Messaging System For Inter Organization Communication

**Team Members:**

1. Yadnyesh Darekar

2. VijayKumar Katta

3. Pratiksha Rathod

**Mentors**:

1. Milan Sharma

2. Sumit Mishra

3.Tanuj Kumar

Introduction

This POC documentation will introduce how to build part of a messaging system using the microservices, python, NoSQL database, Kafka cluster, ELK and SMTP server. Kafka is a distributed messaging system that we developed for collecting and delivering high volumes of message log data with low latency.

In this documentation we will demonstrate details of all software tools which are used to build this whole system. Also the technique and process which is showed here can be applied to build the other major components in this messaging system.

Working Phases:

This POC is divided into three working modules:

* Onboarding of organizations and users
* Once a user is onboarded, send a welcome email to the user
* Provide mailbox management capability to the users

**Software Tools Installation**

**FastAPI:**

FastAPI is a modern, fast(high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.

Key Features:

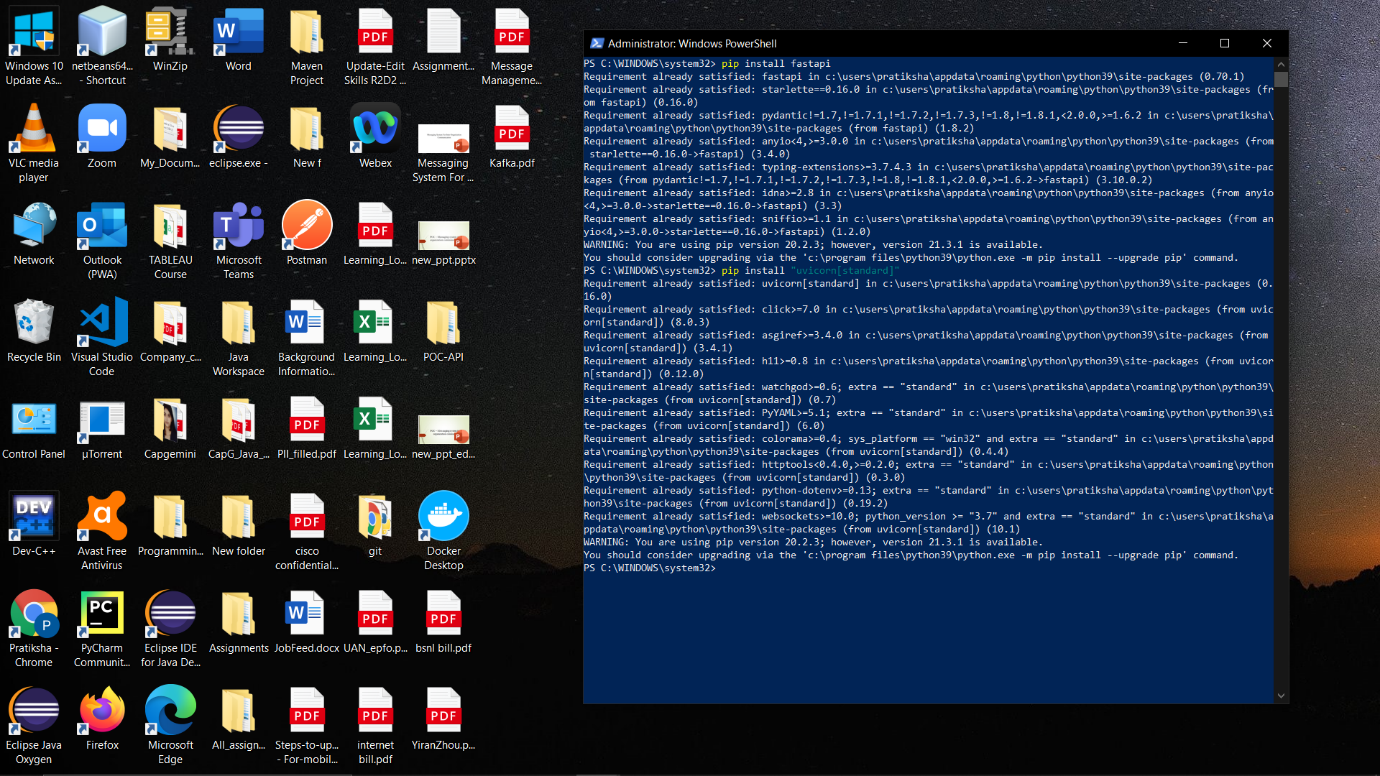
- Fast to code

-Fewer bugs

- Easy to use and learn

- Minimize code duplication

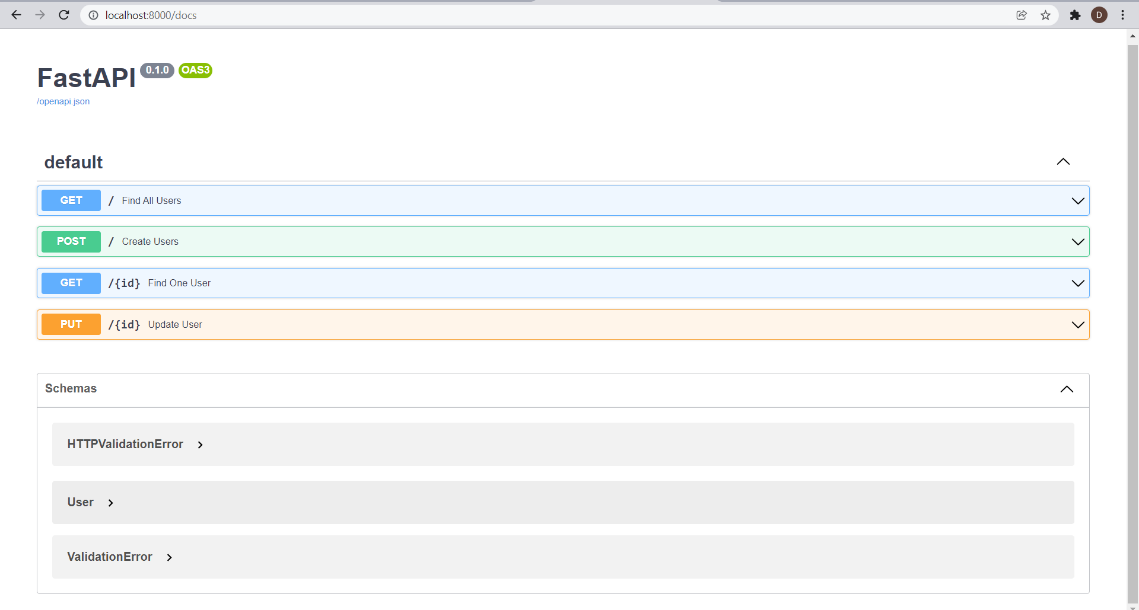
**Installation**: We have installed fast api by executing following command:



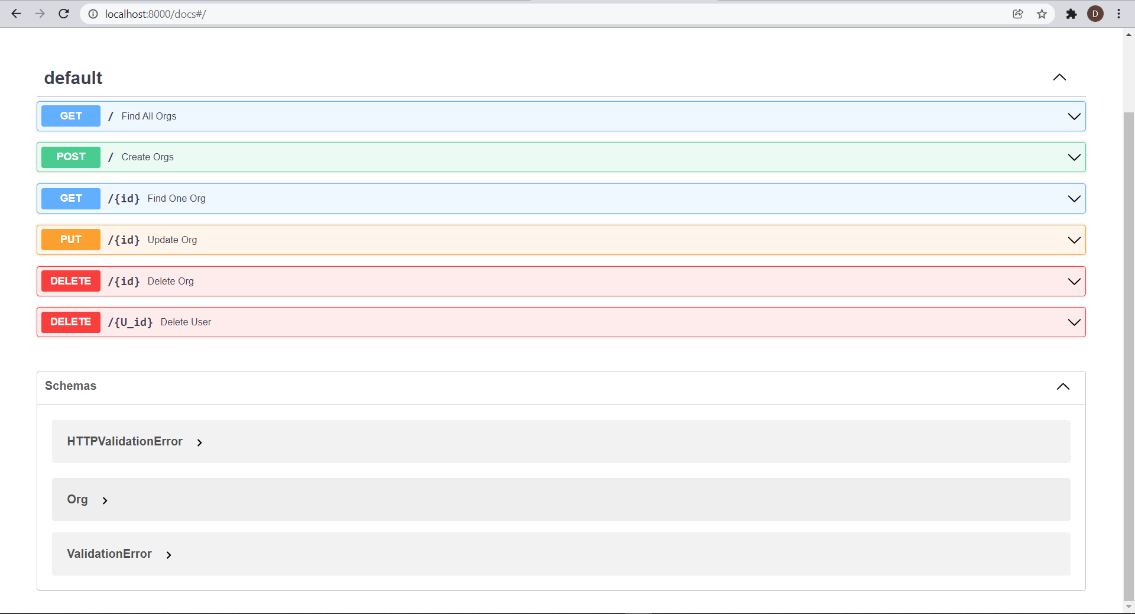
**Swagger:**

Swagger is an Interface Description Language for describing RESTful APIs expressed using JSON. The purpose of this framework is to help python developers create a self-documenting JSON API for mongodb database objects and relationships.

Swagger UI for User Onboarding: For user onboarding we have performed crud operation and is shown in swagger framework.

****

Swagger UI for organization onboarding: For organization onboarding we have performed crud operation and is shown in swagger framework.



**Database:**

MONGODB is a document-oriented and NOSQL database solution that

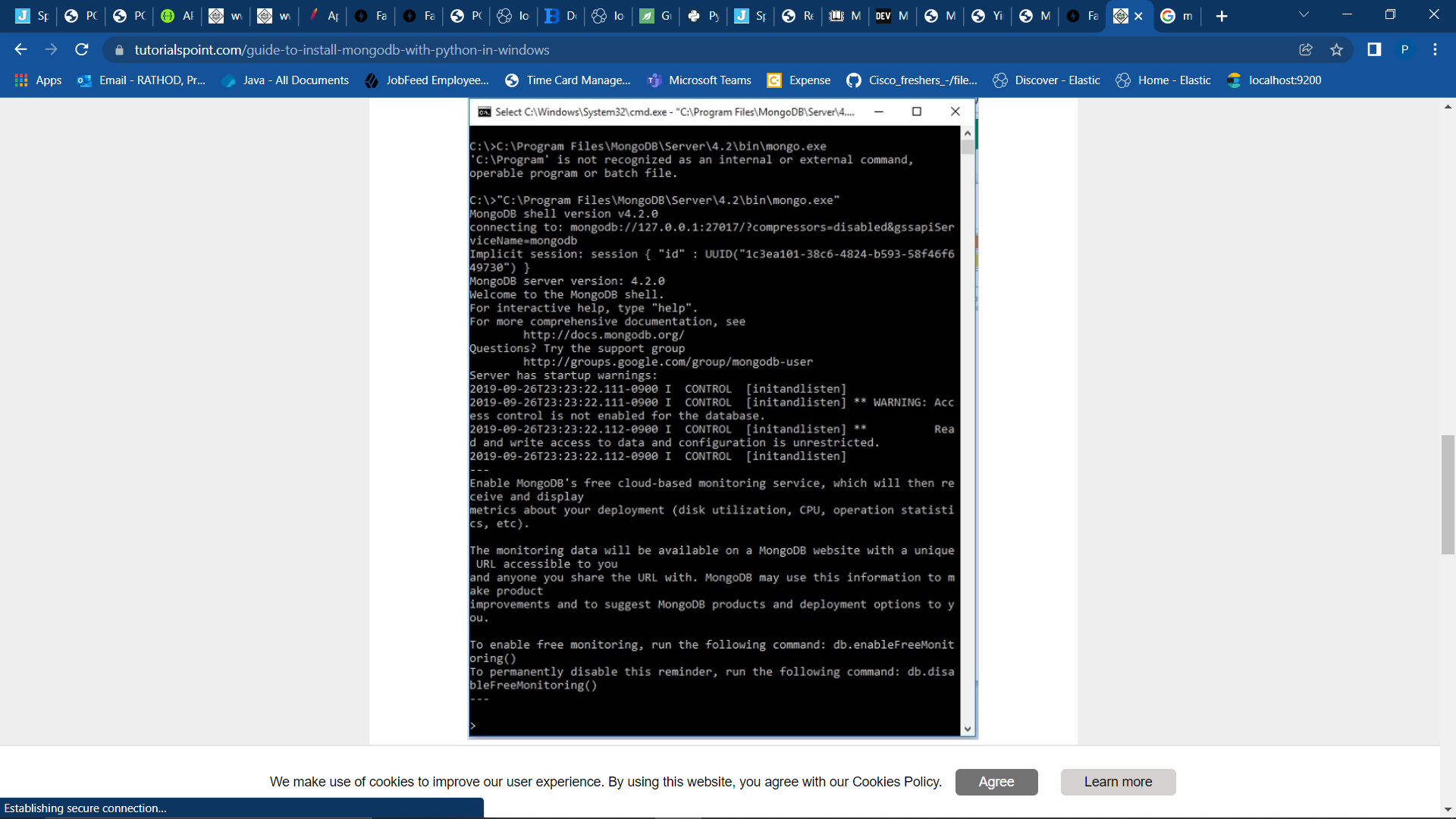
provides great scalability and flexibility along with a powerful

