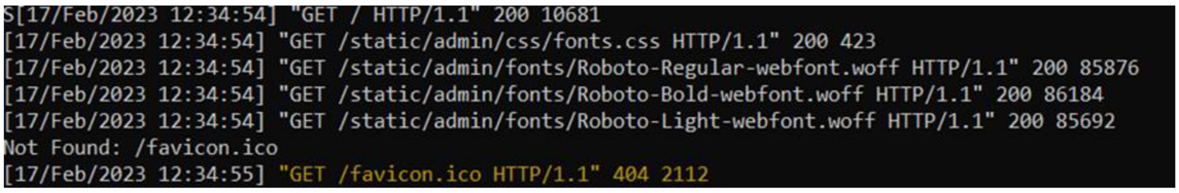
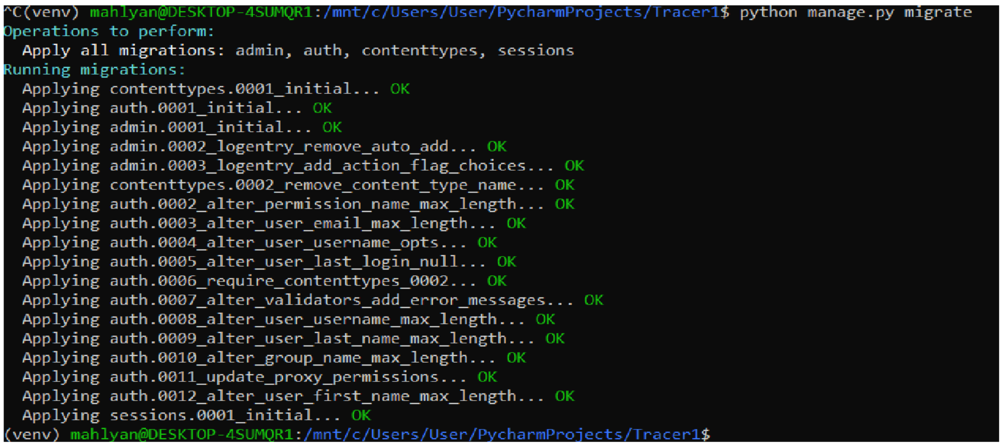

```
s[17/Feb/2023 12:34:54] "GET / HTTP/1.1" 200 10681
[17/Feb/2023 12:34:54] "GET /static/admin/css/fonts.css HTTP/1.1" 200 423
[17/Feb/2023 12:34:54] "GET /static/admin/fonts/Roboto-Regular-webfont.woff HTTP/1.1" 200 85876
[17/Feb/2023 12:34:54] "GET /static/admin/fonts/Roboto-Bold-webfont.woff HTTP/1.1" 200 86184
[17/Feb/2023 12:34:54] "GET /static/admin/fonts/Roboto-Light-webfont.woff HTTP/1.1" 200 85692
Not Found: /favicon.ico
[17/Feb/2023 12:34:55] "GET /favicon.ico HTTP/1.1" 404 2112
```

Figure 14 – Terminal output. Source author.

You can see the whole bunch of output requests made to the server in the terminal. You can see GET requests to the root path. You can notice the version of the HTTP protocol and HTTP Response 200 that requests an actual HTML page. All this information is logged out for every single request. It shows requests made to your Django server, and the client makes them. You can also see the output in red about unapplied migrations of Django default apps. It is written to run "python manage.py migrate" to apply those migrations, which is an important command that applies changes to the database, and you will use it frequently. Now you can execute this command.



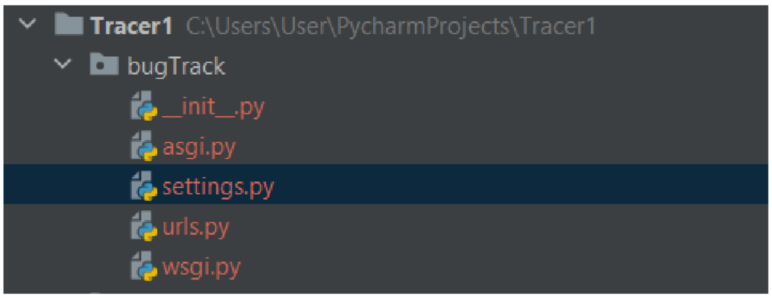
```
^C(venv) mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... 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 auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
(venv) mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$
```

Figure 15 – Migrations. Source: author.

If you look back at our text editor, you can see a new file called "db.sqlite3", your local development database. When you are in local development, Django, by default, comes with a configuration to use SQLite as the database; nevertheless, you will use a different database when you deploy your application. Now you have your server running on the terminal without any errors.

4.2 Project folder

Let's open our "bugTrack" folder. Here you can see files that are important for our project to function.



```
Tracer1  C:\Users\User\PycharmProjects\Tracer1
  bugTrack
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
```

Figure 16 – Project folder files. Source: author

The first one is the "__init__.py" file, making this directory a Python project. The "asgi.py" file gives the opportunity to run the Django server asynchronously. The "wsgi.py" looks similar and works the same way but uses a different code interface. The folder "urls.py" is the root URL configuration of a project, where Django starts to look for URLs in the project, so whenever a web request is made, it is going to go through all URLs, starting from the top, to find the one path that matches the request and use the relevant view. But a standard convention is to point URLs from the project "urls.py" file to the app "urls.py" file, so Django starts to look inside project folder URLs and then goes to app URLs. It looks through URLs to find the requested path. The most important file is the "setting.py" file, a hub of settings where you define the variables you will use throughout your project, like configuring your database connection, creating the path to your templates, and so on. Let's go into it.

```
DEBUG = True

SECRET_KEY = 'some_key'

BASE_DIR = Path(__file__).resolve().parent.parent

ALLOWED_HOSTS = []
```

Figure 17 – Settings file code. Source author.

First, there is "BASE_DIR", which is using the "Path" variable, and this constructs the path to the root of your project, which is not the "bugTrack" folder but the parent folder of "bugTrack", which is the base directory. You also can see "SERET_KEY", which, by its name, means that it should be kept secret and shouldn't be accessible in any way, especially in production. And "DEBUG", which is set to "True" while you are in local development, but once you are in production, you will turn it to "False". The "ALLOWED_HOSTS" is the list of hosts that this project is allowed to be hosted on.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

]
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'whitenoise.middleware.WhiteNoiseMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

Figure 18 – Settings file code. Source author.

"INSTALLED_APPS", which references default Django apps, and if you are to install our apps, you will add them to this list. "MIDDLEWARE", the list of programs that alter Django's request/response processing. "ROOT_URL_CONF", which is the path to the "urls.py" file within the "bugTrack" folder.

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',

'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'bugTrack.wsgi.application'


# Database
# https://docs.djangoproject.com/en/4.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3'
    }
}
```

Figure 19 – Settings file code. Source: author.

"TEMPLATES", which is the list of configuration objects for templates, and the most important is the "DIRS" keyword, which stands for directories where you specify your folders for templates. There is "WSGI_APPLICATION", which is pointing to "wsgi.py" in the "bugTrack" folder. "DATABASES", a database configuration where a built-in sqlite3 database is used, comes from Django, and a keyword name that points to the path of the database is "BASE_DIR".

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValid
ator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/4.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'Europe/Prague'

USE_I18N = True

USE_TZ = True
```

Figure 20 – Settings file code. Source: author.

"AUTH_PASSWORD_VALIDATORS", which validates when a user enters a password. Then there is internationalization, which has "LANGUAGE_CODE" set by default to English, "TIME_ZONE" set by default to UTC", and the other two variables, which are internationalization itself and can be set to "True" or "False". And the last one is "STATIC_URL", which is the URL inside the web application that will host all static files like CSS, JavaScript, and images.

4.3 App folder

In this chapter, we will figure out how the app folder works, what files it contains, and the functionality of each file. But first, you need to understand what is meant by the app. The app is a containerized module of code that relates to one aspect of your project or web application. This means that it handles one part of the project, like user authentication, for example. We will create an app called "tickets" because we are building an app that manages tickets. This app will deal with creating and managing a ticket through its entire life cycle. Write "python manage.py startapp tickets" to create one in your terminal. And to make it visible to Django, you should add "tickets" to "INSTALLED_APPS".

4.3.1 Models.py

Now you will look at one of the most crucial concepts in Django, which is called "model". Models are representations of your database schema, and the use of Django models is to present the structure of stored data. To see an example, go to the "tickets" app in the "models.py" file.

In this file, you can see that Django already provides us with the ability to import models so we can use them. The way you write a model is by using Python classes, so type the "class" keyword with the name "User," and this model is going to inherit from "models.Model", where "Model" is a Python class inside "models", which makes class a model.

```
from django.db import models

class User(models.Model):
    username = models.CharField(max_length=20)
    first_name = models.CharField(max_length=20)
    last_name = models.CharField(max_length=20)
    age = models.IntegerField(default=0)
```

Figure 21 – User model code. Source: author.

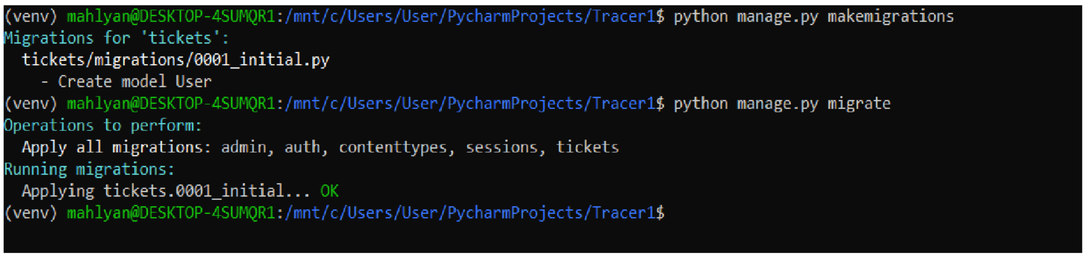
Inside the model, you will create properties that represent how the data inside the model would look like. Take, for example, "username," which is a string, so if you want to create it, you need to type it in the field "username." We would use "models" that come from Django to make the data type of this field, which in this case will be the character field because the database table requires you to restrict the type of data inside each column, and the character field does exactly that. It creates a column in the database table that limits the data type to a string. You also must pass in so-called metadata, or information about that property. In the case of the character field, use "max_field" to limit the length of the data you save in this field in the database table.

You can define what data is stored in each model field by its data type. In our project, you can distinguish such data types as "CharField" for small to large-sized strings, "TextField" for a large text field, "BooleanField" field for boolean values, "DateField" for date values, and "DateTimeField" for date and time values. There are also two types of relationship fields: "OneToOne" for one-to-one relationships and "ForeignKey" for one-to-many relationships. Such metadata for those fields was used as "max_length" to limit the length of the string, "unique" to exclude repetition, "default" as a starting value, and "null" which means that there is or there is no constraint for the field to be filled, "blank" means that you can leave this field blank or not, "on_delete" is an action taken after the deletion of another object in the database, "auto_now_add" is used to set the time when an instance is created, "auto_now" is the time when an instance is modified, and "choices" is used to pick from a tuple of values.

```
def __str__(self):
    return self.ticket.title


def post_user_created_signal(sender, instance, created, **kwargs):
    if created:
        Account.objects.create(user=instance)

post_save.connect(post_user_created_signal, sender=User)
```

Figure 22 – Model functions code. Source: author.

You can also see the function "__str__" within classes to display objects by their name in Django admin and "post_user_create_signal", which listens to the event when the user is created to create a corresponding user account.



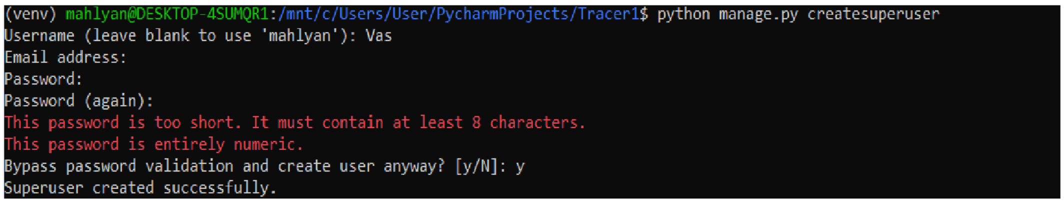
```
(venv) mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$ python manage.py makemigrations
Migrations for 'tickets':
  tickets/migrations/0001_initial.py
    - Create model User
(venv) mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, tickets
Running migrations:
  Applying tickets.0001_initial... OK
(venv) mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$
```

Figure 23 – Making migrations. Source: author.

Whenever you create a model, you need to do two things: solidify it as part of the database schema and build it inside the database. First, you need to open the terminal and run "python manage.py makemigrations" to create a schema according to your model. The schema will be located in the "migrations" folder. Here is one class with a name for a model created with field operations and a list of operations to apply to the database. And in this file is "migrations.CreateModel", where "CreateModel" will create a table in the database. It will give a table the "name" of the model specified and apply the following fields to this table. The first one is "id", which is given by default by Django and all other fields that were coded in models.py. This migration class represents what you want to happen to the database whenever you create a schema. In other words, it is just a blueprint, so still, nothing is happening to the database. Open the terminal and input the "python manage.py migrate" command. After executing this command, Django will look through all your apps and look inside the migrations folders of those apps. It will run through all the migrations inside those apps, and those migrations that are not applied yet will be executed and applied to the database.

