## What is Django ?

Django is an open-source python web framework used for rapid development, pragmatic, maintainable, clean design, and secures websites. ... It takes care of a lot of hassle involved in the web development; enables users to focus on developing components needed for their application.

## 

## 

## Why do you need a framework?

To understand what Django is actually for, we need to take a closer look at the servers. The first thing is that the server needs to know that you want it to serve you a web page.

Imagine a mailbox (port) which is monitored for incoming letters (requests). This is done by a web server. The web server reads the letter and then sends a response with a webpage. But when you want to send something, you need to have some content. And Django is something that helps you create the content.

# APPS USED

# 

# 

# What is the use of blog?

They can be used for anything that involves communicating or publishing information on the World Wide Web. Common **uses** include teaching and educational and corporate **use**. . Your **blog** can be a personal diary, a project collaboration tool, a guide, or any means of communicating and publishing information on the web.

# Advantage of Django blog

The biggest advantage of using Django to make a blog or web app is pretty clear, it’s speed. However, that speed is two-fold when dealing with Django. First, it is in development, as you don’t need to reinvent the wheel, you will find that you just breeze through many of the cumbersome and time-consuming parts of development. Your new site or app will be able to soar farther and higher than ever before because it was built on the backs of giants. Most, if not all, of the work you’d be doing, will have already been done AND tested for you when you choose to work within a framework. Second, speed in performance.

Django is based in Python, which has time and time again outperformed both Ruby and PHP. In today’s day of mobile websites and apps, it is important that you are relaying data as quickly as possible with making as few and lightweight server calls as possible. You want to be sure that your clients don’t waste both time and precious data off their plans, just trying to load your website.

# FILES

# Managing files

This document describes Django’s file access APIs for files such as those uploaded by a user. The lower level APIs are general enough that you could use them for other purposes. If you want to handle “static files” (JS, CSS, etc.), see [Managing static files (e.g. images, JavaScript, CSS)](https://docs.djangoproject.com/en/3.0/howto/static-files/).

By default, Django stores files locally, using the [**MEDIA\_ROOT**](https://docs.djangoproject.com/en/3.0/ref/settings/#std:setting-MEDIA_ROOT) and [**MEDIA\_URL**](https://docs.djangoproject.com/en/3.0/ref/settings/#std:setting-MEDIA_URL) settings. The examples below assume that you’re using these defaults.

However, Django provides ways to write custom [file storage systems](https://docs.djangoproject.com/en/3.0/topics/files/#file-storage) that allow you to completely customize where and how Django stores files. The second half of this document describes how these storage systems work.

# USERS

# User authentication in Django

Django comes with a user authentication system. It handles user accounts, groups, permissions and cookie-based user sessions. This section of the documentation explains how the default implementation works out of the box, as well as how to [extend and customize](https://docs.djangoproject.com/en/3.0/topics/auth/customizing/) it to suit your project’s needs.

# Overview

The Django authentication system handles both authentication and authorization. Briefly, authentication verifies a user is who they claim to be, and authorization determines what an authenticated user is allowed to do. Here the term authentication is used to refer to both tasks.

The auth system consists of:

* Users
* Permissions: Binary (yes/no) flags designating whether a user may perform a certain task.
* Groups: A generic way of applying labels and permissions to more than one user.
* A configurable password hashing system
* Forms and view tools for logging in users, or restricting content
* A pluggable backend system

The authentication system in Django aims to be very generic and doesn’t provide some features commonly found in web authentication systems. Solutions for some of these common problems have been implemented in third-party packages

* Password strength checking
* Throttling of login attempts
* Authentication against third-parties (OAuth, for example)
* Object-level permissions