querying system. With MONGODB and Python, we can develop

many different types of database applications quickly.

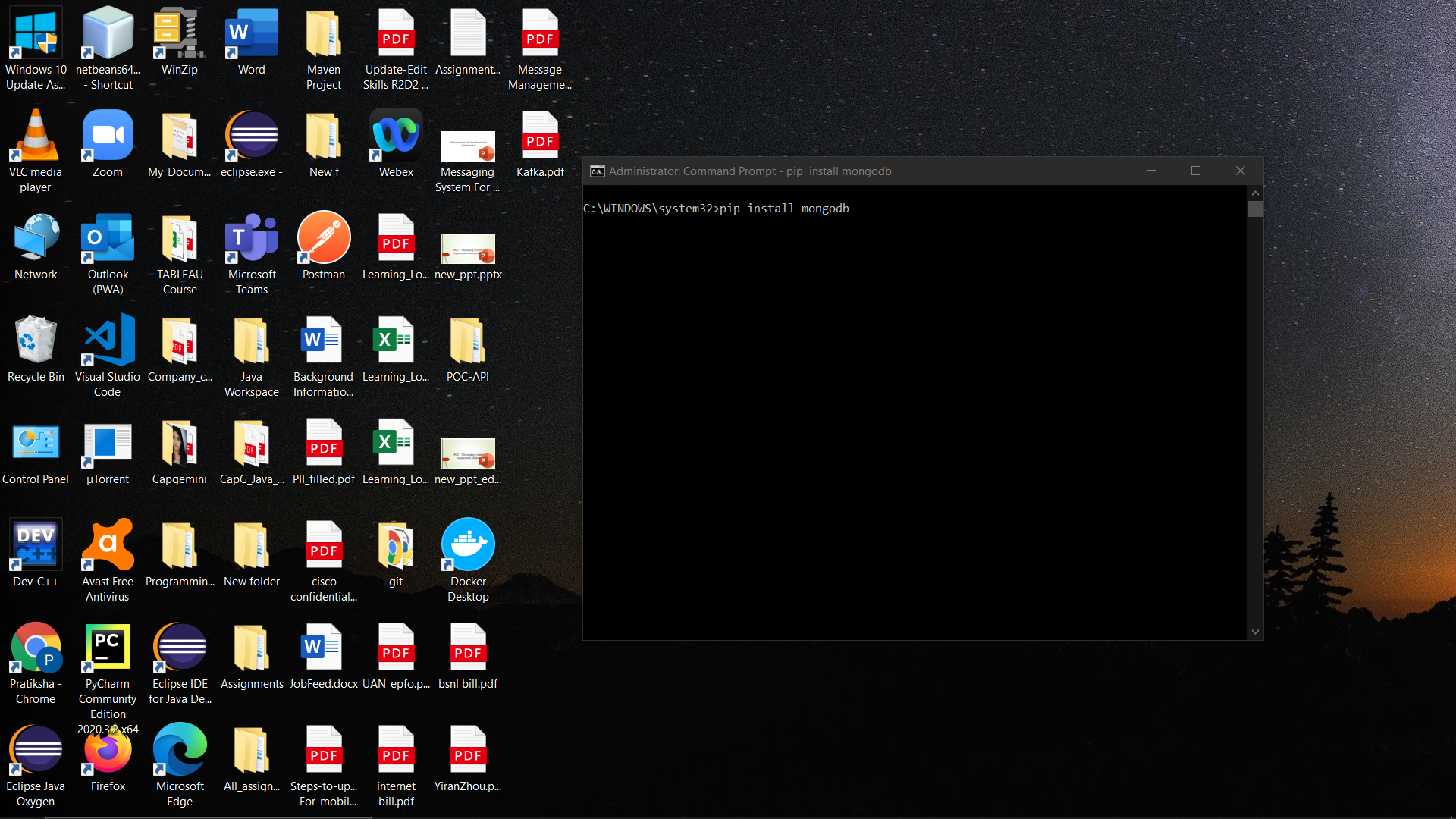
Installation steps:

1. Install the community edition of MONGODB compass.
2. Run installed MONGODB windows installer package that we have downloaded.
3. After installation, set MONGODB environment by creating data directory to store data.
4. Start MONGODB by running “mongod.exe” command in command prompt.



Installing Python Driver:

We install the python driver so that python can interact with MONGODB. For this, we go to the python environment already installed in windows and add the package PYMONGO. The command to do this is shown below.