4.3.2 Admin.py

So now that you understand how Django models work, we can proceed with Django admin functionalities. Django administration is usually used for working with models created and specified in "admin.py". It is a handy tool because you can do all kinds of filtering, create new entries, delete them, retrieve them, update them, and look at the information inside the provided dashboard.



```
(venv) mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$ python manage.py createsuperuser
Username (leave blank to use 'mahlyan'): Vas
Email address:
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

Figure 24 – Creation of superuser. Source: author.

To have the possibility to work in Django admin, you need to create a superuser. To do so, you need to run "python manage.py createsuperuser". Then you should input your credentials with your name and password, which you will use to enter.

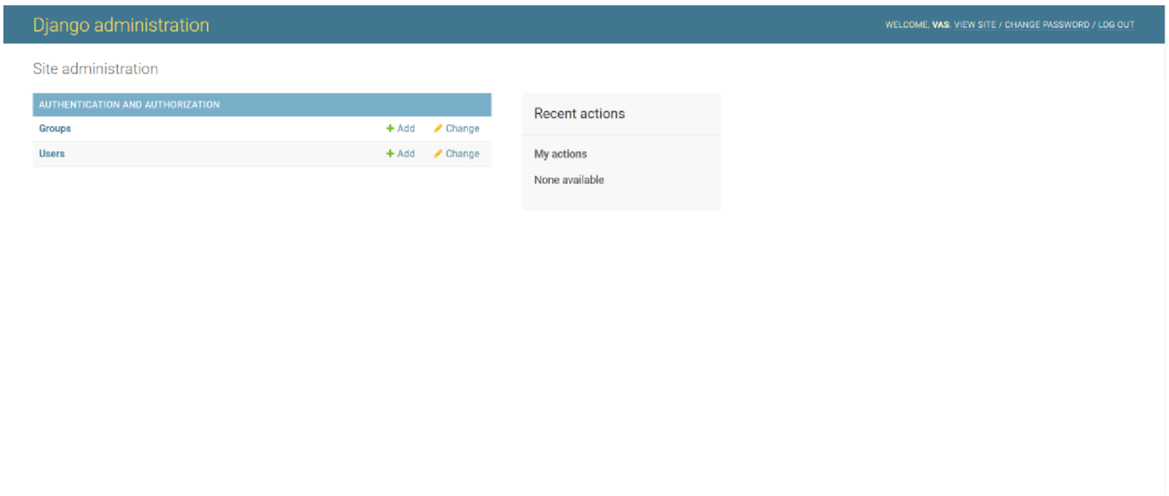


Figure 25 – Admin page. Source: author.

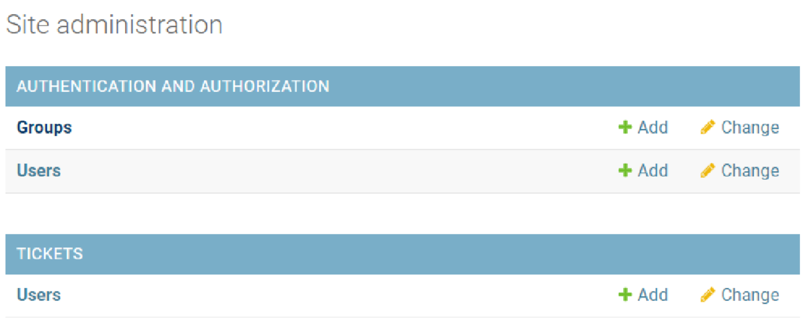
After you create a superuser, you can run our server using the "python manage.py runserver" command, then copy-paste the localhost address into your web browser and add to it "/admin/" which you can find as a default path in the project's directory "urls.py". After that, you can sign in and have your first view of Django's administration.

To start your work with models, you need to state them first. So, open the "admin.py" file in the "tickets" folder, where you are going to import the model.

```
from django.contrib import admin
from .models import *

admin.site.register(User)
```

Figure 26 – Registration of user model in code. Source: author.

And if you go back and refresh the administration page, you can see that you have a whole new section there called "Tickets". You can observe a "User" subsection inside this app section, which is a user model.

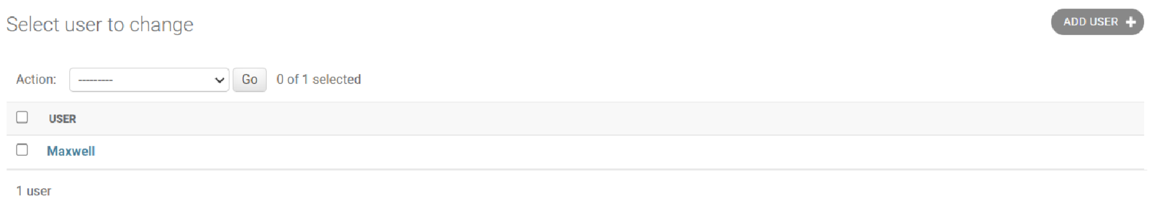

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

TICKETS		
Users	+ Add	Change

Figure 27 – Administration sections. Source: author.

Let’s go to the "User" section. You can add a new user by pressing "ADD USER" on the top right corner of the screen. Then you can fill in the fields specified in the model and save them to the database. After you quit editing your model, you will be provided with a string name, which is established in the "models.py" "__str__" function. You can update data about this model by clicking on its label and editing it similarly.


Select user to change

ADD USER +

Action: --------- Go 0 of 1 selected

USER
Maxwell

1 user

Figure 28 – Creation of an object. Source: author.

4.3.3 Forms.py

Now you will look at the process of creating new objects in a database that is done using the so-called "form". The form is a big part of Django, and Django has a lot of built-in functionality to help us with it. And what you are going to do is create a form that allows creating objects similar to Django admin.

First of all, you need to create a file, which is called "forms.py" in the "tickets" app. Then you need to import the "forms" module from Django so you can build out your forms that work very well with your models.

```
from django import forms

class UserForm(forms.Form):
    username = forms.CharField()
    first_name = forms.CharField()
    last_name = forms.CharField()
```

Figure 29 – User form code. Source: author.

The way you create forms is by creating a class. In our case, you will make "class UserForm(forms.Form):" where "UserForm" is the name of a class and "forms.Form" is an inherited module. After that, you can specify fields that you want to include in the same way as in models, for example, "username=forms.CharField()". And now, the form is ready to be used.

```
class TicketCategoryUpdateForm(forms.ModelForm):
    class Meta:
        model = Ticket
        fields = (
            'status',
            'priority',
            'type',
        )
```

Figure 30 – Model-form code. Source: author.

But in this project, a different syntax was used. I imported a model from my project, inherited from "forms.ModelForm", to work directly with a model I wanted to change, and used a new class called "Meta", which I set according to the model fields that I wanted to display in the form. In this way, I created a form or told Django to make a form for a model.

```
    def __init__(self, *args, **kwargs):
        request = kwargs.pop("request")
        user = request.user
        statuses = Status.objects.filter(organisation=user.account)
        priorities =
Priority.objects.filter(organisation=user.account)
        types = Type.objects.filter(organisation=user.account)
        super(TicketCategoryUpdateForm, self).__init__(*args,
**kwargs)
```

```
        self.fields["status"].queryset = statuses
        self.fields["priority"].queryset = priorities
        self.fields["type"].queryset = types
```

Figure 31 – Form function code. Source: author.

As you can see in this file, the "__init__" function is responsible for filtering choices for fields in the form. So, I wrote the variable "request", where I get an extra argument from view through the kwargs variable to get the current user. Then I filtered all needed objects with specific information. I called the "super" of the form above with the "__init__" method to call the "__init__" method with expected arguments. And I accessed field properties and assigned values that I filtered. The requesting user was checked according to the filter data that needed to be displayed because otherwise, there would be a free choice of inappropriate data.

4.3.4 Mixins.py

"Mixins.py" is a custom permission-checking file that is useful in your project if your project has some parts or links that are open or forbidden for particular roles. Django also has some built-in mixins, like "LoginRequiredMixin", that check if a user is logged in and, following that, execute some functionality. But I created my own mixins because I have roles in my project, and each role individually requires a custom mixin.

```
class OrganizerAndLoginRequiredMixin(AccessMixin):
    def dispatch(self, request, *args, **kwargs):
        if not request.user.is_authenticated or not
request.user.is_organizer:
            return redirect("dashboard:dashboard-chart")
        return super().dispatch(request, *args, **kwargs)
```

Figure 32 – Example mixin code. Source: author.

All the mixins that I have in my project look similar to each other and differ only in the condition statement. For example, look at the "OrganizerAndLoginRequiredMixin" mixin. Here, you can see that Python classes are used as well. You also need to give it a name; in my case, it is "OrganizerAndLoginRequiredMixin" and inherits functionality from "AccessMixin". Then there is the "dispatch" method that is executed first on the request of a user, a condition that checks if a requested user is not authenticated or is not an organizer;

then you need to redirect the user to the page permissible by code, and if not, then it will allow the user to enter the page requested.

4.3.5 Views.py

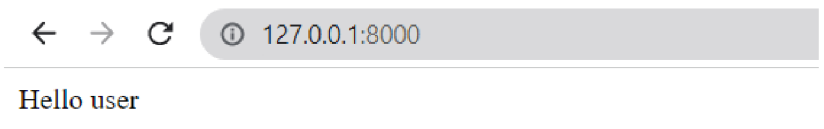
You learned how to use mixins in the project. Now you will look at the most common part of Django—views. The view is part of Django that handles web requests and sends responses. In this chapter, you will create the first simple view as an example and provide an overview of other views in this project.

You will create a function-based view, meaning you will use a Python function to handle the request. To do so, you need to define a standard Python function "def home_page(request)", where "request" is one of the parameters. Django will provide the request to your function every time someone goes to a specific page on the website, and you can get information from this request like the logged-in user, the IP address of the request, what device the user is requesting from, what browser the user uses, and so on. Before you start to write the function, you need to make an import of "from django.http import HttpResponse". And this function is simply going to return an HTTP response with some text on a page to the user's request in a browser that looks like "return HttpResponse("Hello user")" in code.

```python
from django.shortcuts import render
from django.http import HttpResponse

def home_page(request):
    return HttpResponse("Hello user")
```

Figure 33 – Function-based view code. Source: author.

Then you need to go to our project folder and find a file called "urls.py". Here you will create the path to your view so Django can correlate the path in the browser with the path in the project. You already have one path to the admin page in "urlpatterns", and "urlpatterns" is a list of paths that manage the path in a web application, and it tells Django which view handles that path. But you will create your own, situated at the root of the web application. So, you need to import your view from the app folder: "from tickets.views import home_page". After that, you need to write inside "urlpatterns": "path(, "home_page)"- where the first parameter is the root of your web app and the second is the name of a view. So now you can go to your terminal and run the server. Check if you are in the root folder, where the "manage.py" file is situated. Run "source venv/bin/activate" in your terminal. When you have your virtual environment activated, you can write "python manage.py runserver", copy your server's local address into your browser, and now you can see "Hello user" in the top left corner. So, you created the first functional view in the project, which is the basic view in Django.



Hello user

Figure 34 – Rendered page. Source: author.

Function-based views can get more complicated due to the fact that most of the code repeats too often, so the code becomes too massive and confusing. To simplify it, there are class-based views, which are common in this project.

To understand what class-based views are, you first need to know the acronym "CRUD+L", which stands for create, retrieve, update, delete, and list, which are considered actions you can perform on a page. Django generic views also have the same structures, so this project has "CreateView", "DetailView", "UpdateView", "DeleteView," and "ListView". "CreateView" is used to add new data to the database, "DetailView" is used to retrieve and show data of the concrete object in the database, "UpdateView" is used to change the data in the database, and "ListView" is used to list all objects of a model from the database. You can import all of them by writing "from django.views import generic", and then you can start the class-based view like this: "class StatusUpdateView (generic.UpdateView):" - where, as expected, there is a class, not a function, a name of a view and inherited functionality from "generic.UpdateView".

```
class StatusUpdateView(OrganizerAndLoginRequiredMixin,
generic.UpdateView):
    template_name = "tickets/status_update.html"
    form_class = StatusModelForm
    context_object_name = "status"
```

```python
    def get_success_url(self):
        return reverse("tickets:status-list")

    def get_queryset(self):
        user = self.request.user
        queryset = Status.objects.filter(organisation=user.account)
        return queryset

    def form_valid(self, form):
        color_data = form.save(commit=False)
        color_code_client = self.request.POST['CI']
        color_data.color_code = color_code_client
        return super(StatusUpdateView, self).form_valid(form)
```

Figure 35 – Class-based view code. Source: author.

Inside the "StatusUpdateView," you can see some variables with values assigned to them and methods in the form of functions, which we will discuss in the next paragraph. First is the "template_name," where you specify the path to the template, which will be shown when this view is requested. The second is "form_class," which determines what form from "forms.py" will be used in HTML. The third one, "context_object_name", uses the custom name of your query set for more understandable HTML code.

After mentioning class-based variables, you can observe in views such things as method functions that execute some part of the page's functionality. You can define such methods as "form_valid", "get_success_url", "get_queryset", "get_form_kwargs", "get_context_data", "get" and "post". Each of them reacts to the interaction with the part of the page it is responsible for. "form_valid" is triggered after the form is valid to Django and usually used to do some job after the form is validated, "get_success_url" is used when the functionality of the view is fulfilled and done so it redirects the user to other accessible pages, "get_queryset" is a very common method that gets objects of a model from the database that can be used to change, show, or delete them. "get_form_kwargs" is used to put useful information into "kwargs" and use it in form. Using "get_context_data," you can pick some number of objects and then use them in the template with the stated name, "get" and "post" are methods that process GET requests and POST requests, which are sent via the HTTP protocol by the client. And inside them, you usually work with models querying the database to get data and changing it according to our conceived functionality.

4.3.6 Urls.py

You learned and understood how views work, and now let’s take a look at "urls.py," which governs Django and interprets the path in the browser into a path inside the project. Usually, you have "urls.py" files in apps and project folders, but we are going to discuss them both because they are interconnected with each other.

As mentioned, Django uses the "urls.py" file in the project folder as a root URL configuration to look for the path that matches the web request. If it finds that the path’s name includes other paths in the app folder, it will look inside the app folder to find a complete match and then use the view inscribed on it. In this file, there are such paths; all use "include" from the "django.urls" library. Other ones are direct paths to the views or admin pages given to us from the start by Django.

```
from django.conf import settings
from django.conf.urls.static import static
from django.contrib.auth.views import LoginView, LogoutView,
PasswordResetView, PasswordResetDoneView, PasswordResetConfirmView,
PasswordResetCompleteView
from django.contrib import admin
from django.urls import path, include
from tickets.views import LandingPageView, SignupView ,
ActivateAccount
from administration.decorators import already_logged

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', LandingPageView.as_view(), name='landing-page'),
    path('tickets/', include('tickets.urls', namespace="tickets")),
    path('administration/', include('administration.urls',
namespace="administration")),
    path('dashboard/', include('dashboard.urls',
namespace="dashboard")),
    path('archive/', include('archive.urls', namespace="archive")),
    path('account/', include('account.urls', namespace="account")),
    path('notifications/', include('notifications.urls',
namespace="notifications")),
    path('login/',
LoginView.as_view(redirect_authenticated_user=True), name='login'),
    path('signup/', already_logged(SignupView.as_view()),
name='signup'),
    path('activation/<uidb64>/<token>/',
already_logged(ActivateAccount.as_view()), name='activate'),
    path('reset-password/',
already_logged(PasswordResetView.as_view()), name='reset-password'),
    path('password-reset-done/',
already_logged(PasswordResetDoneView.as_view()),
name='password_reset_done'),
    path('password-reset-confirm/<uidb64>/<token>/',
already_logged(PasswordResetConfirmView.as_view()),
name='password_reset_confirm'),
    path('password-reset-complete/',
already_logged(PasswordResetCompleteView.as_view()),
name='password_reset_complete'),
    path('logout/', LogoutView.as_view(), name='logout'),
]
if settings.DEBUG:
    urlpatterns += static(settings.STATIC_URL,
document_root=settings.STATIC_ROOT)
```

Figure 36 – “urls.py” inside the project folder code. Source: author.

As you can see in the picture, there are a lot of built-in views of the authentication process that was given by Django. These views are already functional, and they only need a template with the correct name so Django can find it. In them, such things as decorators are used that work the same way as mixins, and the decorator "already_logged" is used to prevent the user from authentication if a user is already logged in. You can also observe the name of the path with "<uidb64>/<token>/", where "uidb64" is two unique characters and "token" is a hash value that is useful when you want to create a one-time link to verify email that prevents a user from going to the "used" link. Inside the paths, there are such arguments as "namespace" and "name", where "namespace" is a unique way of identifying the application’s "urls.py" file, "name" also works as a unique identifier but for views, and both of them can be useful to create a link in HTML from one page to another. This file also has the condition to check if you are not in production, where you likely turn off debug mode because of malicious attacks that can be done on a server, and you can serve your static files.