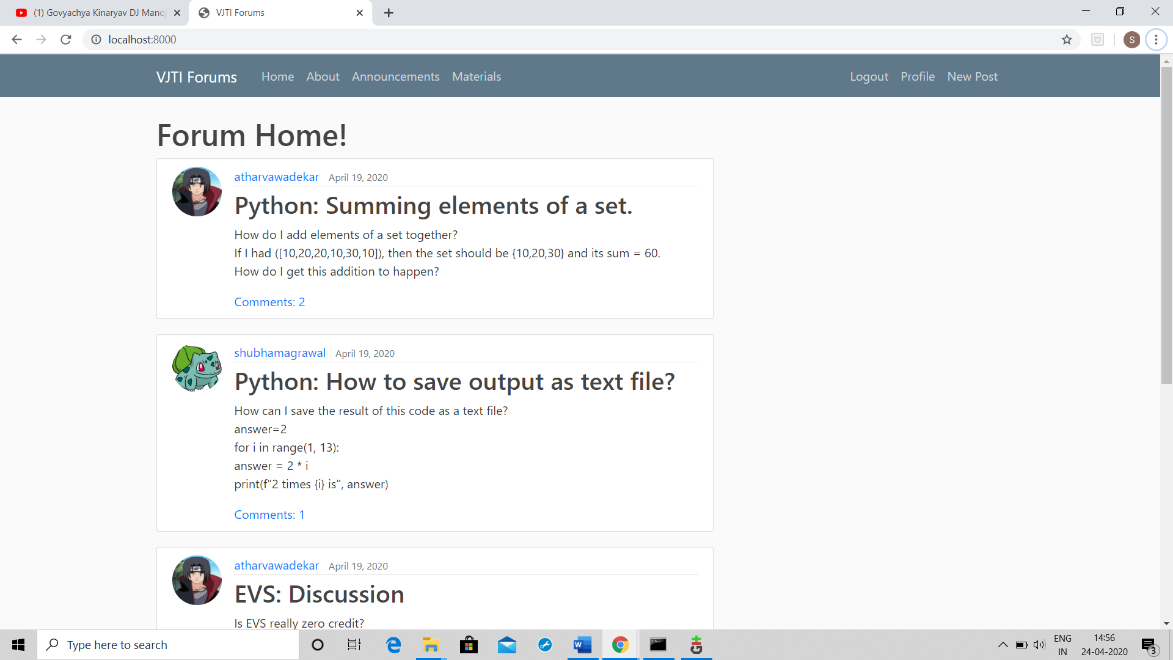
# Aim of the project

* We can see post .
* We can create post .
* We can register.
* We can update our profile picture.

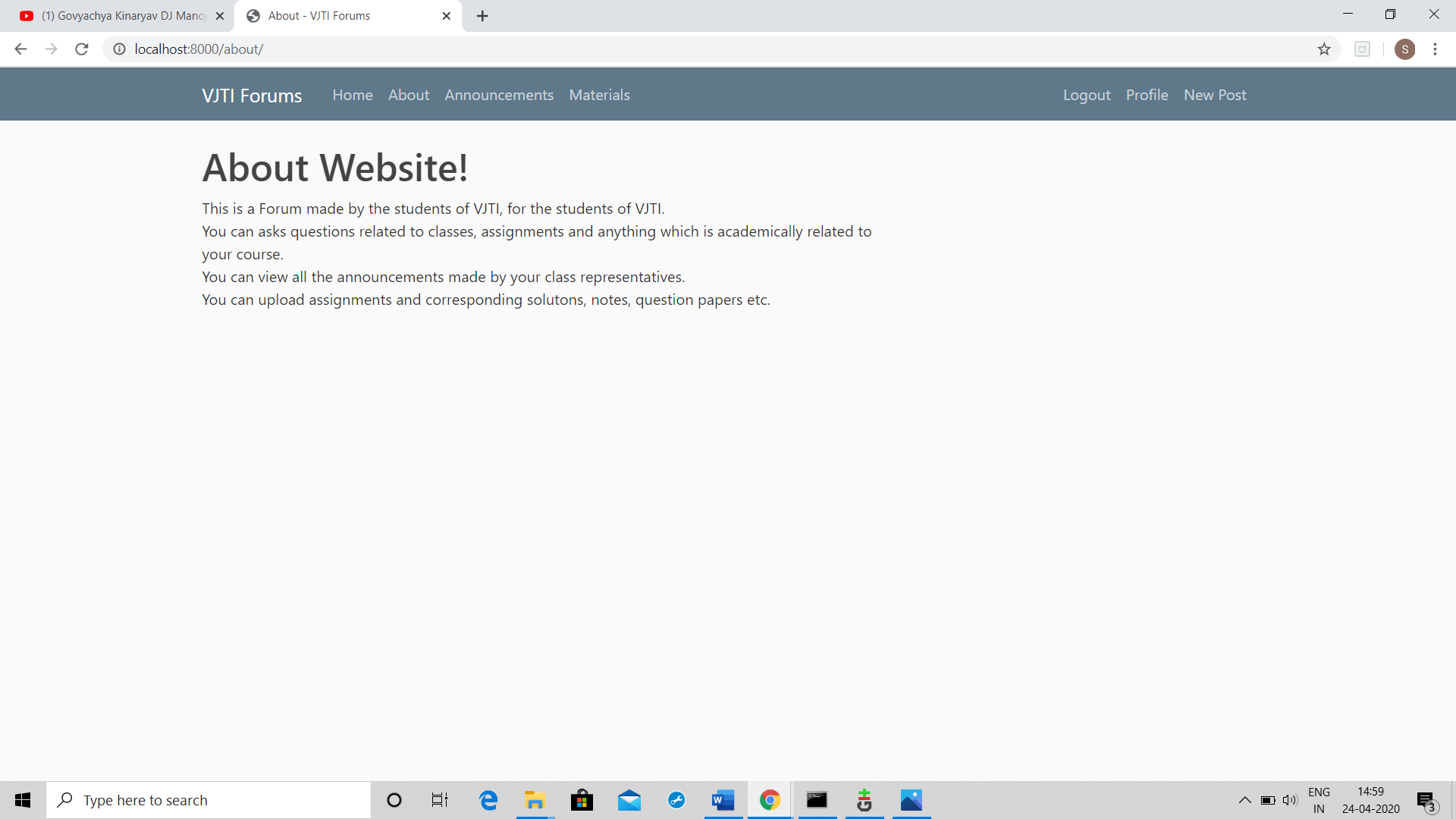
* We can also upload any files and also we can download files.
* We can view notifications from cr.
* We can also comment on post.