**Database structure**:

1. Organization Database:

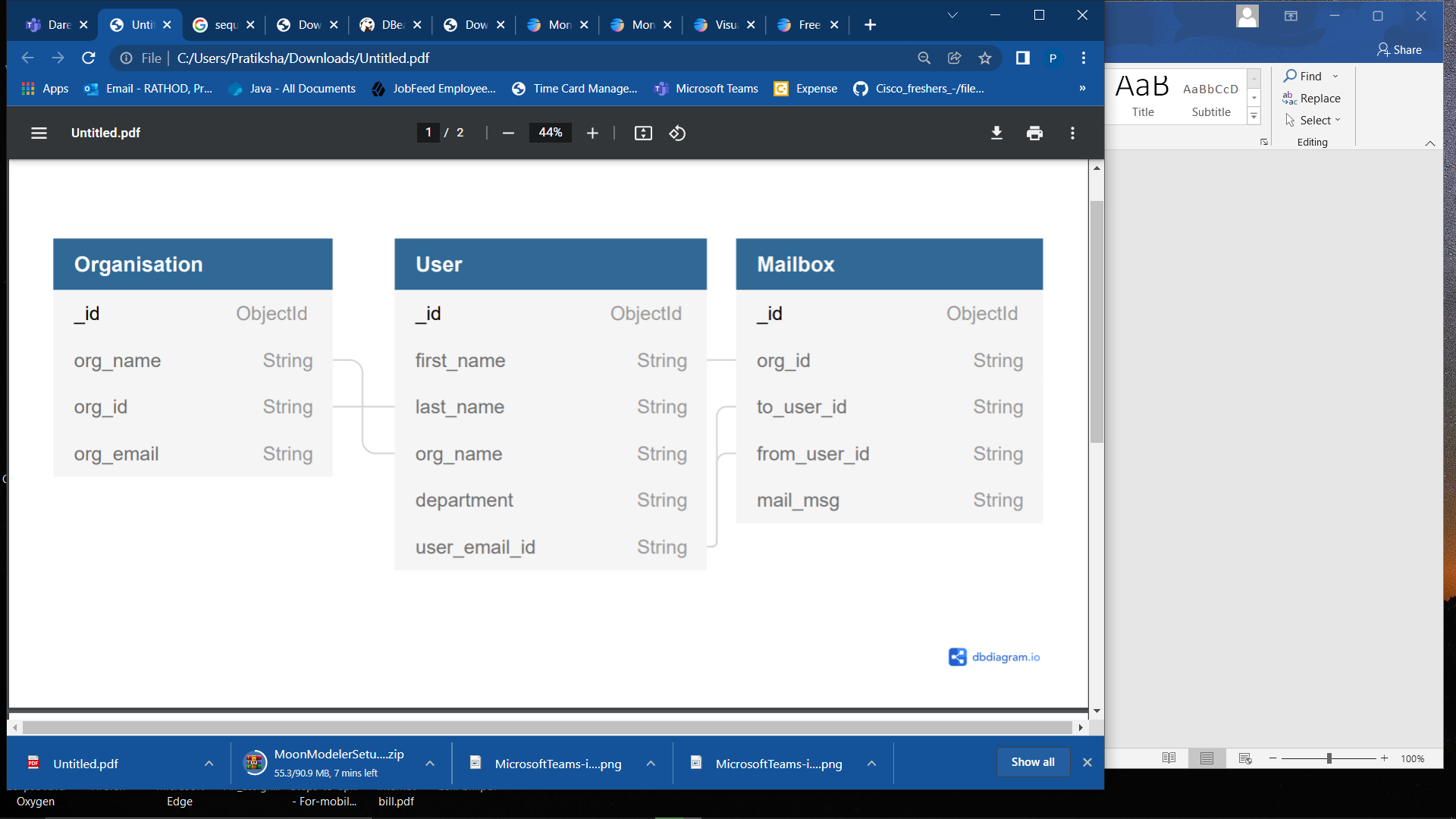


2.User Database: 

3.Mailbox Management:



**Database Design:** In this database, we have entities for Organization, user and mailbox. User entity have attributes like firstname, lastname, org\_name, department and user\_email\_id. In this design, user assigned an organization according to this organization, user get mail from mailbox entity.



Entity Relationship Diagram

**Docker**:

Docker is an open source containerization platform. It enables developers to package applications into containers-standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.

Installation steps:

1. Double-click docker Desktop Installer.exe to run the installer as

we have downloaded.

1. When prompted, ensure the Enable Hyper-V Windows Features

or the Install required Windows components for WSL 2 option is selected on the Configuration page.

1. When the installation is successful, click Close to complete the installation process.

**Kafka**:

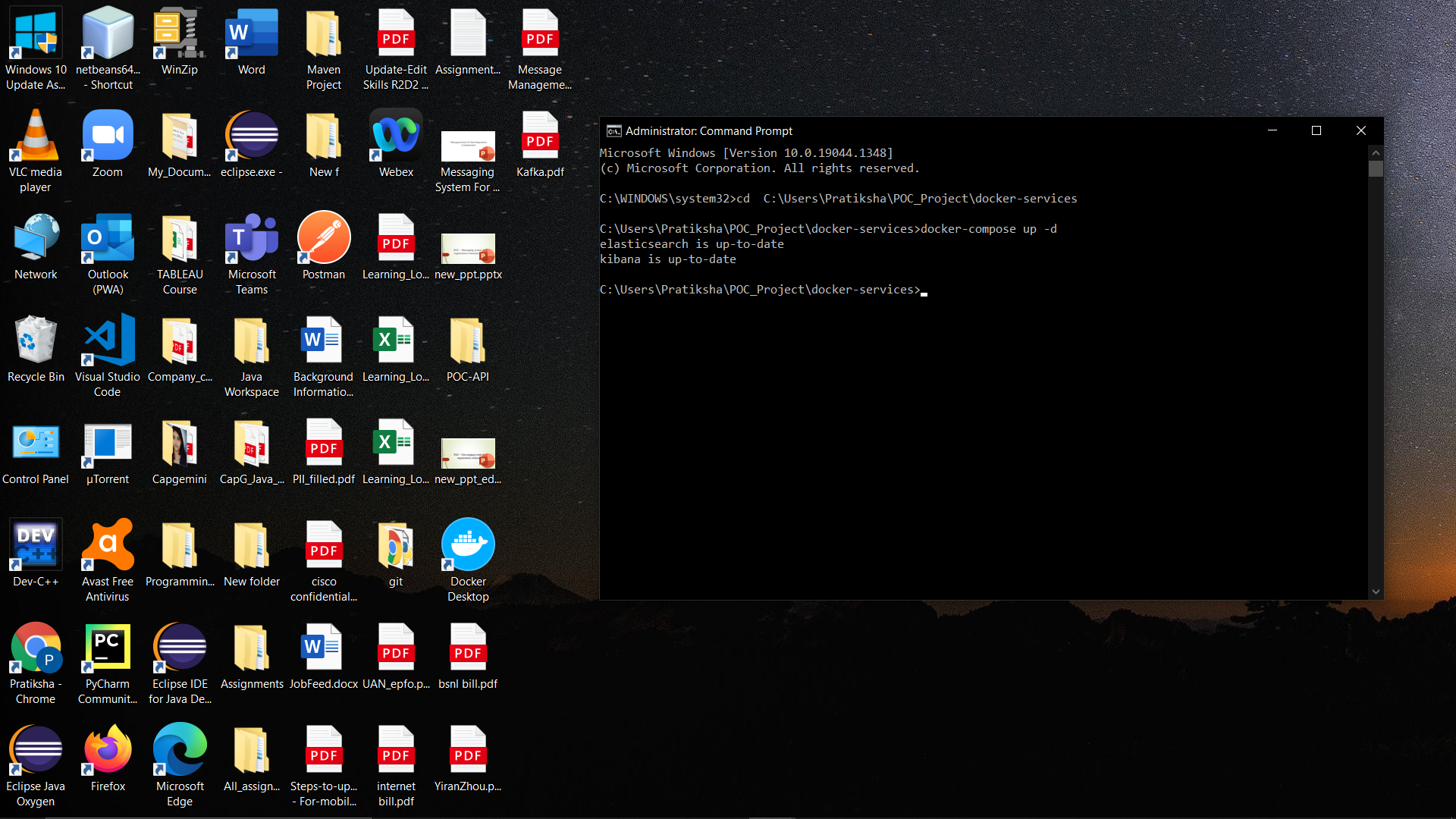
Kafka is primarily used to build real-time streaming data pipelines and applications that adapt to the data streams. It combines messaging, storage, and stream processing to allow storage and analysis of both historical and real-time data.