```
from django.urls import path
from .views import *

app_name = "tickets"

urlpatterns = [
    path('', TicketListView.as_view(), name='ticket-list'),
    path('json/', TicketJsonView.as_view(), name='ticket-list-json'),
    path('create/', TicketCreateView.as_view(), name='ticket-
create'),
    path('type/', TypeListView.as_view(), name='type-list'),
    path('type/<int:pk>/', TypeDetailView.as_view(), name='type-
detail'),
    path('type-create/', TypeCreateView.as_view(), name='type-
create'),
    path('type/<int:pk>/update/', TypeUpdateView.as_view(),
name='type-update'),
    path('type/<int:pk>/delete/', TypeDeleteView.as_view(),
name='type-delete'),
    path('priority/', PriorityListView.as_view(), name='priority-
list'),
    path('priority/<int:pk>/', PriorityDetailView.as_view(),
name='priority-detail'),
    path('priority-create/', PriorityCreateView.as_view(),
name='priority-create'),
    path('priority/<int:pk>/update/', PriorityUpdateView.as_view(),
name='priority-update'),
    path('priority/<int:pk>/delete/', PriorityDeleteView.as_view(),
name='priority-delete'),
    path('status/', StatusListView.as_view(), name='status-list'),
    path('status/<int:pk>/', StatusDetailView.as_view(),
name='status-detail'),
    path('status-create/', StatusCreateView.as_view(), name='status-
create'),
    path('status/<int:pk>/update/', StatusUpdateView.as_view(),
name='status-update'),
    path('status/<int:pk>/delete/', StatusDeleteView.as_view(),
name='status-delete'),
    path('comments/<int:pk>/', CommentCreateView.as_view(),
name='ticket-comment-create'),
    path('comments/<int:pk>/delete/', CommentDeleteView.as_view(),
name='ticket-comment-delete'),
    path('<int:pk>/comments/update/', CommentUpdateView.as_view(),
name='ticket-comment-update'),
    path('<int:pk>/category/', TicketCategoryUpdateView.as_view(),
name='ticket-status-update'),
    path('<int:pk>/', TicketDetailView.as_view(), name='ticket-
detail'),
    path('<int:pk>/update/', TicketUpdateView.as_view(),
name='ticket-update'),
    path('<int:pk>/completed/', TicketCompletedView.as_view(),
name='ticket-completed'),
    path('<int:pk>/reopen/', TicketReopenView.as_view(),
name='ticket-reopen'),
    path('<int:pk>/delete/', TicketDeleteView.as_view(),
name='ticket-delete'),
    path('<int:pk>/request/change/',
TicketRequestChangeView.as_view(), name='ticket-request-change'),
    path('filter/', filter_, name='ticket-filter'),
]
```

Figure 37 – “urls.py” file inside the app folder code. Source: author.

The "urls.py" file inside the app folder looks similar to the project folder but has minor differences. You can see the "app_name" variable that says this file belongs to the apps folder. And inside the path name is the "<int: pk>", which tells Django that if you go to a primary key that is always a number, it will use an appropriate view to handle this request.

4.3.7 HTML templates

As you know, the main building block of any web application is the HTML template, which is very important to understand because it is the part that users can see and interact with. In this subchapter, we will touch on all the necessary details about HTML templates considering this project.

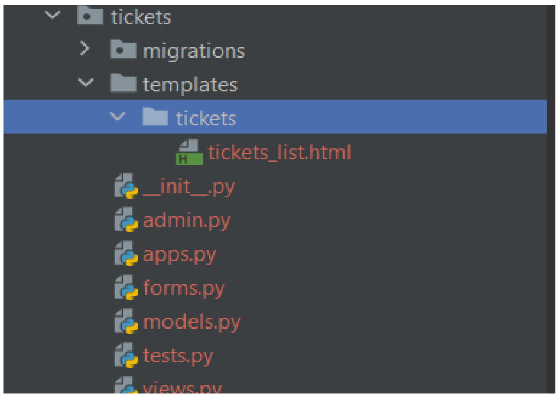


Figure 38 – Template folder example. Source: author

This step has already been done, so you can go to any template folder inside the app folder and open any file you want. First, you can notice that strange symbols like curly braces and percent signs are in the code because of the built-in Jinja extension in Django. It helps to divide HTML structure into the blocks of code in your base template and extend other HTML templates from it, load some static files or other packages, write conditions and loops, use objects from views, get object attributes, and so on.

Another thing that you can notice is that there is CSS inside HTML classes instead of CSS code in a separate file. It is possible by using the Tailwind CSS framework, which helps you write fast and easy CSS code inside an HTML file. To start working with it, you can copy the CDN link on the Tailwind CSS site to the base template. You can also use frameworks such as Bootstrap, which is a component-based framework, compared to Tailwind, which is a utility-first framework. But I would recommend you use Tailwind because, with Bootstrap, you have only prebuilt components that can cause many problems when changing code, while in Tailwind, you can easily change it.
All other code is simple HTML and JavaScript code, where JavaScript goes under the "scripts" block to filter what the user sees on the page and HTML is under the "content"

block that goes in hierarchal order by the rules of HTML5 starting from the extension of the base template.

4.4 Other folders and files

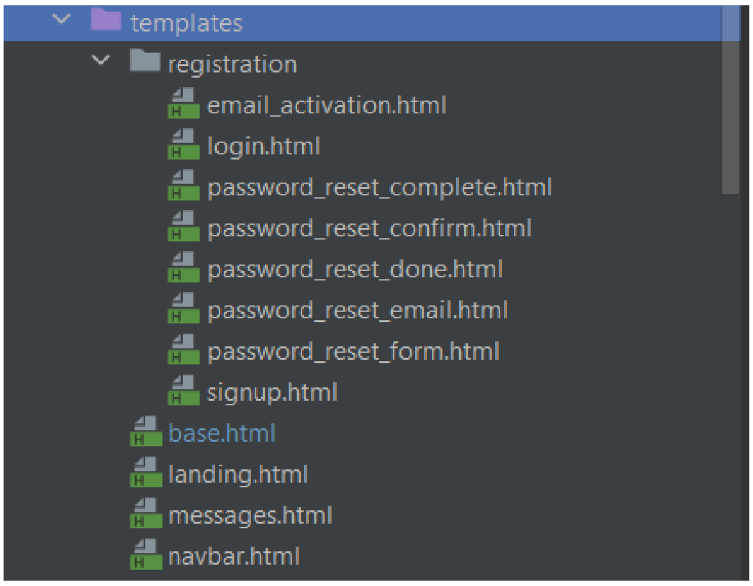
In the previous paragraphs, we discussed project and app folders, and now we will sort out what other folders and files are responsible for. The order is from top to bottom, starting with the "static" folder.

In the "static" folder, all of the files are static files, and they are called "static" because they never change through the pages. Static files are JavaScript, CSS, and images that are used throughout the project. The main idea is to put your static files in one folder and reference them in HTML templates.

```
STATIC_URL = '/static/'
STATICFILES_DIRS = [
    BASE_DIR / "static"
]
STATIC_ROOT = BASE_DIR / "staticfiles-cdn"
```

Figure 39 – Static configuration code. Source: author

To make it work, you must configure the path to the "static" folder in "settings.py". As you can see, there is "STATIC_URL", the default setting where the user can access your static files. "STATICFILES_DIRS = [BASE_DIR / "static"]" is a list of paths that you want to include for Django to recognize as folders where static files are going to be held. Another setting is "STATIC_ROOT = BASE_DIR / "staticfiles-cdn"". It is the root folder of all static files, and it is used to grab all of your static files that could be in multiple places into one folder when you are deploying. You can create such a file using the "python manage.py collectstatic" command in the terminal.



```
templates
    registration
        email_activation.html
        login.html
        password_reset_complete.html
        password_reset_confirm.html
        password_reset_done.html
        password_reset_email.html
        password_reset_form.html
        signup.html
    base.html
    landing.html
    messages.html
    navbar.html
```

Figure 40 – Main templates folder. Source: author.

The following folder is the "templates" folder, where all templates for registration, messages, and the base template for this project are situated. Inside the "registration" folder, there are "email_activation.html" and "password_reset_email.html", which are just messages sent by email to go to the link to confirm the identity of a user. "password_reset_complete.html" and "password_reset_done.html" are pages that tell the user about the completion of the password reset process. All other templates are forms the user must fill out to be authorized. Then there is "navbar.html", which is part of "base.html", that has a lot of "if" statements for different roles and statuses of authentication plus the project logo. "messages.html" is also part of "base.html" and is triggered by views sending some type of message with the request and the message itself that creates a box with some colored text depending on the message type to inform the user about the status of registration. "landing.html" is our project’s first page, which contains congratulations, a picture, and a button to log in. And "base.html" is the template from which every other template extends; it contains the name of the project, has a link to the Tailwind CSS, includes "navbar.html" and "messages.html", divides the template into two blocks of HTML and JS code, and has basic HTML structure.

You’ve already encountered the "venv" folder at the beginning when you made a starter project. The virtual environment is used to store all dependencies that you need for your project in one folder.

"gitignore" is a file that is useful when you are deploying your project on GitHub, but you want to prevent some files from being deployed, so you write the name of a file or a folder inside this file, and they will be ignored in deployment. When creating a new repository in GitHub, you can choose the language you will write in as an option.

"manage.py" is the primary file name in the command line when we want to appeal to Django to do something inside our project.

The "README.md" file connects to GitHub because it allows users to write about and explain this project's repository. You can create this file the same way as "gitignore". You need to pick this option when you create a new repository.

And the last one is the "requirement.txt" file, which is a blueprint of all dependencies that can be useful when deploying your project on a host server. You can write all new requirements from the virtual environment using "pip freeze > requirements.txt".

4.5 Installations and configurations

In this project, there are a lot of programs that would not be on my local server if I did not install them, and they would not work if I did not configure them right. So, this chapter is dedicated to understanding this process. In the "requirements.txt" file, we will only overview the dependencies installed and used in this project.

One small remark: to install the package, you need to write "pip install" and the name of the package, and at the end of all installations, you should "pip freeze > requirements.txt" to create a blueprint of dependencies in the "requirements.txt" file.

The first one is the installation of "Django", which allows us to communicate with the Django framework, use its libraries, and work with its architecture.

The next one is "django-crispy-forms", which does work with the forms much more effortlessly. You need to install it and state it in the project settings. Inside "INSTALLED_APPS", you must write "'crispy_forms'" to make it work.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'whitenoise.runserver_nostatic',
    'django.contrib.staticfiles',
    'crispy_forms',
    'crispy_tailwind',
    'tailwind',
    'tickets',
    'administration',
    'dashboard',
    'account',
    'archive',
    'notifications',
]
```

Figure 41 – Installed apps code. Source: author.

Once you do that, you need to decide on a template pack, which is essentially a set of stylings for how the form will be styled. In my case, I used Tailwind CSS to style my templates, so I installed "crispy-tailwind" to use crispy forms for Tailwind. But you also need to configure it in your settings by writing "'crispy_tailwind'" inside "INSTALLED_APPS" and adding two variables.

Configure crispy forms in your settings file using the following code.

```
CRISPY_ALLOWED_TEMPLATE_PACKS = "tailwind"
CRISPY_TEMPLATE_PACK = 'tailwind'
```

Figure 42 – Crispy forms config code. Source: author.

"django-environ" is a package that ciphers your settings variables that should be kept secret by writing all of them into a ".env" file. Then, if you deploy your project, for example, to GitHub, it will be ignored by the "gitignore" file, and all settings that should be confidential will not leak.

I installed "psycopg2-binary" to change my default SQLite database to PostgreSQL, which is more suitable for production, and it also needs to be configured in the settings file, which you will do in the next chapter.

"gunicorn" is a WSGI server that you use in production to send requests from the reverse proxy to the Django project and vice versa.

There is also such a package as "whitenoise", which helps us serve static files in production because Django itself doesn't have any solution to this problem. You will see how to configure it as we prepare to deploy our project.

4.6 Getting ready to deploy

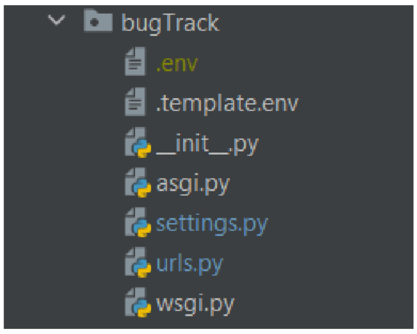
Before even considering deploying on DigitalOcean, you should go through the basic preparation steps. You will install third-party packages and configure settings that will allow you to deploy your Django project safely. This procedure protects confidential information or makes some parts of a project ready for production. And "push" a project inside the GitHub repository, from where you will clone it on a host server.

First, you can see such variables inside the "settings.py" file as "SECRET_KEY", "DEBUG", and other sensitive values that are stored there, which you do not want to be in the actual code but inside the ".env" file. All variables inside the ".env" file are environment variables, like in the "venv" file, but not packages, but variables you can access before running the server. To create this file, you must install "django-environ" using the command "pip install django-environ" in our terminal. Then you make the ".env" file inside the project folder, and you can copy your "SECRET_KEY" and "DEBUG" variables inside this file. After that, you need to code inside "settings.py", as illustrated in a picture.

```
import environ


env = environ.Env(
    DEBUG=(bool, False)
)

READ_DOT_ENV_FILE = env.bool('READ_DOT_ENV_FILE', default=False)
if READ_DOT_ENV_FILE:
    environ.Env.read_env()

DEBUG = env('DEBUG')
SECRET_KEY = env('SECRET_KEY')
```

Figure 43 – ".env" file config code. Source: author

Here you can witness that the "environ" was imported and the "DEBUG" was cast a value of "False" by default. After that, the "READ_DOT_ENV_FILE" is also by default "False", but you can export it using "export READ_DOT_ENV_FILE =True" in your session, and by doing that, you answer the condition to read the ".env" file, and then you ask for variables with the appropriate names inside the ".env" file.



```
bugTrack
    .env
    .template.env
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
```

Figure 44 – ".env" file. Source: author

As you can see, the ".env" file is colored in yellow, meaning that GitHub will ignore it due to the "gitignore" file because you need those variables only for local development. But in production, you will have different values. There is also a ".template.env" file that lists variables without values that a user should have inside the ".env" file. If the user clones a repository from GitHub, for example, and the user wants to know the environment variables that this project needs.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': env("DB_NAME"),
        'USER': env("DB_USER"),
        'PASSWORD': env("DB_PASSWORD"),
        'HOST': env("DB_HOST"),
        'PORT': env("DB_PORT")
    }
}
```

Figure 45 – Database config code. Source: author.

Next, you need to change the SQLite database to PostgreSQL in the "settings.py" file to prepare the code for production. To accomplish this, you can comment SQLite configurations in "settings.py" and write the same code as on the picture.