# Website snapshots

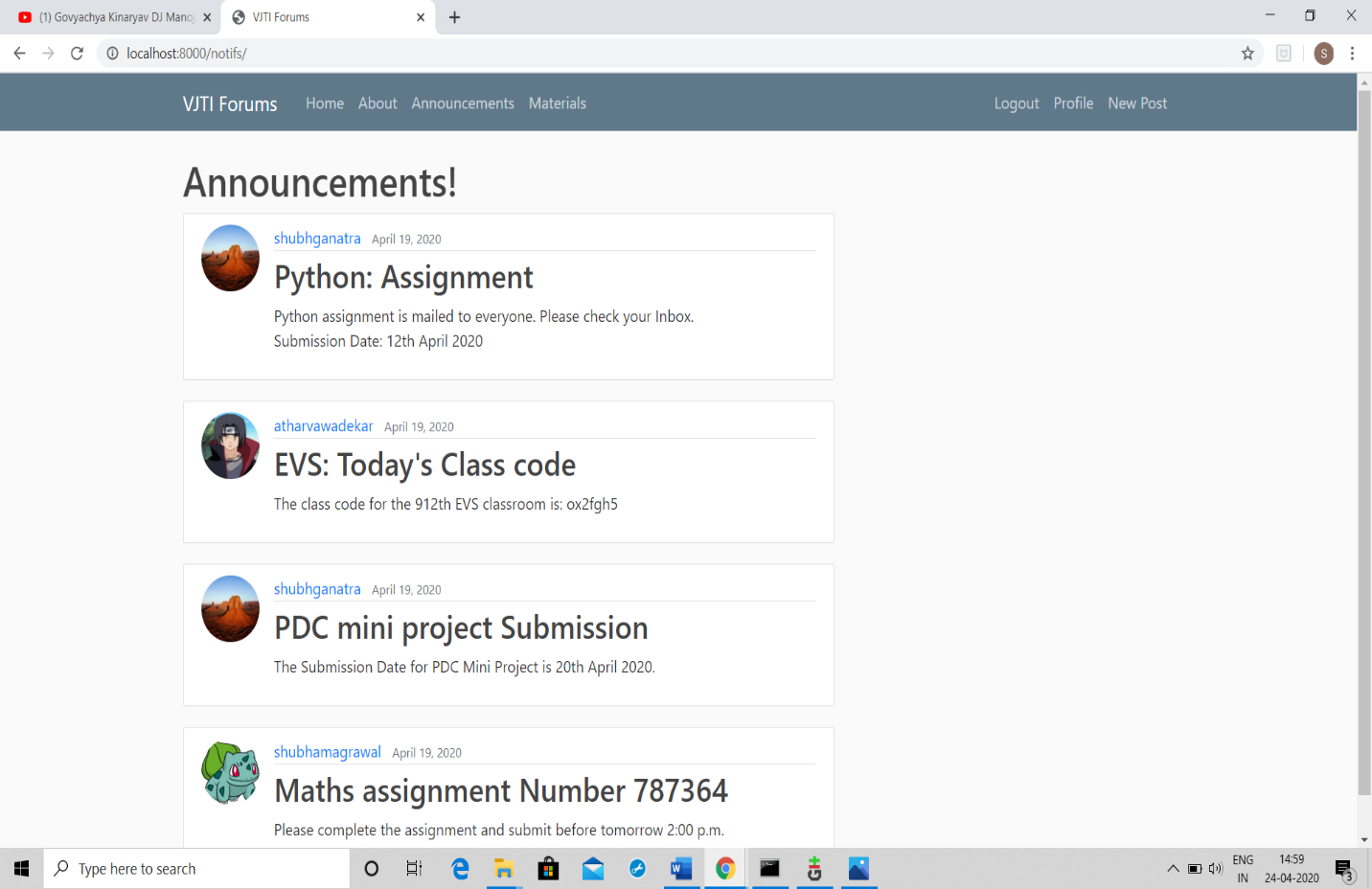
# HOME PAGE