Installation:

To start an Apache Kafka server, we'd first need to start a Zookeeper server.

We can configure this dependency in a docker-compose.yml file, which will

ensure that the Zookeeper server always starts before the Kafka server and stops after it. In this setup, our Zookeeper server is listening on port=2181 for the KAFKA service, which is defined within the same container setup. However, for any client running on the host, it'll be exposed on port 22181.Let's start the Kafka server by spinning up the containers using the docker-compose command:

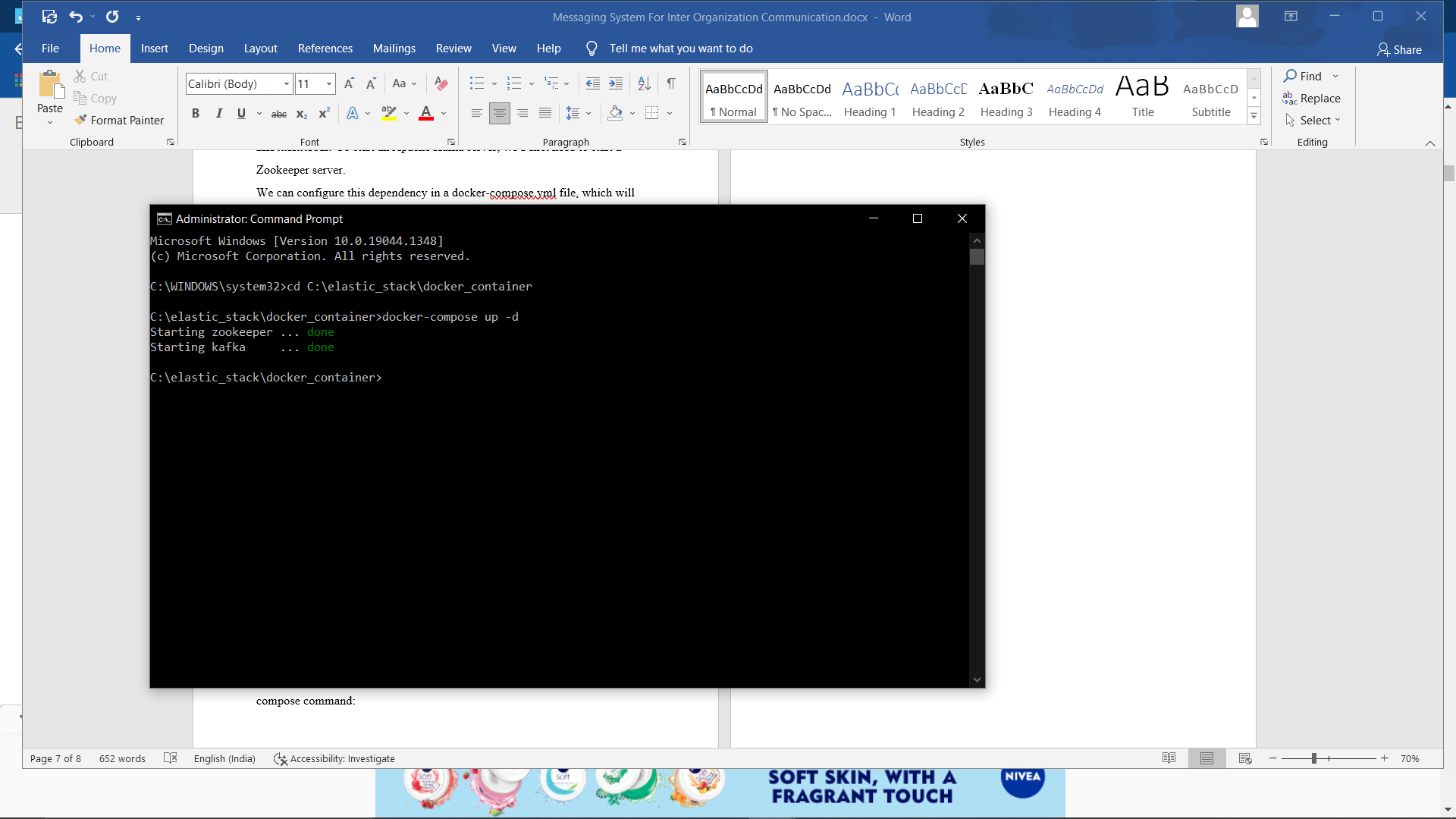


**ELK:**

ELK stack gives you the ability to aggregate logs from all your systems and applications, analyze these logs, and create visualizations for application and infrastructure monitoring, faster troubleshooting, security analytics.