```
DEBUG=True
SECRET_KEY=1234
DB_NAME=
DB_USER=postgres
DB_PASSWORD=
DB_HOST=localhost
DB_PORT=
```

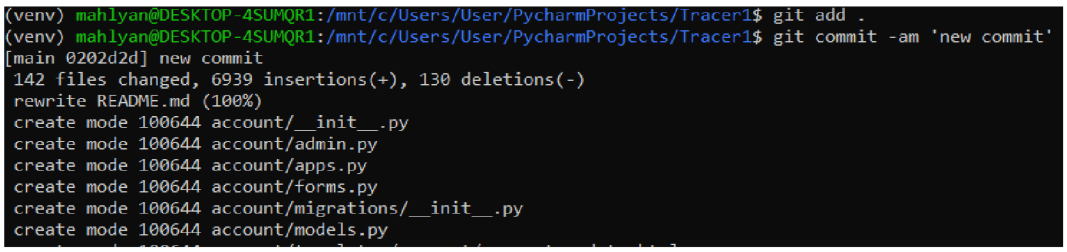
Figure 46 – “.template.env” file code. Source: author.

Install "psycopg2-binary" using the "pip install" command. Then you need to write all variable names in the ".template.env" file. After we "push" changes on GitHub, you can comment on PostgreSQL configurations and work with SQLite in local development.

```
STATICFILES_STORAGE =
'whitenoise.storage.CompressedManifestStaticFilesStorage'
```

Figure 47 – WhiteNoise configuration code. Source: author.

The last thing you want to configure before you deploy your project on a GitHub server is static file configuration using the package "whitenoise", which helps serve static files on a host server. Start with installing "whitenoise" by issuing the "pip install whitenoise" command. Then you need to add "whitenoise" to your "MIDDLEWARE" in your "settings.py" that will allow "whitenoise" to handle requests from the server. Then you should specify the "STATICFILES_STORAGE" value, which points to "whitenoise". You also need to add "‘whitenoise.runserver_nostatic’" inside "INSTALLED_APPS" to use WhiteNoise in development as well as in production.



```
(venv) mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$ git add .
(venv) mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$ git commit -am 'new commit'
[main 0202d2d] new commit
 142 files changed, 6939 insertions(+), 130 deletions(-)
 rewrite README.md (100%)
 create mode 100644 account/__init__.py
 create mode 100644 account/admin.py
 create mode 100644 account/apps.py
 create mode 100644 account/forms.py
 create mode 100644 account/migrations/__init__.py
 create mode 100644 account/models.py
```

Figure 48 – Committing changes to GitHub. Source: author.

After completing all those steps, you can go to your terminal inside the root project folder. Here you will write "git add," which adds all changes compared to your repository on GitHub. Then you will write "git commit -am ‘new commit’ ", which creates a commit on the branch you are in with its name. And then you can deploy using "git push" with a path to your repository on GitHub and a safety token that you need to set up in your GitHub account.

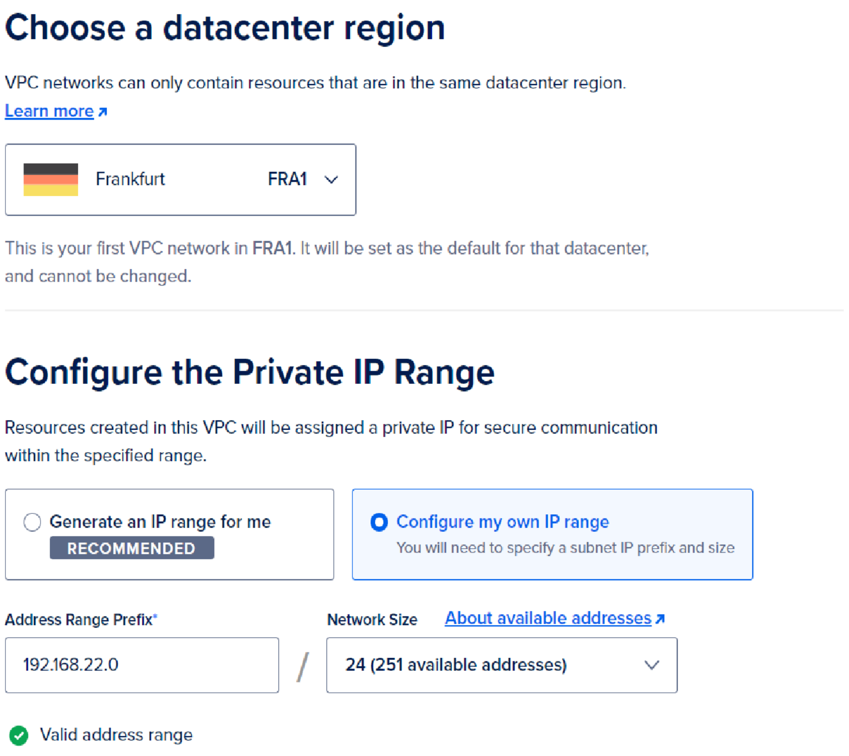
Now, you can go to your GitHub repository and see that you have new folders and files that you posted.

4.7 Deploying on Digital Ocean

This is the final chapter of the practical part, where we will discuss all the necessary steps to deploy the Django project on DigitalOcean. You will learn to work with the DigitalOcean interface, set up the PostgreSQL database, and configure the Gunicorn/Nginx/Django server.

To begin with, you need to register on DigitalOcean by using your email and specifying the payment method. Proceed to the main page, and from there, we will begin.

The first thing typical in deployment is creating a VPC network to make a private network for your server and database so you can be sure that traffic is not sniffed or sent to the public internet. Press the "Networking" tab on the sidebar, go to "VPC," and press "Create VPC Network". You will be provided with the following form. Fill in the fields as shown in the picture and create your VPC network.



Choose a datacenter region

VPC networks can only contain resources that are in the same datacenter region.
Learn more ↗

Frankfurt FRA1

This is your first VPC network in FRA1. It will be set as the default for that datacenter, and cannot be changed.

Configure the Private IP Range

Resources created in this VPC will be assigned a private IP for secure communication within the specified range.

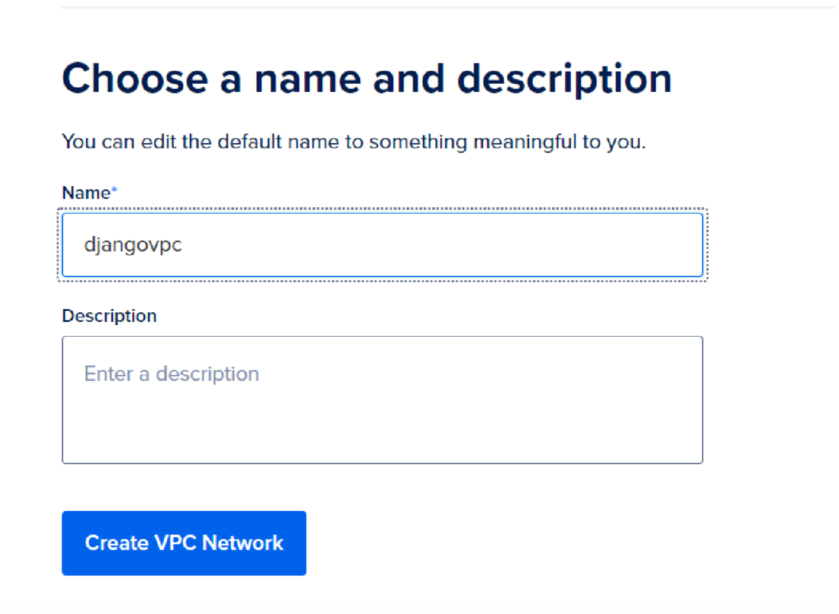
○ Generate an IP range for me RECOMMENDED [✓ Configure my own IP range] You will need to specify a subnet IP prefix and size

Address Range Prefix*	192.168.22.0
Network Size	24 (251 available addresses)

About available addresses ↗

✓ Valid address range

Figure 49 – VPC form 1. Source: author.



Choose a name and description

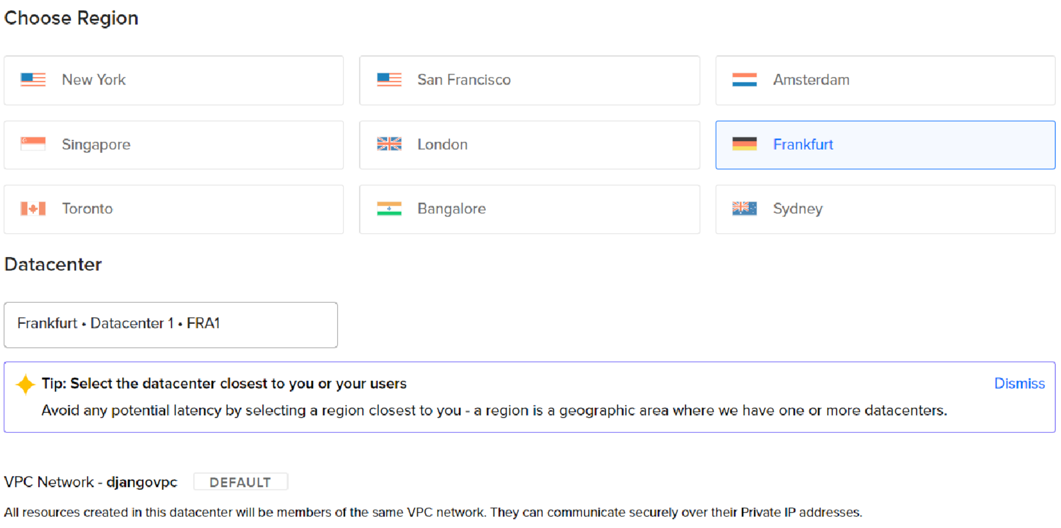
You can edit the default name to something meaningful to you.

Name*	djangovpc
Description	Enter a description

Create VPC Network

Figure 50 – VPC form 2. Source: author.

After you create your own private network, you can open the section called "Droplet," where you will create your server or Linux-based virtual machine; in our case, it will be an Ubuntu server. Press "Create Droplet". It will render a form where you will specify all parameters for this droplet, and after you are done, press "Create Droplet".



Choose Region

New York	San Francisco	Amsterdam
Singapore	London	[✓ Frankfurt]
Toronto	Bangalore	Sydney

Datacenter

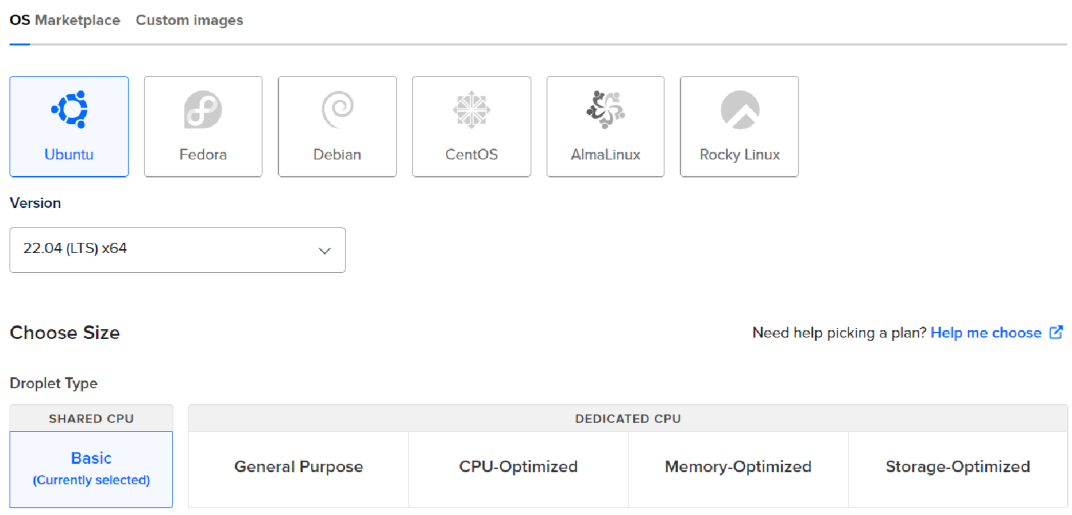
Frankfurt • Datacenter 1 • FRA1

Tip: Select the datacenter closest to you or your users
Avoid any potential latency by selecting a region closest to you - a region is a geographic area where we have one or more datacenters.

VPC Network - djangovpc DEFAULT

All resources created in this datacenter will be members of the same VPC network. They can communicate securely over their Private IP addresses.

Figure 51 – Droplet form 1. Source: author.



OS Marketplace Custom images

[✓ Ubuntu] Fedora Debian CentOS AlmaLinux Rocky Linux

Version

22.04 (LTS) x64

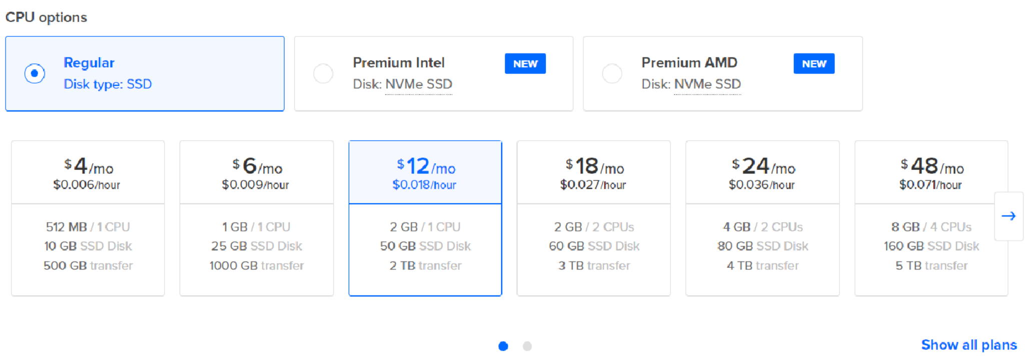
Choose Size

Need help picking a plan? Help me choose

Droplet Type

SHARED CPU	DEDICATED CPU			
[✓ Basic (Currently selected)]	General Purpose	CPU-Optimized	Memory-Optimized	Storage-Optimized

Figure 52 – Droplet form 2. Source: author.



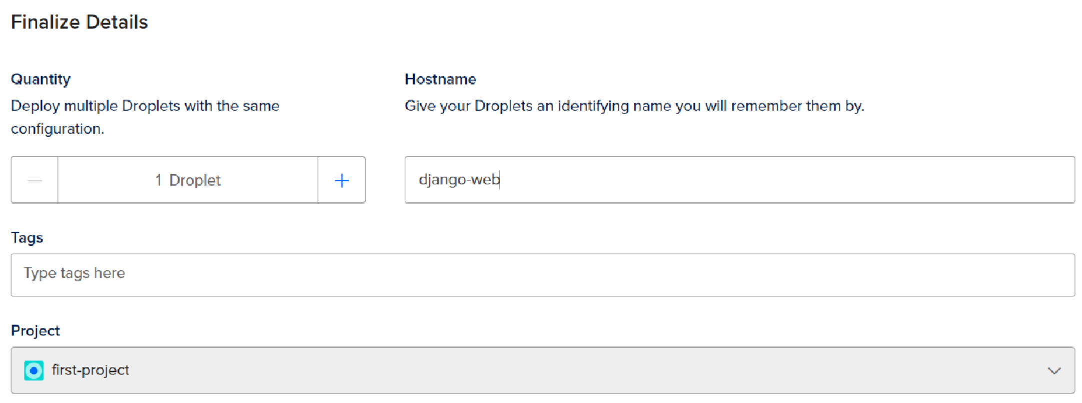
CPU options

[✓ Regular Disk type: SSD] Premium Intel NEW Disk: NVMe SSD Premium AMD NEW Disk: NVMe SSD

$4/mo $0.006/hour	$6/mo $0.009/hour	[✓ $12/mo $0.018/hour]	$18/mo $0.027/hour	$24/mo $0.036/hour	$48/mo $0.071/hour
512 MB / 1 CPU 10 GB SSD Disk 500 GB transfer	1 GB / 1 CPU 25 GB SSD Disk 1000 GB transfer	2 GB / 1 CPU 50 GB SSD Disk 2 TB transfer	2 GB / 2 CPUs 60 GB SSD Disk 3 TB transfer	4 GB / 2 CPUs 80 GB SSD Disk 4 TB transfer	8 GB / 4 CPUs 160 GB SSD Disk 5 TB transfer

Show all plans

Figure 53 – Droplet form 3. Source: author.



Finalize Details

Quantity
Deploy multiple Droplets with the same configuration.

— 1 Droplet +

Hostname
Give your Droplets an identifying name you will remember them by.

django-web

Tags

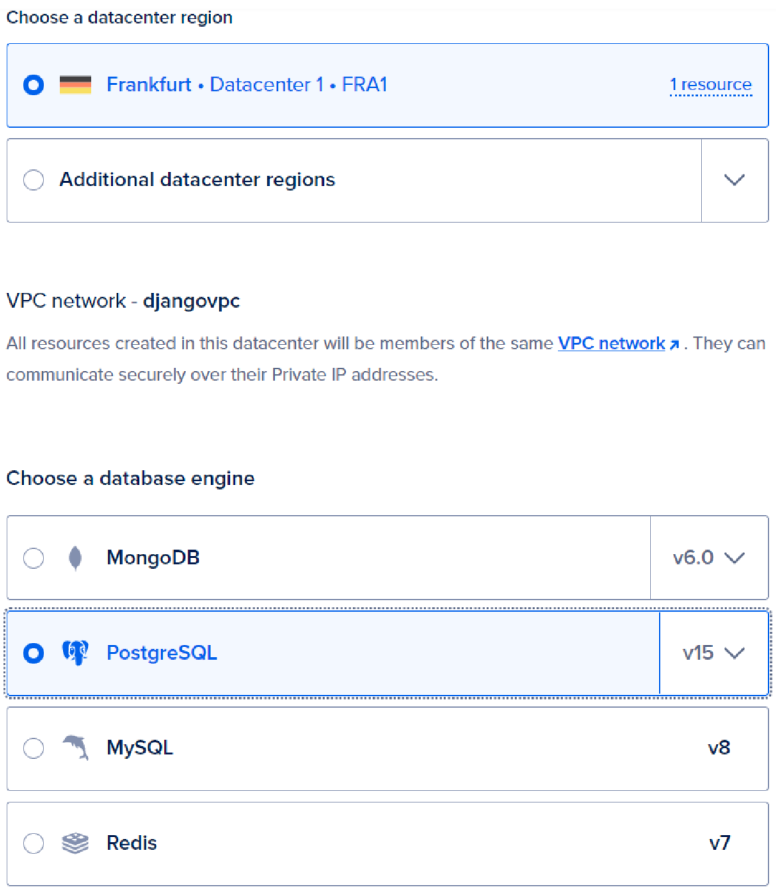
Type tags here

Project

first-project

Figure 54 – Droplet form 4. Source: author.

Next, instead of using SQLite as was done locally, we can play with the DigitalOcean managed database offering, where, as was said earlier, you will create a PostgreSQL cluster inside the VPC network so the communication between the "droplet" and database will not go out over the public Internet. It will not cost us our public Internet bandwidth. Press the green button at the top right corner and go to the "Databases" option. There you will configure and create your database, as illustrated in a picture. Typically, a database cluster takes four to six minutes to be created, so do not get frustrated, and once it is done, you will continue to configure the database.


Choose a datacenter region

[✓ Frankfurt • Datacenter 1 • FRA1] 1 resource

Additional datacenter regions

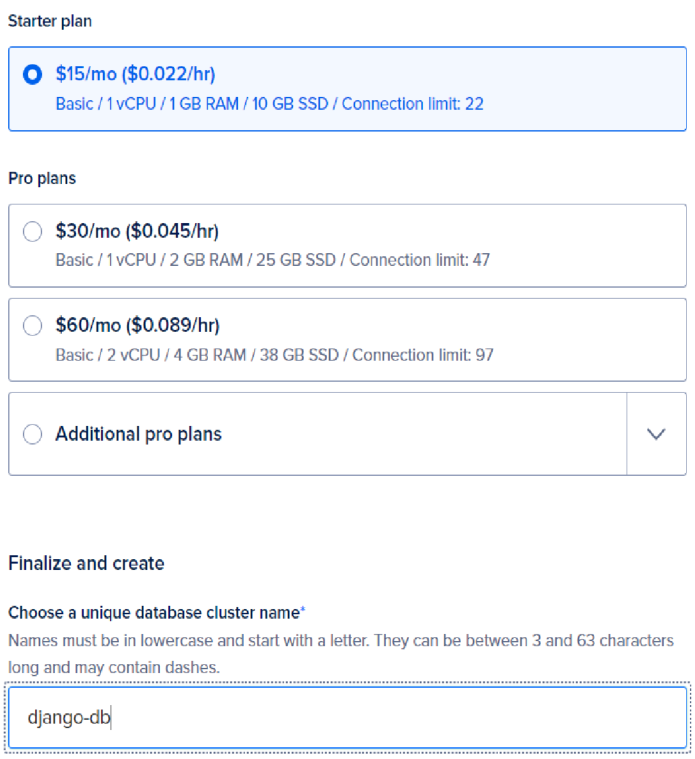
VPC network - **djangovpc**

All resources created in this datacenter will be members of the same VPC network. They can communicate securely over their Private IP addresses.

Choose a database engine

MongoDB	v6.0
[✓ PostgreSQL]	v15
MySQL	v8
Redis	v7

Figure 55 – Database form 1. Source: author.


Starter plan

[✓ $15/mo ($0.022/hr)
Basic / 1 vCPU / 1 GB RAM / 10 GB SSD / Connection limit: 22]

Pro plans

$30/mo ($0.045/hr)
Basic / 1 vCPU / 2 GB RAM / 25 GB SSD / Connection limit: 47

$60/mo ($0.089/hr)
Basic / 2 vCPU / 4 GB RAM / 38 GB SSD / Connection limit: 97

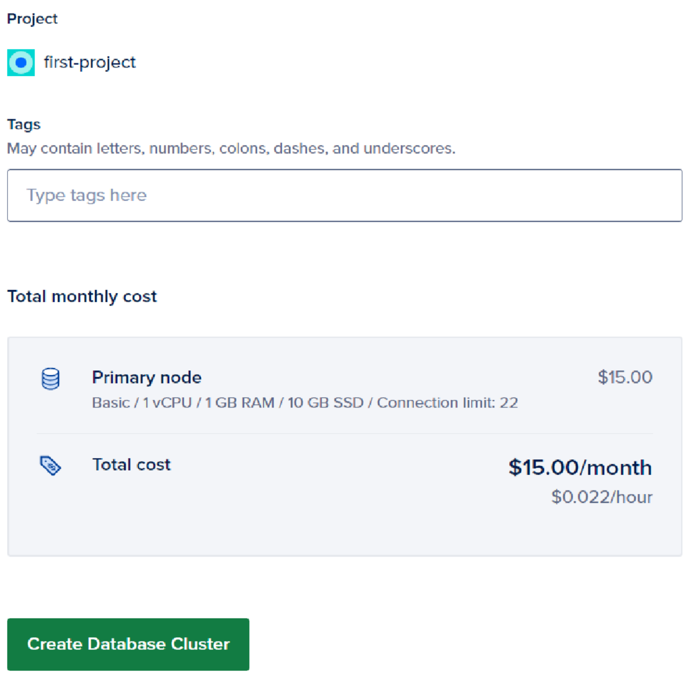
Additional pro plans

Finalize and create

Choose a unique database cluster name*
Names must be in lowercase and start with a letter. They can be between 3 and 63 characters long and may contain dashes.

django-db

Figure 56 – Database form 2. Source: author.


Project

[✓ first-project]

Tags
May contain letters, numbers, colons, dashes, and underscores.

Type tags here

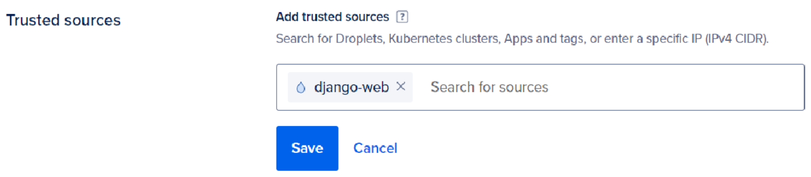
Total monthly cost

Primary node $15.00
Basic / 1 vCPU / 1 GB RAM / 10 GB SSD / Connection limit: 22

Total cost $15.00/month
$0.022/hour

Figure 57 – Database form 3. Source: author.

When the database is done provisioning, you need to secure it. As you can see, it is positioned inside the VPC network, which is correct, but it is open to all incoming connections, so you need to secure this database by restricting access. And the way you can do that is by selecting your "django-web" source, which allows traffic to come in only from that "droplet".


Trusted sources

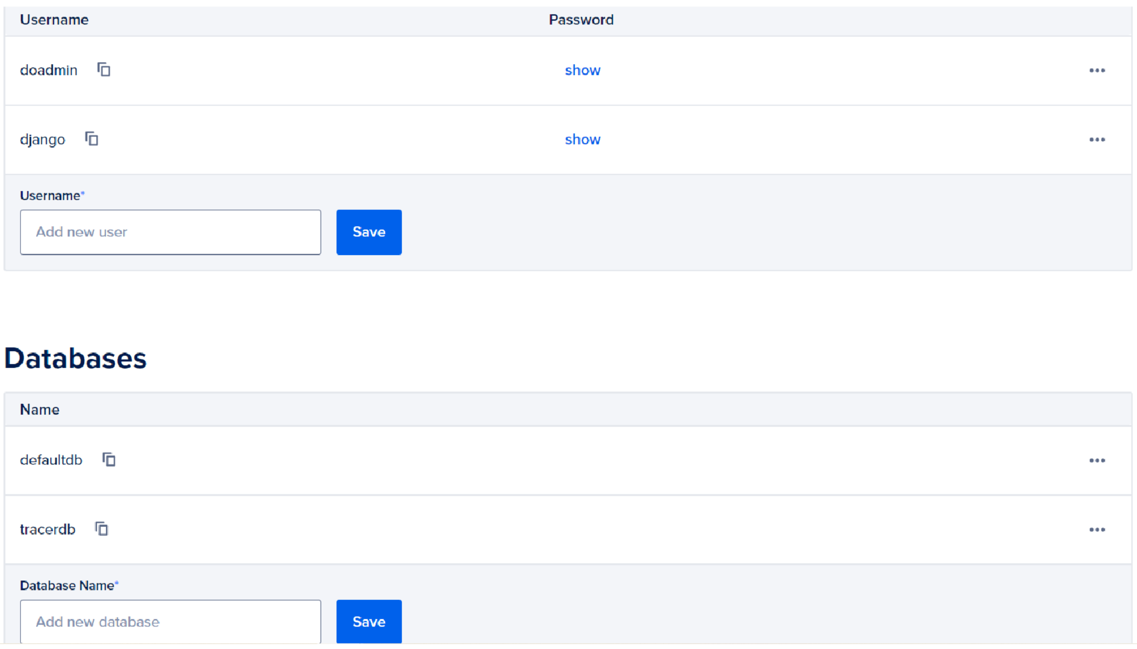
Add trusted sources

Search for Droplets, Kubernetes clusters, Apps and tags, or enter a specific IP (IPv4 CIDR).

django-web × | Search for sources

Figure 58 – Adding trusted sources. Source: author.

Another thing you want to do is create a new database and a new user for this database, so you need to open the "Users & Databases" tab and name your user and database as you wish. New user "django" and "tracerdb" database were created.


Username	Password
doadmin	show
django	show

Username*: Add new user

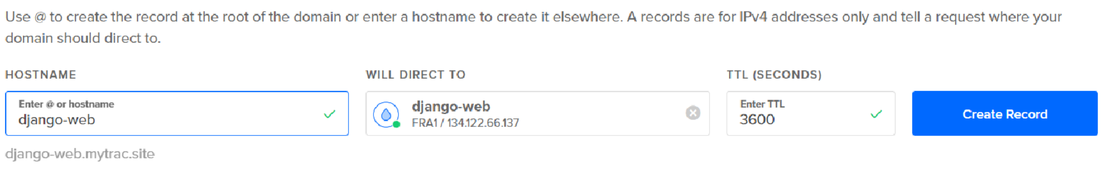
Databases

Name
defaultdb
tracerdb

Database Name*: Add new database

Figure 59 – Database and user creation. Source: author.

The last thing you need to do inside DigitalOcean is create a DNS record for your project so you can secure your app via HTTPS using "Certbot". To register your DNS on DigitalOcean, you write your domain name and copy all NS from DigitalOcean into the field "Nameservers" in your domain registrar. Then you will create a subdomain where your project will be situated. Write your "Hostname" and pick your "droplet" as "Will Direct To" and create your record.


Use @ to create the record at the root of the domain or enter a hostname to create it elsewhere. A records are for IPv4 addresses only and tell a request where your domain should direct to.

HOSTNAME: Enter @ or hostname: django-web

django-web.mytrac.site

WILL DIRECT TO: django-web FRA1 / 134.122.66.137

TTL (SECONDS): Enter TTL: 3600

Figure 60 – Registration of subdomain. Source: author.

Now you can open your terminal and execute "ssh root@<hostname>".

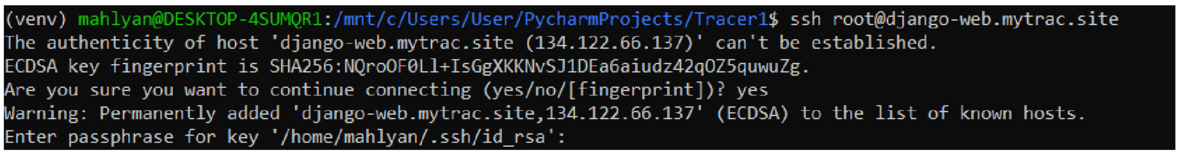

```
(venv) mahlyan@DESKTOP-4SUMQR1:/mnt/c/Users/User/PycharmProjects/Tracer1$ ssh root@django-web.mytrac.site
The authenticity of host 'django-web.mytrac.site (134.122.66.137)' can't be established.
ECDSA key fingerprint is SHA256:NQroOF0Ll+IsGgXKKNvSJ1DEa6aiudz42qOZ5quwuZg.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'django-web.mytrac.site,134.122.66.137' (ECDSA) to the list of known hosts.
Enter passphrase for key '/home/mahlyan/.ssh/id_rsa':
```

Figure 61 – Entering your server. Source: author.

You will be located at the root of your server. You will first create a user because it is not preferred to run everything as the root user constantly. You can enter just a username and password; all other fields are not needed.

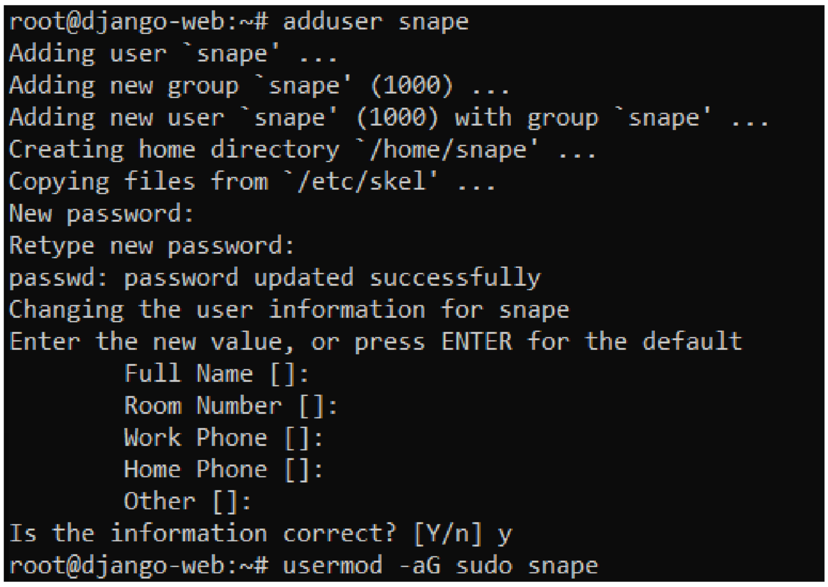
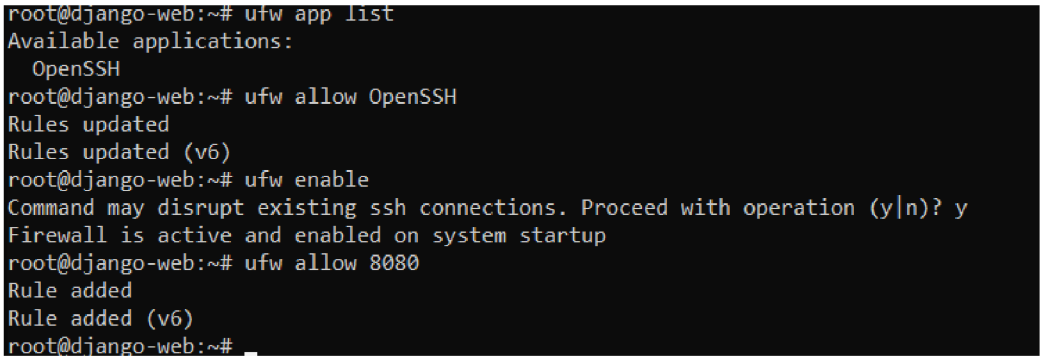

```
root@django-web:~# adduser snape
Adding user `snape' ...
Adding new group `snape' (1000) ...
Adding new user `snape' (1000) with group `snape' ...
Creating home directory `/home/snape' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for snape
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
root@django-web:~# usermod -aG sudo snape
```

Figure 62 – Creating a user. Source: author.

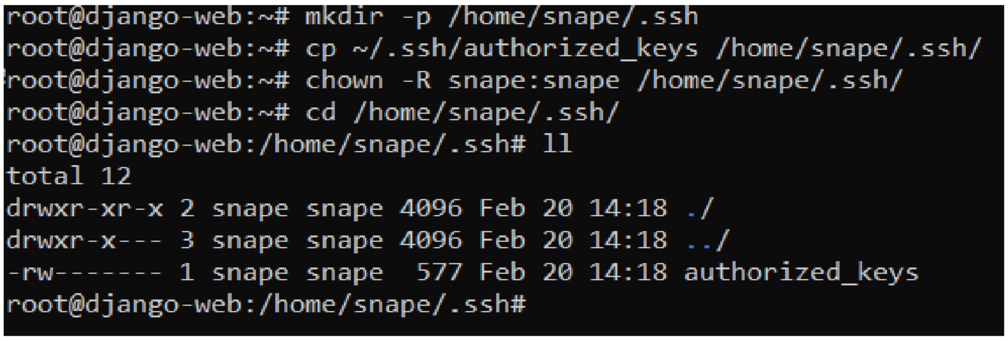
The next thing you will do is set up your firewall. Via commands on a picture, you will allow connections on SSH and 8080 ports.



```
root@django-web:~# ufw app list
Available applications:
  OpenSSH
root@django-web:~# ufw allow OpenSSH
Rules updated
Rules updated (v6)
root@django-web:~# ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
root@django-web:~# ufw allow 8080
Rule added
Rule added (v6)
root@django-web:~# _
```

Figure 63 – Opening ports. Source: author.

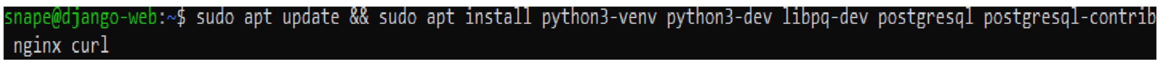
If you want to SSH in as "snape" using the same key you used as root, you first need to create a dot SSH directory in "snape's" home directory, copy the authorized keys file from yours into "snape's", and then give ownership of this directory to "snape".



```
root@django-web:~# mkdir -p /home/snape/.ssh
root@django-web:~# cp ~/.ssh/authorized_keys /home/snape/.ssh/
root@django-web:~# chown -R snape:snape /home/snape/.ssh/
root@django-web:~# cd /home/snape/.ssh/
root@django-web:/home/snape/.ssh# ll
total 12
drwxr-xr-x 2 snape snape 4096 Feb 20 14:18 ./
drwxr-x--- 3 snape snape 4096 Feb 20 14:18 ../
-rw------- 1 snape snape  577 Feb 20 14:18 authorized_keys
root@django-web:/home/snape/.ssh#
```

Figure 64 – Giving ownership over directory. Source: author.

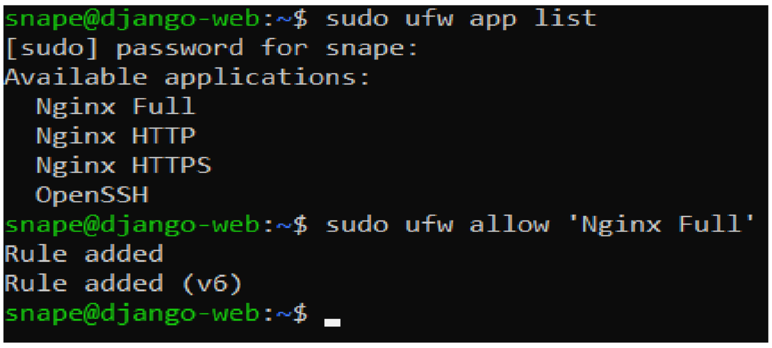
Now you should be able to use your SSH key as the "ssh snape@django-web.mytrac.site" command, and after you SSH in, you will make some installations. There will be an installed virtual environment for Python, Python files to build Gunicorn, PostgreSQL, and libraries connected with it and the NGINX web server.



```
snape@django-web:~$ sudo apt update && sudo apt install python3-venv python3-dev libpq-dev postgresql postgresql-contrib
 nginx curl
```

Figure 65 – Installations. Source: author.

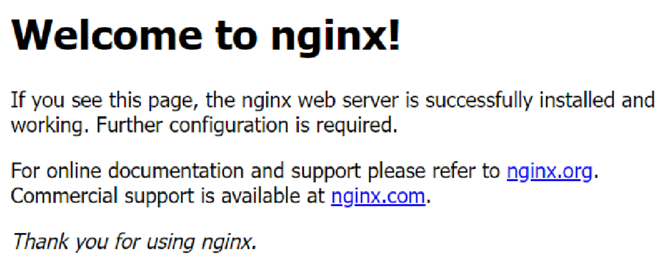
After you are done installing packages, you can go to your browser and type in the host server name, which will not work because you have not enabled NGINX yet. So, you need to input the following commands.



```
snape@django-web:~$ sudo ufw app list
[sudo] password for snape:
Available applications:
  Nginx Full
  Nginx HTTP
  Nginx HTTPS
  OpenSSH
snape@django-web:~$ sudo ufw allow 'Nginx Full'
Rule added
Rule added (v6)
snape@django-web:~$
```

Figure 66 – Allowing NGINX app. Source: author.

You will be provided with the following page when you open your browser and go to your host server.



Welcome to nginx!

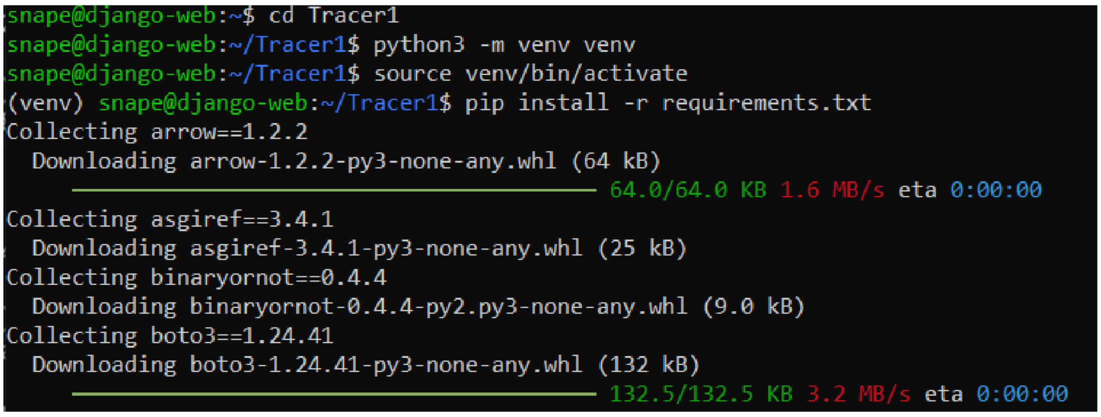
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Figure 67 – Welcome screen. Source: author.

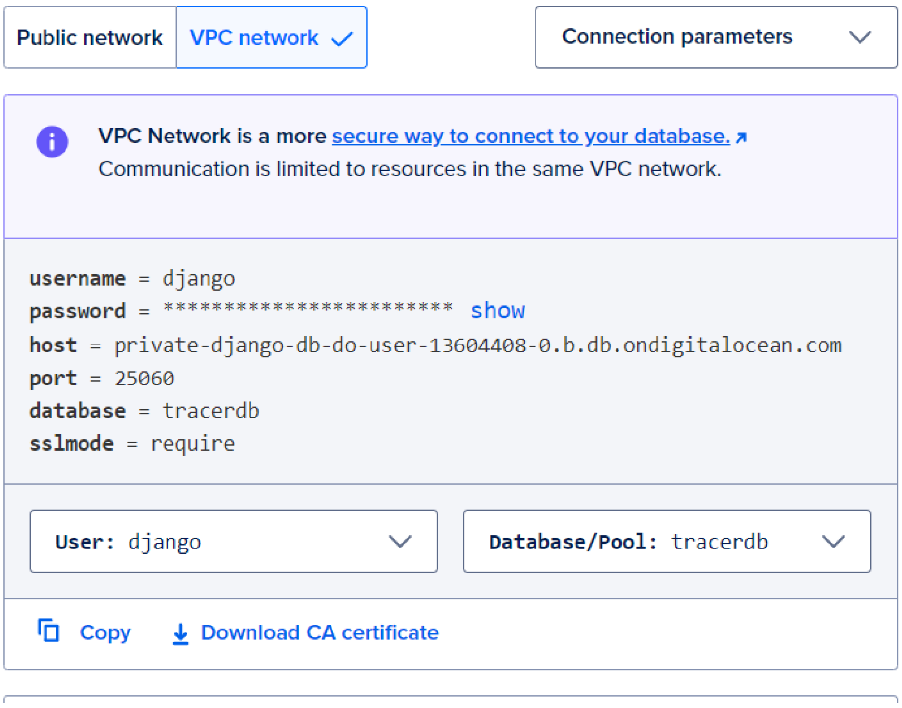
After you are reassured that NGINX is working, you can stop it using the "sudo systemctl stop nginx" command. Next, you will download the code from GitHub onto your server by writing "git clone https://github.com/Matteralmar/Tracer1.git". Now, you can "cd" into your project, create a virtual environment using it, activate it, and install all dependencies listed in the "requirements.txt" file.



```
snape@django-web:~$ cd Tracer1
snape@django-web:~/Tracer1$ python3 -m venv venv
snape@django-web:~/Tracer1$ source venv/bin/activate
(venv) snape@django-web:~/Tracer1$ pip install -r requirements.txt
Collecting arrow==1.2.2
  Downloading arrow-1.2.2-py3-none-any.whl (64 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 64.0/64.0 KB 1.6 MB/s eta 0:00:00
Collecting asgiref==3.4.1
  Downloading asgiref-3.4.1-py3-none-any.whl (25 kB)
Collecting binaryornot==0.4.4
  Downloading binaryornot-0.4.4-py2.py3-none-any.whl (9.0 kB)
Collecting boto3==1.24.41
  Downloading boto3-1.24.41-py3-none-any.whl (132 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 132.5/132.5 KB 3.2 MB/s eta 0:00:00
```

Figure 68 – Making installations to virtual environment. Source: author.

Moving forward, you will set up proper variables in the ".env" file for your project to be configured correctly. You can open the ".env" file using the "nano" text editor by executing "sudo nano bugTrack/.env". There, you should change the "DEBUG" value to "False" and the "SECRET_KEY" value to your local project secret key. Then you need to go back to DigitalOcean and copy all the details about your database on the VPC network into the environment file.



VPC Network is a more secure way to connect to your database.
Communication is limited to resources in the same VPC network.

```
username = django
password = ************************ show
host = private-django-db-do-user-13604408-0.b.db.ondigitalocean.com
port = 25060
database = tracerdb
sslmode = require
```

User: django
Database/Pool: tracerdb

Figure 69 – Database details. Source: author.

Next, you will activate your virtual environment, export "READ_DOT_ENV_FILE", collect static files, and run our server on port 0.0.0.0:8080.

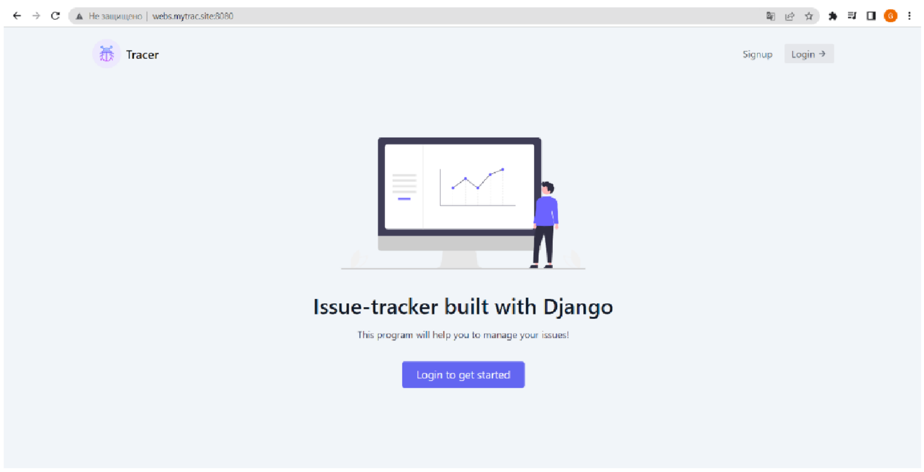
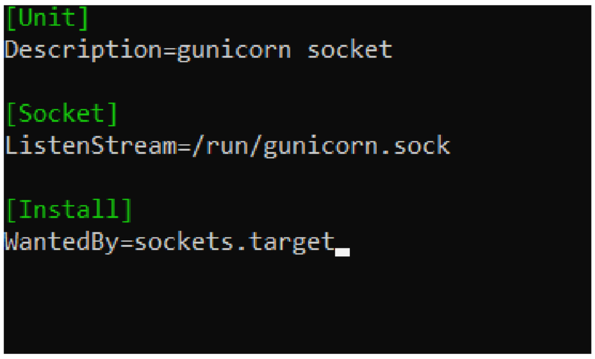

Не защищено | webs.mytrac.site:8080
Tracer
Signup
Login
Issue-tracker built with Django
This program will help you to manage your issues!
Login to get started

Figure 70 – Rendered page. Source: author.

Now that you know that your site can run on your server, the next thing you will do is set up your server to run as a "systemd" service using Nginx and Gunicorn. As you no longer need to use this port, you can deactivate your virtual environment using "deactivate" and close port 8080 by using "sudo ufw deny 8080".

Go ahead and create the Gunicorn socket file using the "sudo nano /etc/systemd/system/gunicorn.socket" command. Then you can wire a code as shown in the picture.

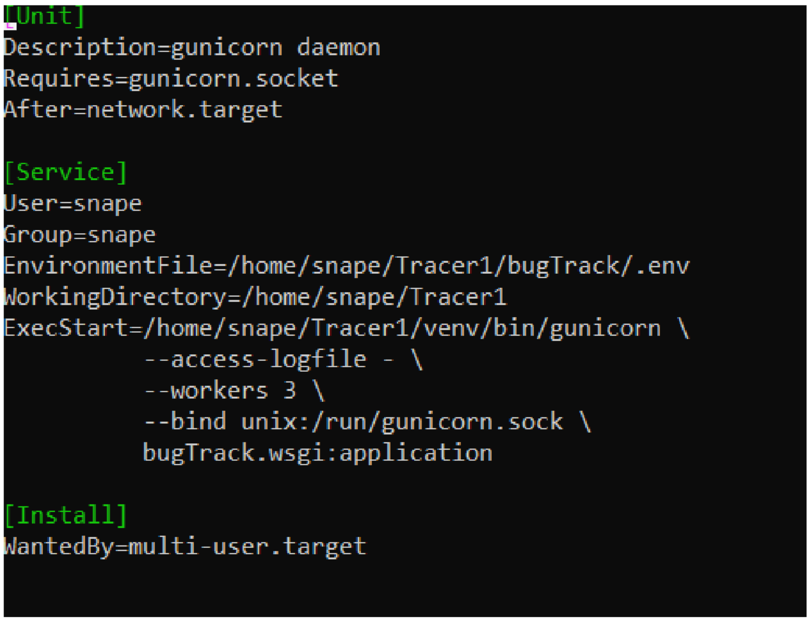


```
[Unit]
Description=gunicorn socket

[Socket]
ListenStream=/run/gunicorn.sock

[Install]
WantedBy=sockets.target
```

Figure 71 – Gunicorn socket file. Source: author.

This file gives the description of a socket in the "[Unit]" section, defines the socket location in the "[Socket]" section, and makes sure that the socket is created at the right time in the "[Install]" section.

The last thing you will configure about Gunicorn is "gunicorn.service", so run "sudo nano /etc/systemd/system/gunicorn.service" in your terminal and copy the code from the picture below.

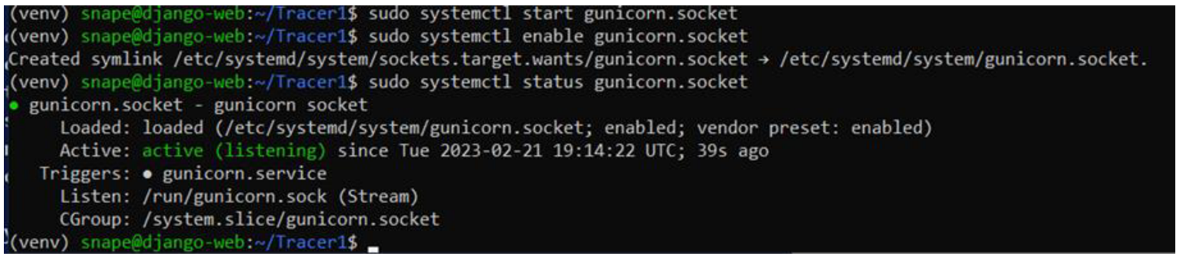


```
[Unit]
Description=gunicorn daemon
Requires=gunicorn.socket
After=network.target

[Service]
User=snape
Group=snape
EnvironmentFile=/home/snape/Tracer1/bugTrack/.env
WorkingDirectory=/home/snape/Tracer1
ExecStart=/home/snape/Tracer1/venv/bin/gunicorn \
          --access-logfile - \
          --workers 3 \
          --bind unix:/run/gunicorn.sock \
          bugTrack.wsgi:application

[Install]
WantedBy=multi-user.target
```

Figure 72 – Gunicorn service file. Source: author.

The logic behind these two files is that the socket will sit there and start up at boot and listen for traffic. If the traffic comes in, it will immediately start a service, and the service will provide everything you need.

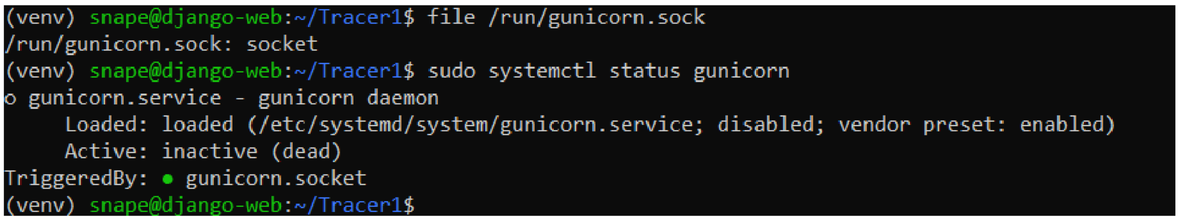
Now, you can start and enable the Gunicorn socket, and you should get the following status. And as you can see, it is running.



```
(venv) snape@django-web:~/Tracer1$ sudo systemctl start gunicorn.socket
(venv) snape@django-web:~/Tracer1$ sudo systemctl enable gunicorn.socket
Created symlink /etc/systemd/system/sockets.target.wants/gunicorn.socket → /etc/systemd/system/gunicorn.socket.
(venv) snape@django-web:~/Tracer1$ sudo systemctl status gunicorn.socket
● gunicorn.socket - gunicorn socket
     Loaded: loaded (/etc/systemd/system/gunicorn.socket; enabled; vendor preset: enabled)
     Active: active (listening) since Tue 2023-02-21 19:14:22 UTC; 39s ago
   Triggers: ● gunicorn.service
     Listen: /run/gunicorn.sock (Stream)
     CGroup: /system.slice/gunicorn.socket
(venv) snape@django-web:~/Tracer1$
```

Figure 73 – Gunicorn socket status. Source: author.

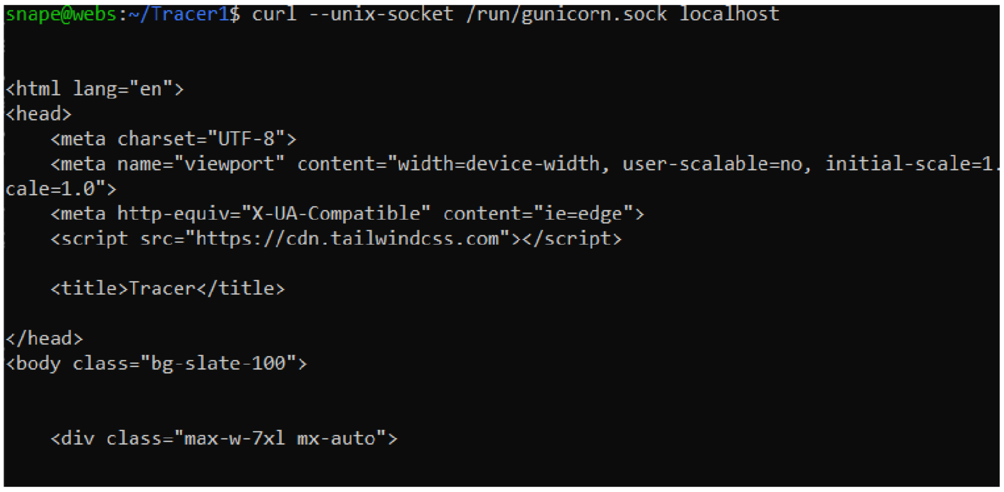
To be sure, you can check if the socket exists using the command below, and you should get the "/run/gunicorn.sock: socket" output. And then, you can test the activation status of Gunicorn. For now, it is inactive because nobody has tried to access the data on a socket, so it has not needed to start the service yet.



```
(venv) snape@django-web:~/Tracer1$ file /run/gunicorn.sock
/run/gunicorn.sock: socket
(venv) snape@django-web:~/Tracer1$ sudo systemctl status gunicorn
o gunicorn.service - gunicorn daemon
     Loaded: loaded (/etc/systemd/system/gunicorn.service; disabled; vendor preset: enabled)
     Active: inactive (dead)
TriggeredBy: ● gunicorn.socket
(venv) snape@django-web:~/Tracer1$
```

Figure 74 – Gunicorn status. Source: author

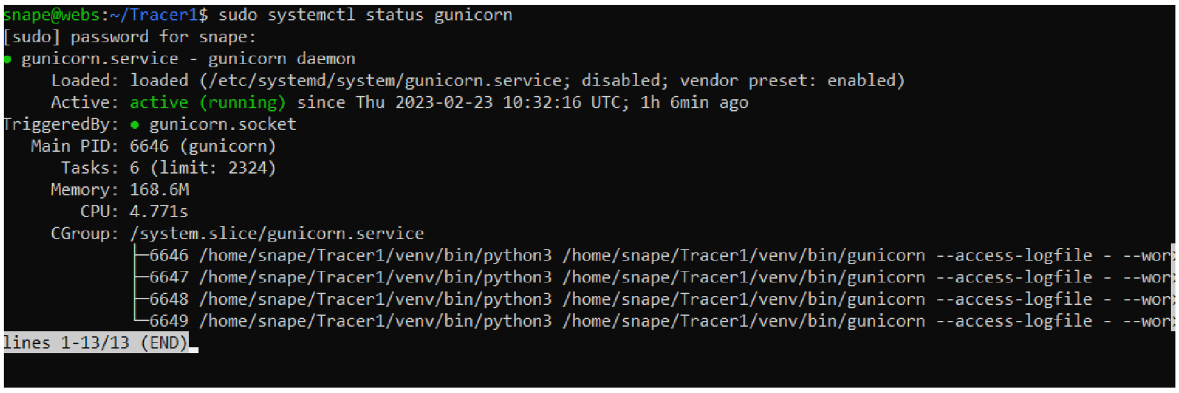
You can quickly start the service using "curl --unix-socket /run/gunicorn.sock localhost" to hit the socket, and as a result, you will get an HTML page.



```
snape@webs:~/Tracer1$ curl --unix-socket /run/gunicorn.sock localhost


<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.
cale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <script src="https://cdn.tailwindcss.com"></script>

    <title>Tracer</title>

</head>
<body class="bg-slate-100">


    <div class="max-w-7xl mx-auto">
```

Figure 75 – Rendered HTML page. Source: author.

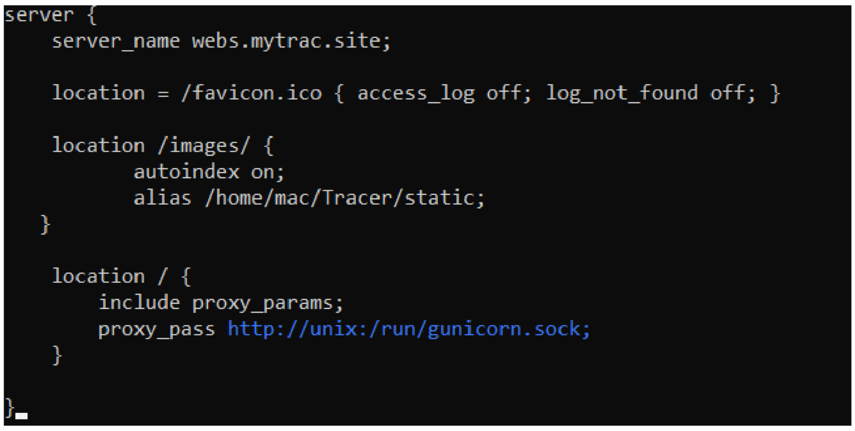
After that, you can verify the status of Gunicorn again, and now it should be running.



```
snape@webs:~/Tracer1$ sudo systemctl status gunicorn
[sudo] password for snape:
● gunicorn.service - gunicorn daemon
     Loaded: loaded (/etc/systemd/system/gunicorn.service; disabled; vendor preset: enabled)
     Active: active (running) since Thu 2023-02-23 10:32:16 UTC; 1h 6min ago
TriggeredBy: ● gunicorn.socket
   Main PID: 6646 (gunicorn)
      Tasks: 6 (limit: 2324)
     Memory: 168.6M
        CPU: 4.771s
     CGroup: /system.slice/gunicorn.service
             ├─6646 /home/snape/Tracer1/venv/bin/python3 /home/snape/Tracer1/venv/bin/gunicorn --access-logfile - --wor
             ├─6647 /home/snape/Tracer1/venv/bin/python3 /home/snape/Tracer1/venv/bin/gunicorn --access-logfile - --wor
             ├─6648 /home/snape/Tracer1/venv/bin/python3 /home/snape/Tracer1/venv/bin/gunicorn --access-logfile - --wor
             └─6649 /home/snape/Tracer1/venv/bin/python3 /home/snape/Tracer1/venv/bin/gunicorn --access-logfile - --wor
lines 1-13/13 (END)
```

Figure 76 – Gunicorn status. Source: author.

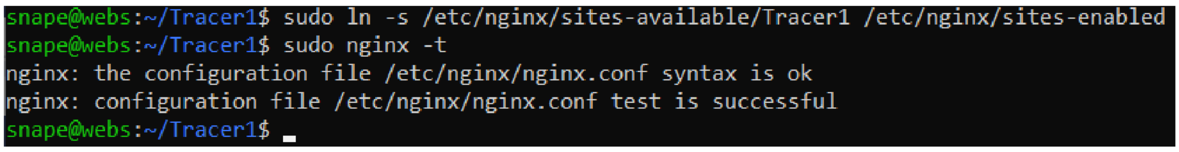
The Gunicorn is up and running, and the next thing you will do is configure Nginx to pass traffic to the process. To make it work, we need to create an Nginx config file inside the "sites-available" directory and write the following code.