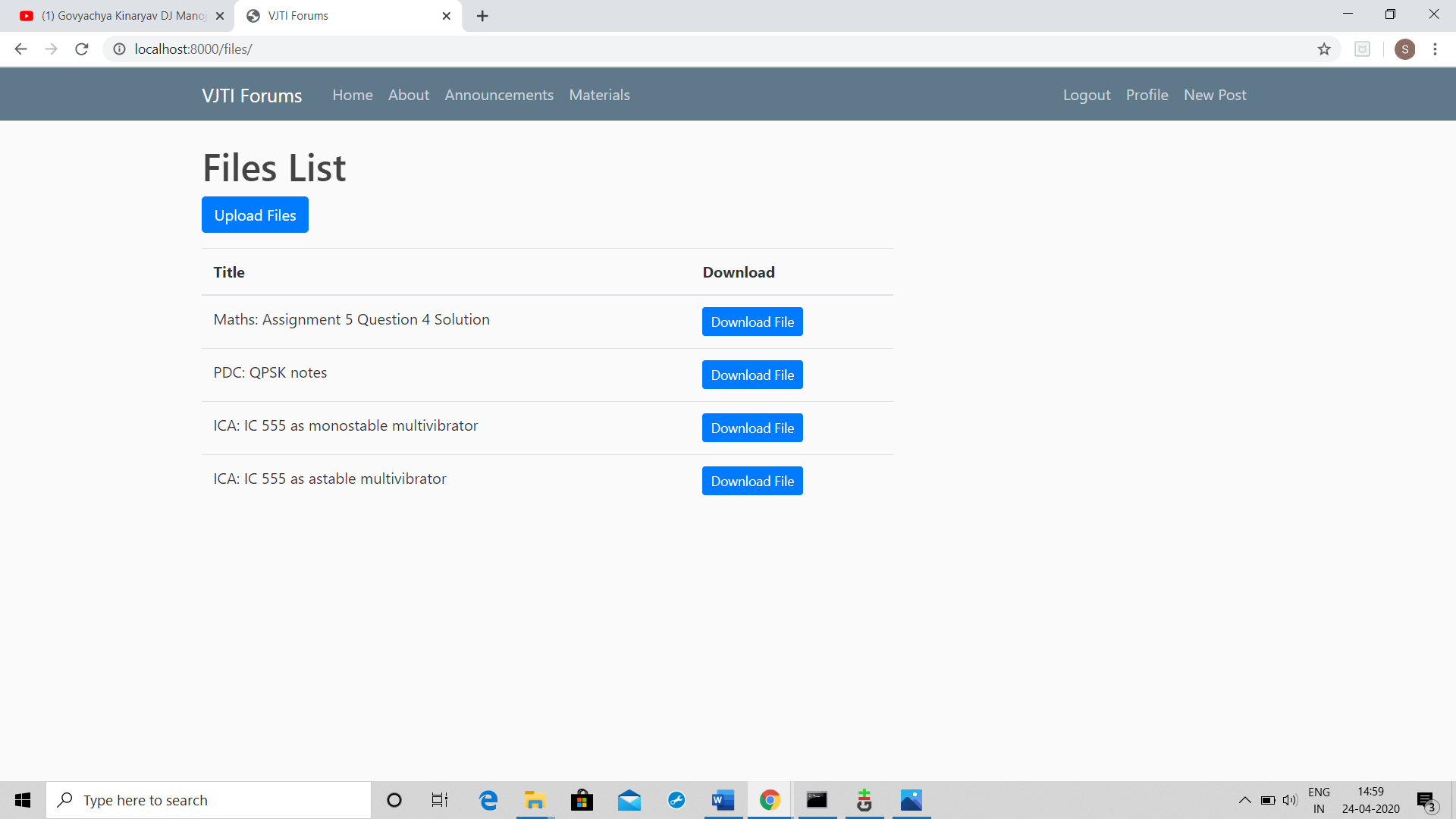
* 1. ABOUT PAGE