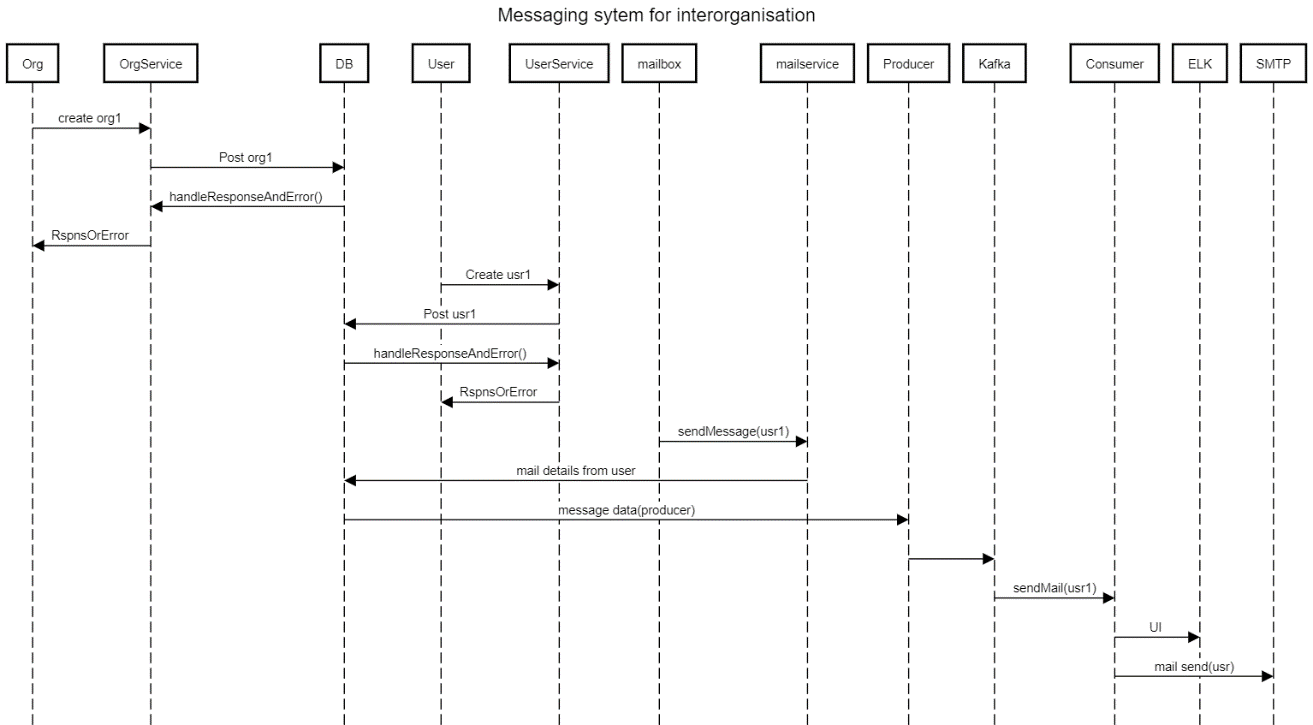
**Installatin steps:**

kibana containers so we can see how things work and visualize.Let's start the ELK stack by spinning up the containers using the docker-compose command:



**Sequence Diagram**:

It shows the interaction between user, organization, mailbox and software tools like kafka, mongodb, ELK and SMTP.



**Unit Testing:**

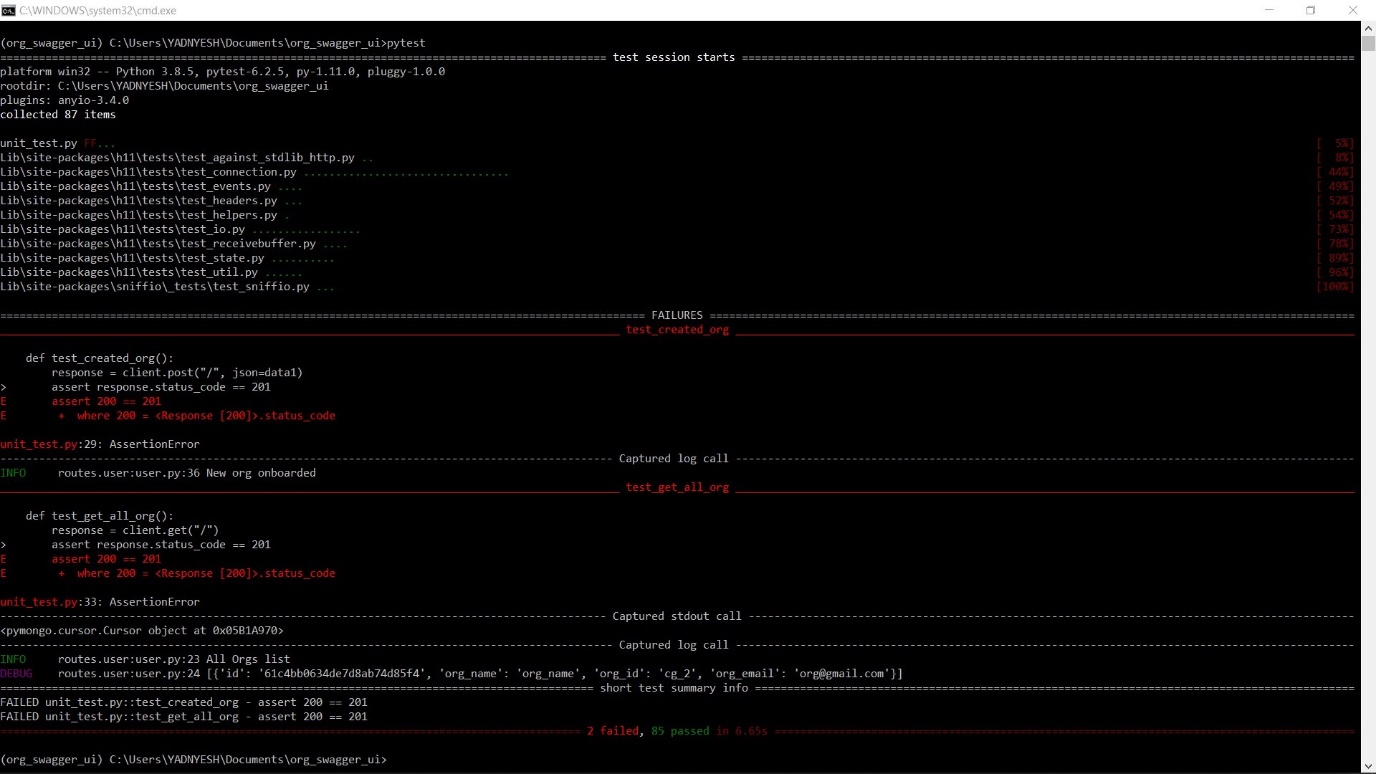
Unit Testing is the first level of software testing where the smallest testable parts of a software are tested. This is used to validate that each unit of the software performs as designed.  
The unittest test framework is python’s xUNIT style framework. To achieve this, [unittest](https://docs.python.org/3/library/unittest.html#module-unittest) supports test fixture, test case, test suite, test runner.