```
server {
    server_name webs.mytrac.site;

    location = /favicon.ico { access_log off; log_not_found off; }

    location /images/ {
           autoindex on;
           alias /home/mac/Tracer/static;
   }

    location / {
        include proxy_params;
        proxy_pass http://unix:/run/gunicorn.sock;
    }

}
```

Figure 77 – NGINX configuration. Source: author.

This code tells NGINX to respond to your server's domain name, ignore problems finding a favicon, look for images inside the static folder, and pass the traffic to Gunicorn. Next, you will enable this file by linking it to the sites-enabled directory and checking for errors.



```
snape@webs:~/Tracer1$ sudo ln -s /etc/nginx/sites-available/Tracer1 /etc/nginx/sites-enabled
snape@webs:~/Tracer1$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
snape@webs:~/Tracer1$
```

Figure 78 - Linking it to the sites-enabled directory. Source: author.

If you get no errors, you can restart your NGINX by using the "sudo systemctl restart nginx" command and open your site in a browser.

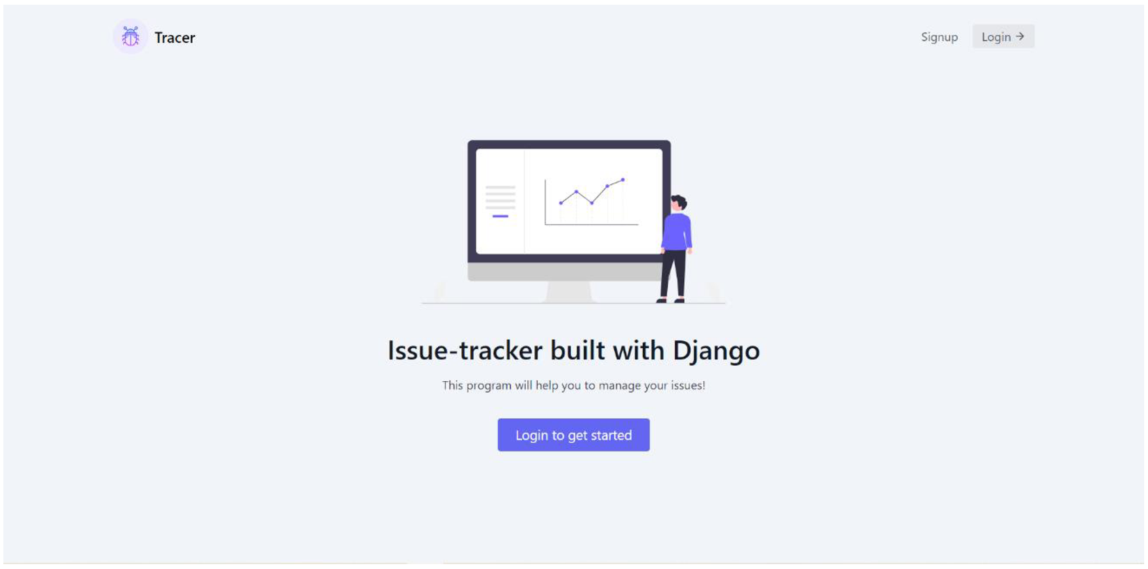
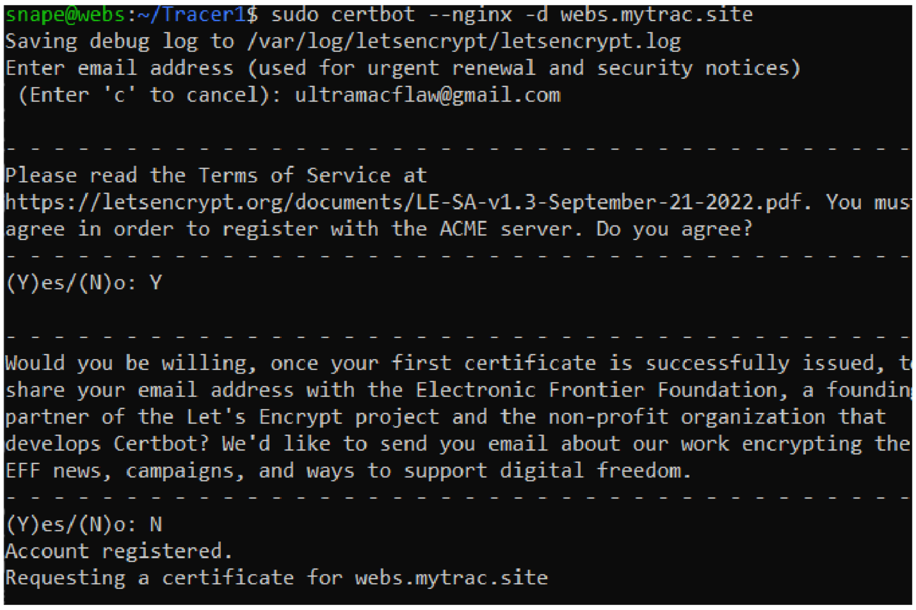

Tracer
Signup
Login →
Issue-tracker built with Django
This program will help you to manage your issues!
Login to get started

Figure 79 – Site page. Source: author.

You can see that your site is now working, but it is not secure to pass your information because it is using HTTP. As our final step, we will change it to HTTPS using "certbot". Go to your terminal and run "sudo apt install certbot python3-certbot-nginx". Then you can execute the following command to get your license.



```
snape@webs:~/Tracer1$ sudo certbot --nginx -d webs.mytrac.site
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Enter email address (used for urgent renewal and security notices)
 (Enter 'c' to cancel): ultramacflaw@gmail.com

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Please read the Terms of Service at
https://letsencrypt.org/documents/LE-SA-v1.3-September-21-2022.pdf. You mus
agree in order to register with the ACME server. Do you agree?
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(Y)es/(N)o: Y

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Would you be willing, once your first certificate is successfully issued, t
share your email address with the Electronic Frontier Foundation, a foundin
partner of the Let's Encrypt project and the non-profit organization that
develops Certbot? We'd like to send you email about our work encrypting the
EFF news, campaigns, and ways to support digital freedom.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(Y)es/(N)o: N
Account registered.
Requesting a certificate for webs.mytrac.site
```

Figure 80 – Configuring certbot. Source: author.

Ultimately, you will get your website up and running on a host server under the HTTPS protocol and functional for your goals.

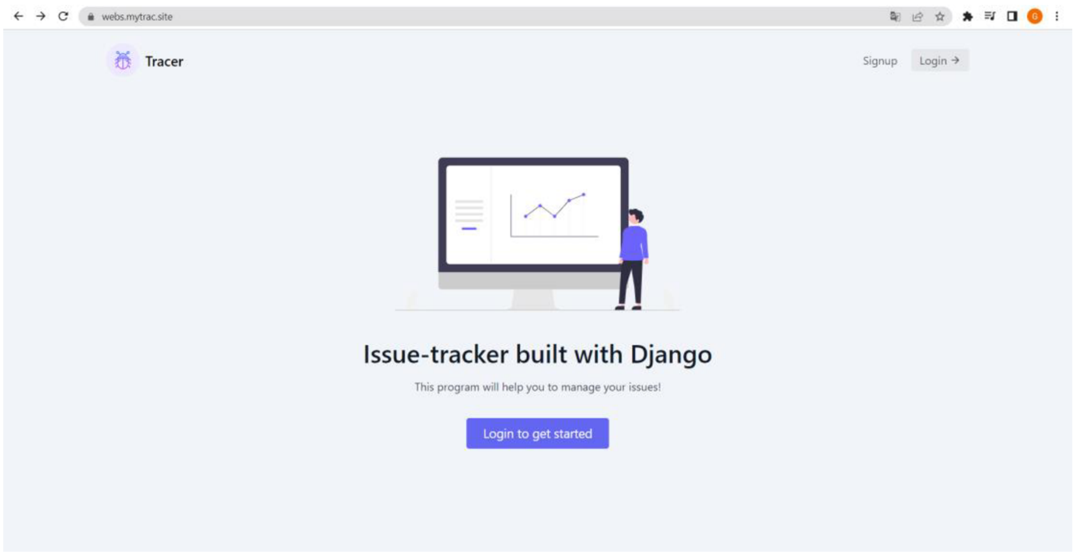

webs.mytrac.site
Tracer
Signup
Login →
Issue-tracker built with Django
This program will help you to manage your issues!
Login to get started

Figure 81 – Running website via HTTPS. Source: author.

5 Results and Discussion

Based on the reviewed literature, all tools for development and thematic concepts were discussed. In the practical part, according to research materials, the project was created and consequently overviewed in this thesis.

The project is a functional and ready-to-use web application with most of the features of bug-tracking systems that people use. You can distinguish main features such as different role functionalities and prioritization of them, notification for changes related to the user, filtering of objects by each letter in the word, customizable categories, the comment section for tickets, the dashboard for ticket categorization count, and the ability to CRUDL all projects, members, and tickets.

The project was tested manually, and from my experience during testing, the project works fine, but there are some minor things that could be improved. These errors have been evaluated and written into the project and are currently being fixed. Nevertheless, it does not touch the project or web server functionality, so you can easily register on a website and apply it to your bug-tracking process.

In the future, I prefer to make a complete bug-tracking system like the big bug-tracking applications on the Internet. It should have more advanced authentication, management of files, different dashboards, be adjusted to the software development frameworks, etc.

6 Conclusion

As we mentioned earlier, nowadays, we have to pave our path through the bugs in the IT sphere to move technological progress forward. Bug-tracking systems can help us with it by providing software to manage all information about issues in our project. Another essential part of progress is making things public and open on the Internet, so there is no need to create your own system if you have an alternative. This process is called web development, which is one of the most valuable things in the human world because it makes space for communication and the spread and sharing of information with other people across the globe. And this thesis combines the two to examine those areas and provide a brief overview of each.

The thesis's primary and secondary objectives have been attained. Sources of expert knowledge have been investigated and assessed. Based on a synthesis of the acquired information, tools and technologies for the web development process were selected. As a result of that, the bug-tracking application was designed and implemented. The practical part of the thesis aimed to explain the content of a project and get acquainted with the process of its creation. It went through an explanation of how to create a starter project, project folder content, what the app is and consists of, what other folders and files are responsible for, how to install and configure packages, how to get the project ready to deploy, and eventually, how to deploy it.

The project was deployed and made open for public use using the mentioned tools and platforms. The created web application is currently available on the Internet and functional for issue-tracking purposes.

7 References

1. *A short history of the web*. (n.d.). Retrieved from CERN: https://home.cern/science/computing/birth-web/short-history-web
2. Alex Paul. (2023, February 26). *What is a web database?* Retrieved from wiseGEEK: clear answers for common questions: https://www.wise-geek.com/what-is-a-web-database.htm
3. Anquetil, R. (2019). *Fundamental Concepts For Web Development: HTML5, CSS3, JavaScript and much more!* Webstreet Learning.
4. Codecademy. (n.d.). *What is an IDE?* Retrieved from Codecademy: https://www.codecademy.com/article/what-is-an-ide
5. *Difference between MVC and MVT Design Patterns*. (2022, September 14). Retrieved February 9, 2023, from GeeksforGeeks: https://www.geeksforgeeks.org/difference-between-mvc-and-mvt-design-patterns/
6. *Django overview*. (n.d.). Retrieved from Django Project: https://www.djangoproject.com/start/overview/
7. G., D. (27, February 2023). *What is web hosting – web hosting explained for Beginners*. Retrieved from Hostinger Tutorials: https://www.hostinger.com/tutorials/what-is-web-hosting/
8. *General Python FAQ*. (n.d.). Retrieved from Python documentation: https://docs.python.org/3/faq/general.html#why-was-python-created-in-the-first-place
9. Joy, A. (2019, October 29). *Difference between MVC and MVT Patterns*. Retrieved from Pythonista Planet: https://pythonistaplanet.com/difference-between-mvc-and-mvt/
10. Marionletendart. (2021, May 10). *What is web development? definition from OpenClassrooms*. Retrieved from The OpenClassrooms Blog: https://blog.openclassrooms.com/en/2018/03/28/web-development-definition/
11. Martelli, A. A. (2005). *Python cookbook.* O'Reilly Media, Inc.
12. Melé, A. (2020). *Django 3 By Example: Build powerful and reliable Python web applications from scratch.* Packt Publishing Ltd.
13. *MVC vs MVT Architectural Pattern*. (2021, September 28). Retrieved from Medium: https://medium.com/dsc-umit/mvc-vs-mvt-architectural-pattern-d306a56dce55

14. Nagappan, M. (2022, November 14). *What is Bug Tracking System?: Kissflow workflow - issue tracking*. Retrieved from Kissflow: https://kissflow.com/issue-tracking/what-is-bug-tracking-system/
15. Nishant, V. (2020, August 20). *Django MVT Architecture*. Retrieved from AskPython: https://www.askpython.com/django/django-mvt-architecture
16. Prokopets, M. (2020, March 21). *The beginner's guide to chrome developer tools*. Retrieved from Nira: https://nira.com/chrome-developer-tools/
17. *PYPL popularity of Programming Language index*. (n.d.). Retrieved from index: https://pypl.github.io/PYPL.html
18. *Python Virtual Environment: Introduction*. (2022, December 15). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/python-virtual-environment/
19. *Software release of WWW into public domain*. (2009, March 02). Retrieved from CERN Document Server: https://cds.cern.ch/record/1164399
20. Solutions, M. (2017, October 03). *Python: 7 important reasons why you should use python*. Retrieved from Medium: https://medium.com/@mindfiresolutions.usa/python-7-important-reasons-why-you-should-use-python-5801a98a0d0b
21. *The python logo*. (n.d.). Retrieved from Python.org: https://www.python.org/community/logos/
22. *What is bug tracking?* (n.d.). Retrieved from IBM: https://www.ibm.com/topics/bug-tracking
23. *What is software framework? - definition from Techopedia*. (n.d.). Retrieved from Techopedia.com: https://www.techopedia.com/definition/14384/software-framework
24. *What is web development? - definition from Techopedia*. (n.d.). Retrieved from Techopedia.com: https://www.techopedia.com/definition/23889/web-development
25. Williams, A. (2012, July 09). *GitHub pours energies into enterprise – raises $100 million from power VC Andreessen horowitz*. Retrieved from TechCrunch: https://techcrunch.com/2012/07/09/github-pours-energies-into-enterprise-raises-100-million-from-power-vc-andreesen-horowitz/