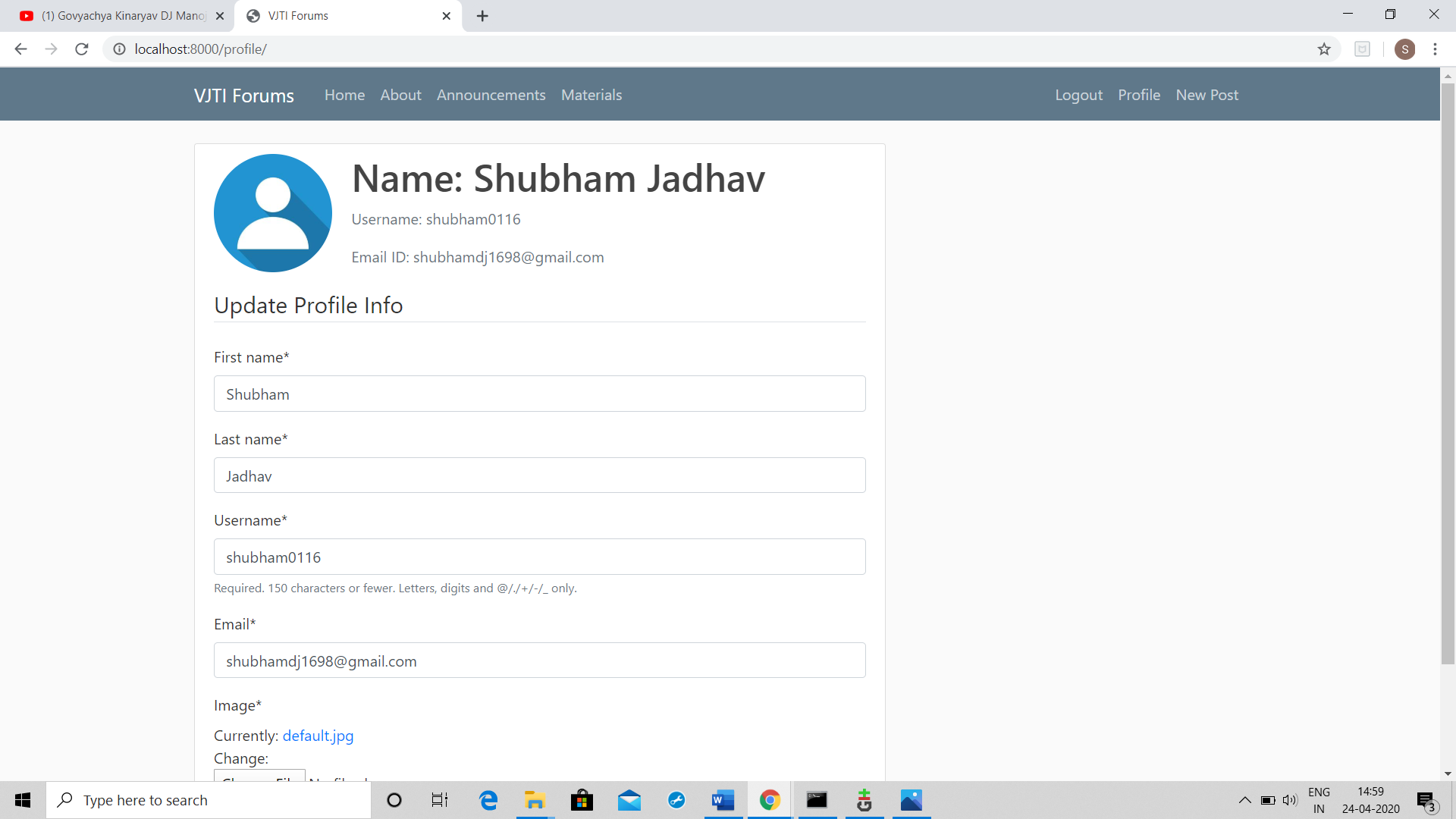
* 1. ANNOUNCEMENTS



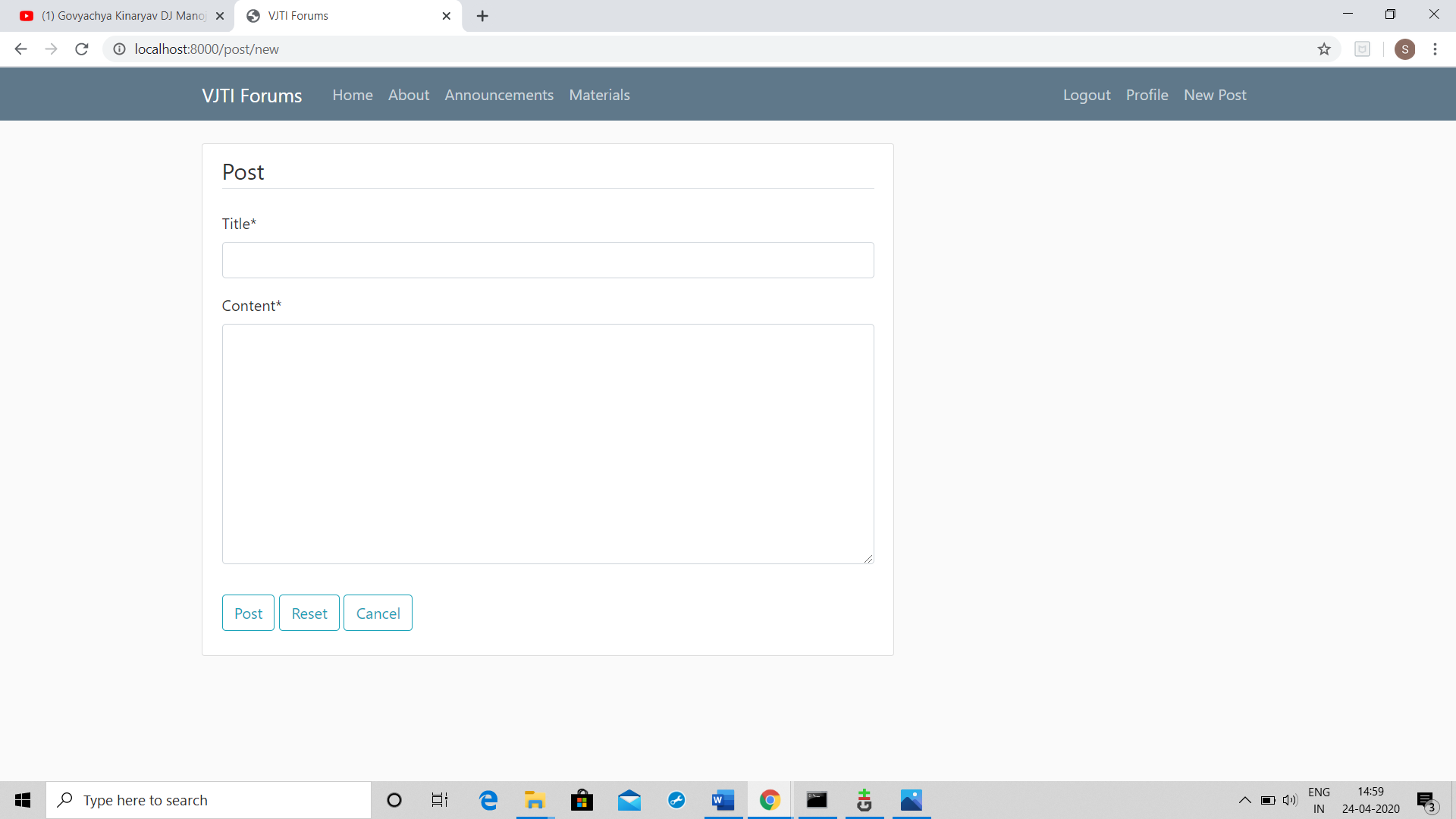
* 1. MATERIALS



* 1. PROFILE PAGE



* 1. NEW POST



* 1. LOGIN PAGE