**PyTest** is a testing framework that allows users to write test codes using Python programming language. It helps you to write simple and scalable test cases for databases, APIs, or UI. PyTest is mainly used for writing tests for APIs. It helps to write tests from simple unit tests to complex functional tests.

* Installation of pytest.: Run “pip install pytest” on terminal
* Write a test file
* For execute test file run “pytest” on terminal

In this project there are four test cases. coverage of unit test case is for the https status code 200

Result when all test cases are pass:

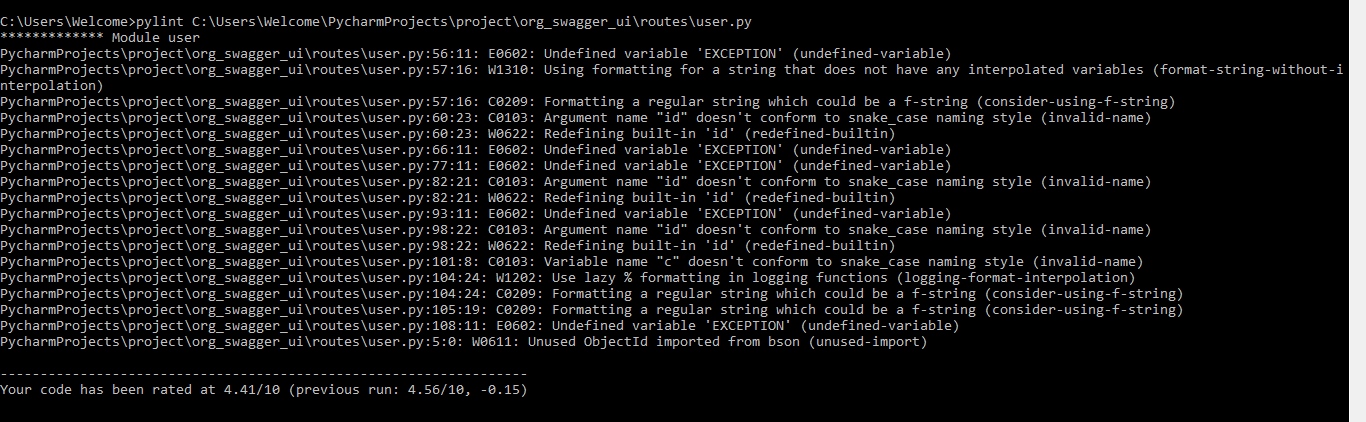
Result with failed test cases:

**Static code analysis:**

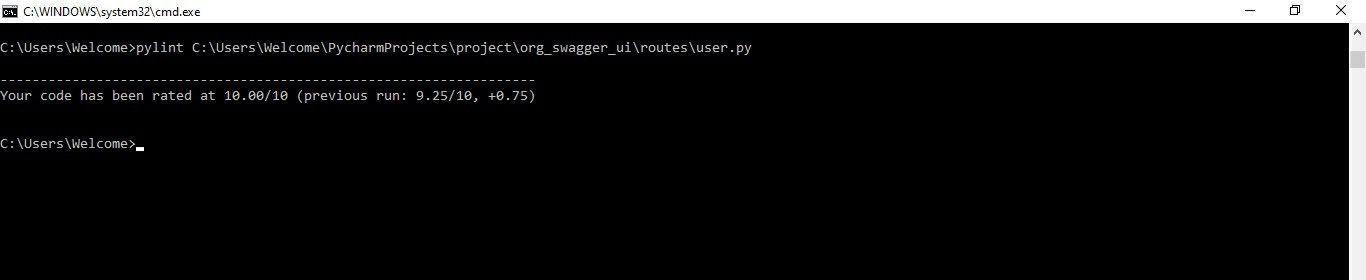
Static code analysis looks at the code without executing it. It is usually extremely fast to execute, requires little effort to add to your workflow, and can uncover common mistakes.

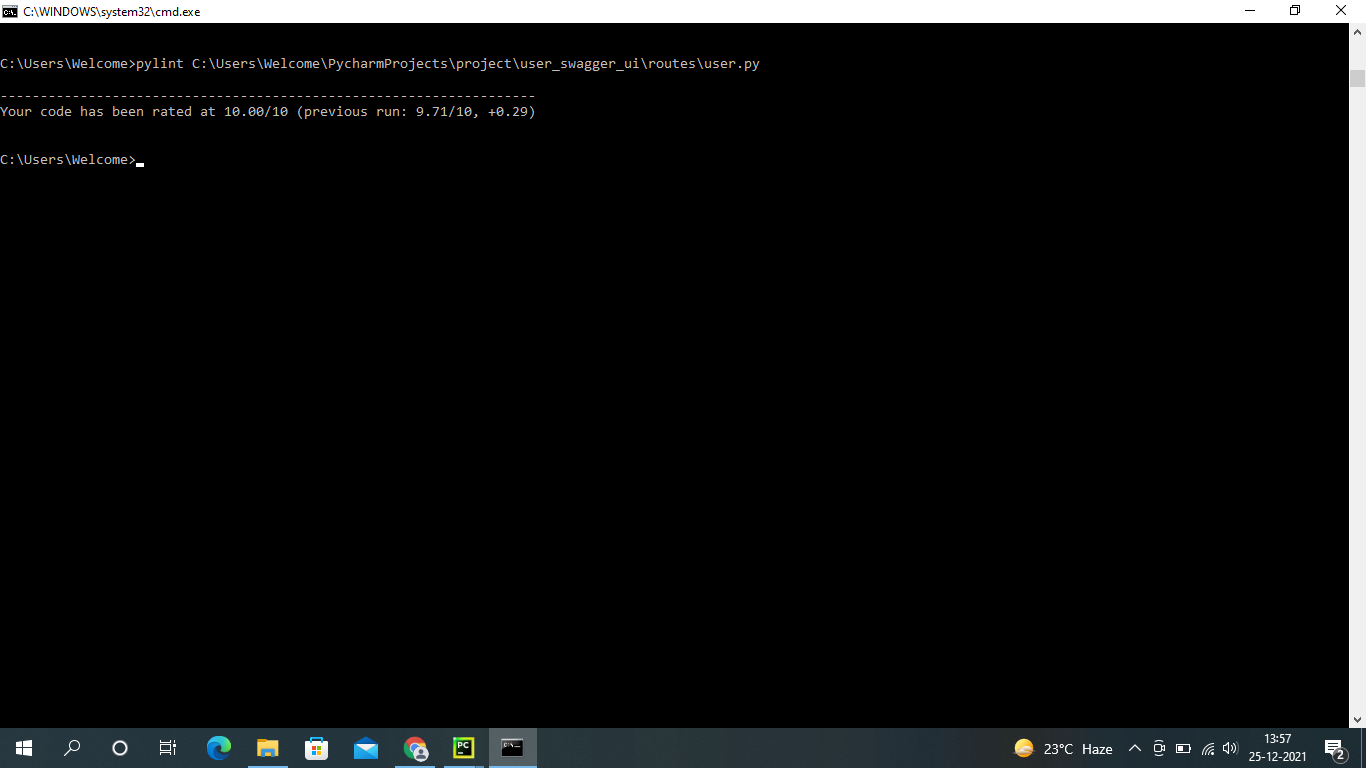
For static analysis we have using “pylint” tool. it checks for errors in python code, tries to enforce a coding standard and looks for code smells. It can also looks for certain type errors. It makes code’s complexity.

Organisation on board -Before code analysis:



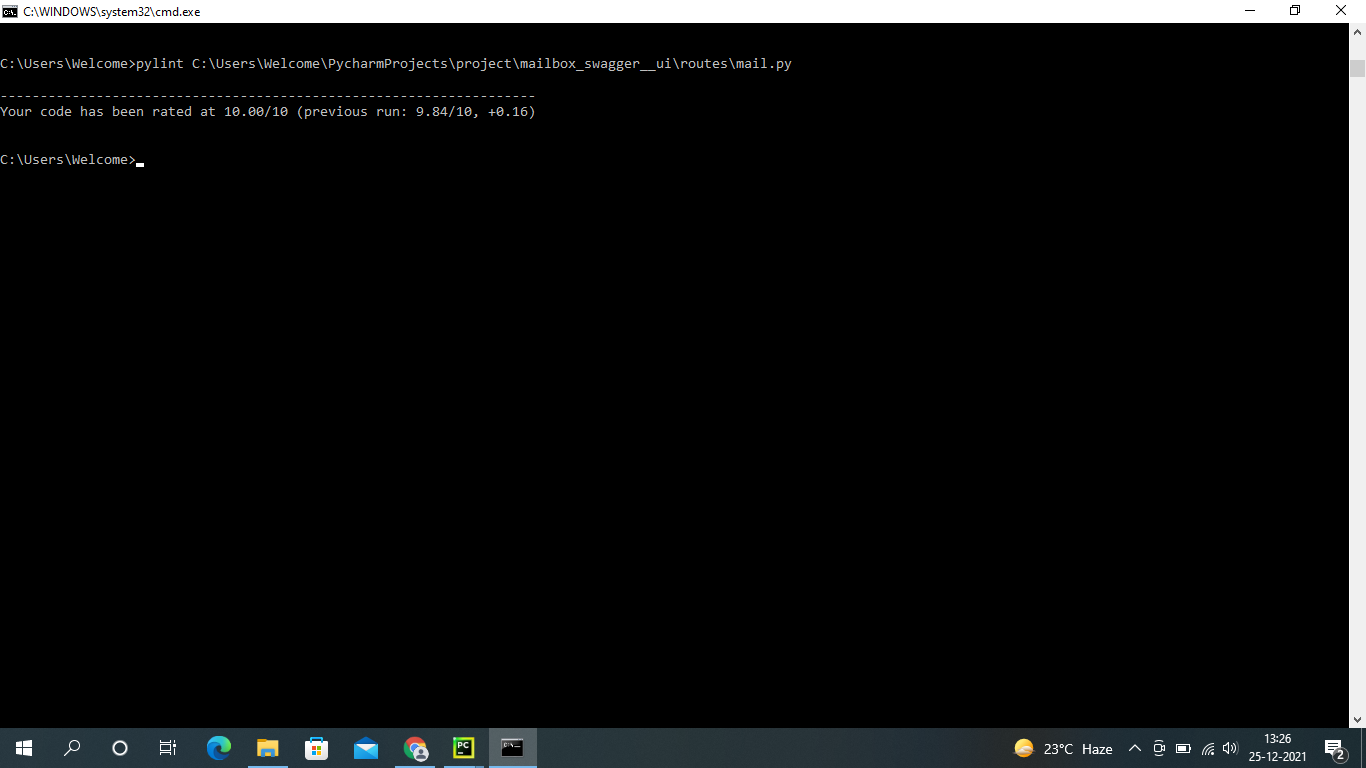
Organisation on board -After code analysis:



User on board – after code analysis:

Mailbox – before code analysis:



Mailbox – after code analysis success:

Kafka-code analysis:

