Deploy Python on Firebase Hosting with Cloud Run – Trang Ha

Demo website: <https://fir-django-a355c.web.app/>

Source code: https://github.com/trangHa1999/firebase-django

Contents

[**Introduction** 3](#_Toc69072810)

[**Firebase** 3](#_Toc69072811)

[Key Features 3](#_Toc69072812)

[**Google Cloud Platform (GCP)** 4](#_Toc69072813)

[Cloud Build 4](#_Toc69072814)

[Container Registry 4](#_Toc69072816)

[Cloud Run 4](#_Toc69072818)

[**I. Configuration** 4](#_Toc69072820)

[**II. Initial set up of Django project** 4](#_Toc69072821)

[Install Python 3 5](#_Toc69072822)

[Install Pip 5](#_Toc69072823)

[Create a Django project 5](#_Toc69072824)

[Templates 7](#_Toc69072825)

[Static files 8](#_Toc69072826)

[Media files 8](#_Toc69072827)

[**III. Create MVC for Homepage** 9](#_Toc69072828)

[Views 9](#_Toc69072829)

[URLs 9](#_Toc69072830)

[Templates 10](#_Toc69072831)

[**IV. Initial set up of Firebase** 13](#_Toc69072832)

[Create a project on Firebase 13](#_Toc69072833)

[Setup Firebase Authentication 14](#_Toc69072834)

[Setup Firebase Realtime Database 14](#_Toc69072835)

[Setup Firebase Storage 15](#_Toc69072836)

[Add a new app 16](#_Toc69072837)

[Configure the Connection to Firebase 16](#_Toc69072838)

[**V. Create MVC for User Authentication** 18](#_Toc69072839)

[Views 18](#_Toc69072841)

[URLs 19](#_Toc69072842)

[Templates 20](#_Toc69072843)

[**VI. Create MVC for File Management** 24](#_Toc69072844)

[Views 24](#_Toc69072845)

[URLs 26](#_Toc69072846)

[Templates 26](#_Toc69072847)

[**VII. Create MVC for Post** 28](#_Toc69072848)

[Views 28](#_Toc69072849)

[URLs 29](#_Toc69072850)

[Templates 30](#_Toc69072851)

[**VIII. Create MVC for Event** 32](#_Toc69072852)

[Views 32](#_Toc69072853)

[URLs 33](#_Toc69072854)

[Templates 33](#_Toc69072855)

[**IX. Deploy to Cloud Run** 35](#_Toc69072856)

[Initial Setup for Cloud Run 35](#_Toc69072857)

[Create a service account 36](#_Toc69072858)

[Requirement.txt and Dockerfile 38](#_Toc69072859)

[Deploy to Cloud Run 39](#_Toc69072860)

[**X. Deploy to Firebase Hosting** 40](#_Toc69072861)

[Initial Setup for Firebase 40](#_Toc69072862)

[Deploy to Firebase Hosting 40](#_Toc69072863)

[**XI. GitHub Actions** 42](#_Toc69072864)

[Setup GitHub Actions to Deploy to Firebase Hosting 42](#_Toc69072866)

[Setup GitHub Actions to Deploy to Cloud Run 42](#_Toc69072867)

[**XII. Bonus** 44](#_Toc69072868)

[Deploy to Compute Engine 44](#_Toc69072870)

[Deploy to App Engine 45](#_Toc69072871)

[Deploy to Kubernetes Engine 46](#_Toc69072872)

[**XV. Source** 48](#_Toc69072873)

# **Introduction**

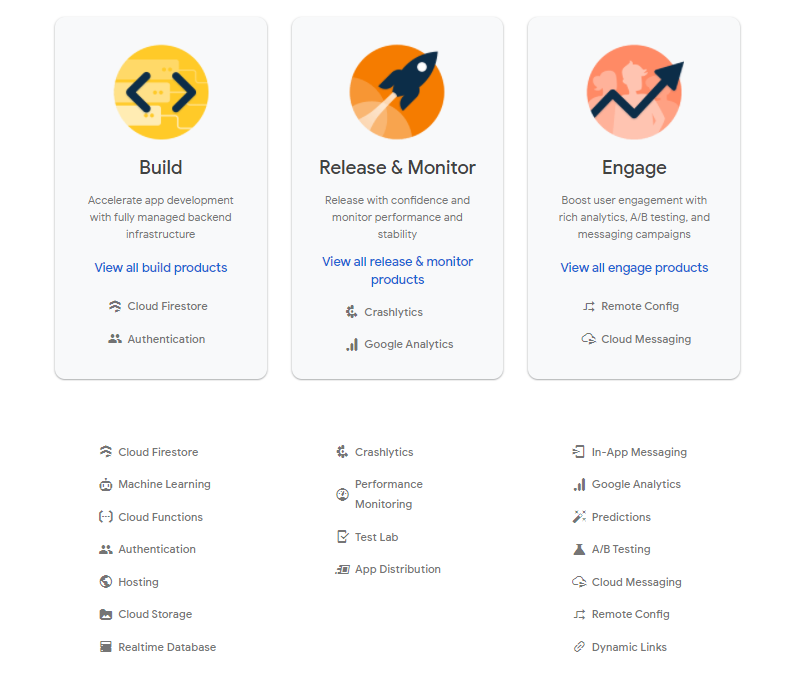
Before 2017, Firebase Hosting only hosts static content, which only runs HTML, CSS, JavaScript, and media files. However, you can now deploy dynamic or static content with any programming language on Firebase. With Firebase Hosting integrating with Cloud Run, you will package the project into Docker images and then upload it to Container Registry then use the service that supports running containers from Docker images such as Cloud Run to deploy the website. In other words, when users request the site, Firebase Hosting will check if it is dynamic or static content. If it is dynamic, Firebase will generate the dynamic parts through Cloud Run and serve back through Firebase Hosting.

# **Firebase**

Firebase is a Backend-as-a-Service (Baas) that works on cloud computing with Google’s powerful server system. It supports building, improving, and growing mobile apps and websites, including simple and powerful APIs without the need for backends or servers. Firebase also has good security and supports both Android and iOS platforms.

## Key Features

Firebase is a versatile platform that offers a wide variety of products. These products divide into three main categories: Build – deploy and build applications, Release & Monitor – develop and test applications, and Engage – analyze and optimize data for users. For this tutorial, we will focus on products that help build a website.



**Firebase Realtime Database** is a NoSQL database store in the cloud, allowing you to store and synchronize data in real-time. In case of network loss, the data will save locally and automatically update to Firebase server when users reconnect.

**Firebase Cloud Storage** allows storing and sharing content that users can upload or download to and from Firebase server, such as image, audio, video, etc.

**Firebase Authentication** is a simple and secure user management service. It offers a variety of methods to authenticate users by email and password or third-party providers likes Google, Facebook, GitHub and directly use the existing account.

**Firebase Hosting** is a secure web hosting CDN with an SSL certificate. In other words, the way it works via CDN helps Firebase Hosting to better optimize data with cloud servers located around the world, and users can get access with faster speed and better stability.

# **Google Cloud Platform (GCP)**

Google Cloud Platform is a cloud computing platform that allows you to build and develop projects on the system that Google uses for their products, such as Google Search, YouTube, Google Maps, etc. It supports the services needed to build, develop, optimize, and manage your own projects. Google Cloud Platform provides services for computation, storage, networking, big data, machine learning, internet of things (IoT), etc.

## Cloud Build

## Cloud Build is one of the services on Google Cloud Platform that allow you to build container images and store them in Container Registry. With Cloud Build, you can build your projects in any programming language and deploy the source code on GitHub, Bitbucket, Cloud Source Repositories, or local. It also works with multiple environments like Firebase, VMs, or Kubernetes.

## Container Registry

## Container Registry is a place to store and manage container images, which specify the programming language and dependencies for the projects to deploy directly on Cloud Run, Kubernetes Engine, App Engine, Cloud Function, or Firebase.

## Cloud Run

## Cloud Run is serverless computing that allows you to deploy stateless containers without configuring the server, worrying about scaling, or overpaying for overprovisioning resources.

# **I. Configuration**

On a Mac or Linux, the command line is call Terminal. To find it, search it in the search bar and double-click to open it.

On Windows, the command line is called Command Prompt, but it is difficult to use. Instead, we can install and use the Linux Bash Shell on Windows. Follow these steps to install it:

* Go to Settings > Apps > Programs and Features > Turn Windows features on or off > Check the Windows Subsystem for Linux checkbox > Restart
* Open Microsoft Store > Search and install Ubuntu

# **II. Initial set up of Django project**

You need to install some required packaged before creating a Django web app.

## Install Python 3

Check if Python 3 is already installed to your computer.

Command line

$ python3 --version

If Python 3 is not installed to your computer. Follow these steps:

Command line

Step 1 – Update and refresh repository list

$ sudo apt update

Step 2 – Install supporting software

$ sudo apt install software-properties-common

Step 3 – Add Deadsnakes PPA

$ sudo add-apt-repository ppa:deadsnake/ppa

Step 4 – Install Python

$ sudo apt install python3.9

## Install Pip

Check if Pip is already installed to your computer.

Command line

$ pip3 --version

If Pip is not installed to your computer. Follow these steps:

Command line

$ sudo apt install python3-pip

## Create a Django project

Create a new folder on Desktop for the project, but you can put your code anywhere you like on your computer.

Command line

Step 1: Direct you to your Windows drive

$ cd /mnt/c/Users/<YOUR-USERNAME>/Desktop

Step 2: Create a new folder call “firebase-django.”

$ mkdir firebase-django

Step 3: Navigate to the new folder

$ cd firebase-django

Virtual environment is an isolated container containing all the dependencies for the project. It is essential for every Python project. To create a virtual environment for project, you need to install pipenv and then activate the virtual environment.

Command line

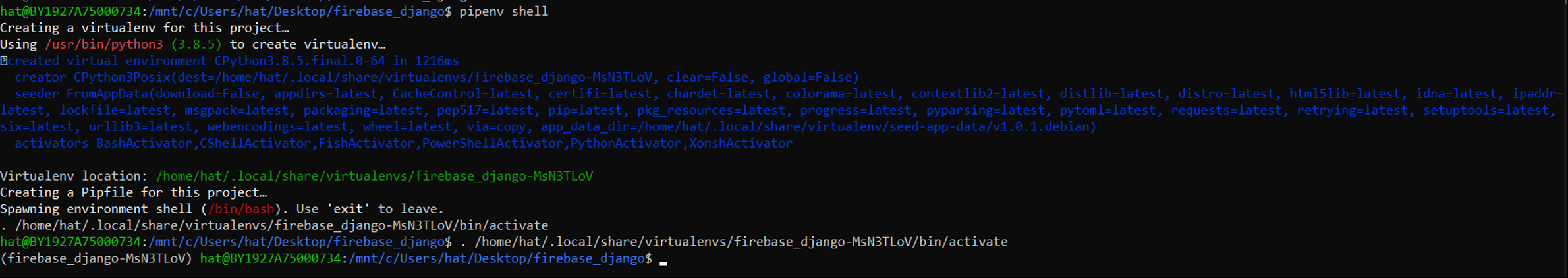
Step 1: Install pipenv

$ sudo apt install pipenv

Step 2: Activate the virtual environment

$ pipenv shell

\*\*\* The virtual environment is successfully activated if the name of virtual environment folder will be shown in parathesis



Now, you are able to install the required dependencies for the project and it will be store in the virtual environment folder. For this project, you need to install Django, Pyrebase, gunicorn, and Regex

Command line

$ pipenv install django

$ pipenv install pyrebase5

$ pipenv install regex

$ pipenv install gunicorn

To create a Django project name bulletinBoard, type this command in the command line.

Command line

$ pipenv run django-admin startproject bulletinBoard

\*\*\* Project folder structure

├── bulletinBoard

│   ├── \_\_init\_\_.py

│   ├── asgi.py

│   ├── settings.py

│   ├── urls.py

│   └── wsgi.py

└── manage.py

To see how every is working, migrate and start the development on the local webserver.

Command line

$ cd bulletinBoard

$ pipenv run python manage.py migrate

$ pipenv run python manage.py runserver

\*\*\* If your setup is correct, you should see Django welcome page when visit <http://127.0.0.1:8000/>. Click CTRL + c to quit the server.

Django: the Web framev.ork for I X 
C 0 127.0.0.1:8000 
django 
Guest 
View release notes for Django 3.1 
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.

## Templates

Templates are the folder containing HTML files that can be served by a view to a web page specified by the URL. To setup templates folder, follow these steps:

Step 1: Create a project-level folder called templates

Command line

$ mkdir templates

Step 2: Update settings.py to tell Django to look at the project-level for templates

Code

# bulletinBoard/settings.py

**from**pathlib **import**Path

**import** os #new

...

TEMPLATES = [

{  …

**'DIRS'**: [os.path.join(BASE\_DIR, **'templates'**)],

    …

}]

## Static files

Static files refer to as CSS and JavaScript. Django project needs a folder to store all those files in one place, and it can be served when called. To setup static folder, follow these steps:

Step 1: Create a project-level folder called static

Command line

$ mkdir static

Step 2: Update settings.py to tell Django where to look for static files

Code

# bulletinBoard/settings.py

...

STATICFILES\_DIRS = [os.path.join(BASE\_DIR, **'static'**)]

Step 3: Add some CSS

Command line

$ mkdir static/css

$ nano static/css/base.css

\*\*\* For this project, we use mostly Bootstrap to save some time and effort.

## Media files

Media files refer to as images, pdf, doc files, etc. Django project needs a folder to store all those files in one place, and it can be served when called. To setup media folder, follow these steps:

Step 1: Create a project-level folder called media

Command line

$ mkdir media

Step 2: Update settings.py to tell Django where to look for media files

Code

# bulletinBoard/settings.py

...

MEDIA\_ROOT = os.path.join(BASE\_DIR, **'media'**)

MEDIA\_URL = **'/media/'**

Step 3: Update project-level urls.py

Code

# bulletinBoard/urls.py

**from**django.contrib **import**admin

**from** django.urls **import**path

**from** django.conf **import**settings #new

**from** django.conf.urls.static **import**static #new

urlpatterns = [

...

] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT) # new

Step 4: Create a new directory within the media folder called image and add all images that will be used for this project

Command line

$ mkdir media/image

\*\*\* For this project, download image from <https://github.com/trangHa1999/firebase-django/tree/master/media/image>

# **III. Create MVC for Homepage**

In this section, we will build a homepage for the Django website with Django’s class-based views and templates. In Django, Views determine the content and template are displayed on a given page, while URLs determine where that content is going. In other words, when users request a site, the URL will map that request to the appropriate view function, which determines the content for the page and template for styling.

## Views

Create views.py in the project-levels folder

Command line

$ nano bulletinBoard/views.py

In the views.py, add the code below to control the display of the homepage site.

Code

# bulletinBoard/views.py

**from** django.shortcuts **import** render

**def** homepage(request):  
 **return** render(request, **"index.html"**)

## URLs

Update the project-level urls.py

Code

# bulletinBoard/urls.py

**from** django.contrib **import** admin  
**from** django.urls **import** path  
**from** django.conf **import** settings  
**from** django.conf.urls.static **import** static

**from** . **import** views #new

urlpatterns = [

…

path('', views.hompage, name='home'),

] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

## Templates

Add base.html in templates and populate it with HTML code

Command line

$ nano templates/base.html

Code

# templates/base.html

{% **load static** %}  
<**html lang="en"**>  
<**head**>  
 *<!--Required meta tags-->* <**meta charset="UTF-8"**>  
 <**meta name="viewport" content="width=device-width,initial-scale=1.0"**>  
 <**title**>{% **block title** %}Bulletin Board{% **endblock title** %}</**title**>  
 *<!--local CSS-->* <**link rel="stylesheet" href="**{% **static 'css/stylesheet.css'** %}**"**>  
 *<!--Google fonts-->* <**link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro:400" rel="stylesheet"**>  
 *<!-- Bootstrap CSS -->* <**link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous"**>  
  
 *<!-- Optional JavaScript -->  
 <!-- jQuery first, then Popper.js, then Bootstrap JS -->* <**script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"**></**script**>  
 <**script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"**></**script**>  
 <**script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"**></**script**>  
</**head**>  
<**body**>  
<**header**>  
 {% **block navbar** %}  
 *<!-- Navigation Bar -->* <**nav class="navbar navbar-expand-lg navbar-light bg-light static-top"**>  
 *<!-- Logo -->* <**a class="navbar-brand" href="**{% **url 'home'** %}**"**>  
 <**img src="/media/image/WIT-Logo.png" alt="logo" width="60" height="60"**>  
 </**a**>  
 *<!-- Logo Brand -->* <**a class="navbar-brand font-weight-bold" href="**{% **url 'home'** %}**"**>Bulletin Board</**a**>  
 *<!-- Hamburger menu icon -->* <**button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarResponsive" aria-controls="navbarResponsive" aria-expanded="false" aria-label="Toggle navigation"**>  
 <**span class="navbar-toggler-icon"**></**span**>  
 </**button**>  
 <**div class="collapse navbar-collapse" id="navbarResponsive"**>  
 <**ul class="navbar-nav ml-auto"**>  
 *<!-- Home -->* <**li class="nav-item active"**>  
 <**a class="nav-link" href="**#**"**>Home</**a**>  
 </**li**>  
 *<!-- Archive (dropdown) -->* <**li class="nav-item dropdown"**>  
 <**a class="nav-link dropdown-toggle" id="navbarDropdown"  
 data-toggle="dropdown"  
 aria-expanded="false"**>Archive</**a**>  
 <**div class="dropdown-menu"**>  
 <**a class="dropdown-item" href="**#**"**>Document</**a**>  
 <**a class="dropdown-item" href="**#**"**>Post</**a**>  
 </**div**>  
 </**li**>  
 *<!-- Event -->* <**li class="nav-item"**>  
 <**a class="nav-link" href="**#**"**>Event</**a**>  
 </**li**>  
 *<!-- User -->* <**li class="nav-item"**>  
 <**a class="nav-link" href="**#**"**>Sign Out</**a**>  
 </**li**>  
 </**ul**>  
 </**div**>  
 </**nav**>  
 {% **endblock navbar** %}  
</**header**>  
<**div class="container"**>  
 {% **block content** %}  
 {% **endblock content** %}  
</**div**>  
</**body**>  
*<!-- Footer -->*<**footer class="p-3 p-md-5 mt-5 bg-light text-center"**>  
 <**div class="container"**>  
 <**p class="mb-0"**>© 2021 Copyright: Trang Ha. All rights reserved. Content used with permission of author. Wentworth Institute of Technology logo used with permission of Wentworth Institute of Tehnology</**p**>  
 </**div**>  
</**footer**>  
</**html**>

Add index.html in templates and populate it with HTML code

Command line

$ nano templates/index.html

Code

# templates/index.html

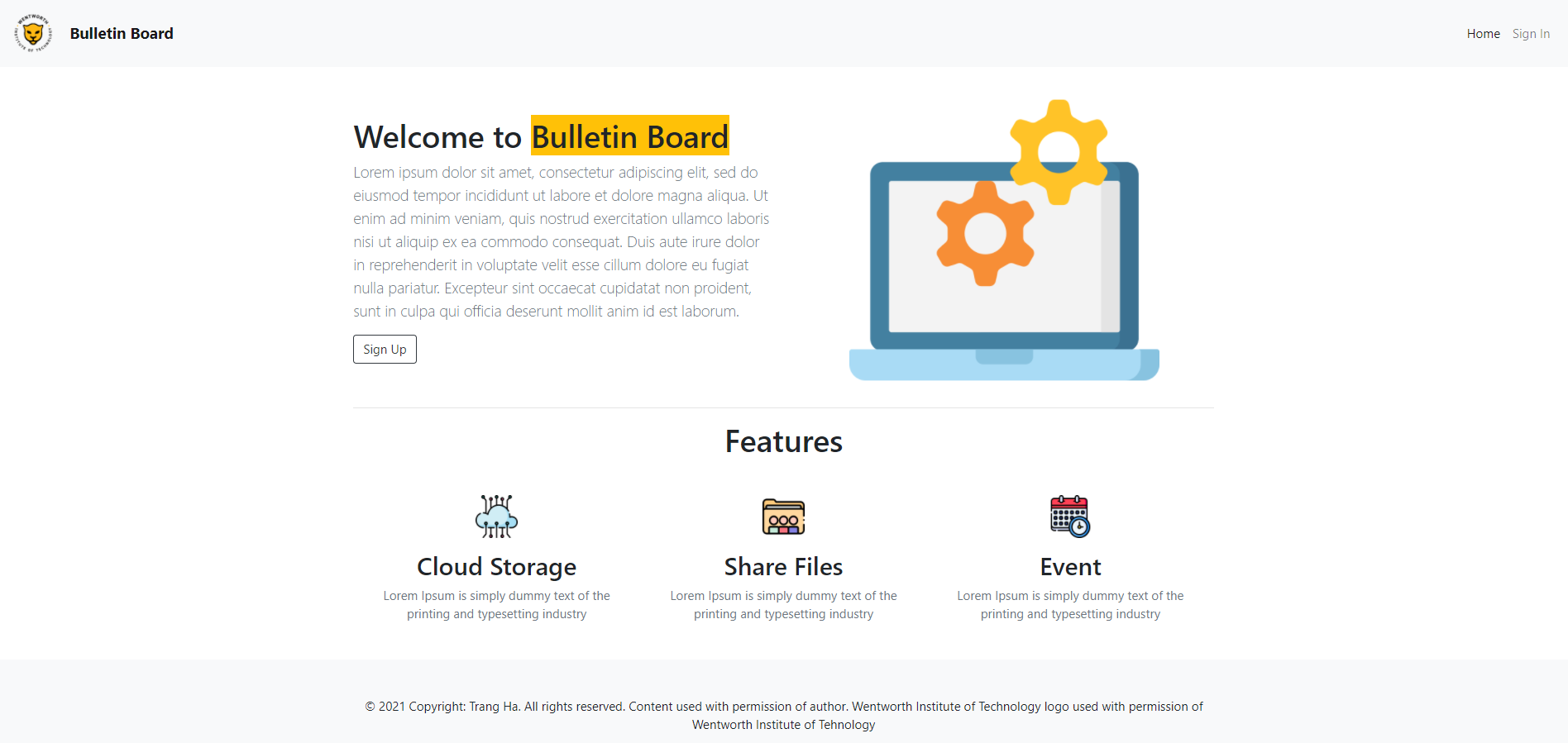
{% **extends 'base.html'** %}  
{% **block navbar** %}  
 *<!-- Navigation Bar -->* <**nav class="navbar navbar-expand-lg navbar-light bg-light static-top"**>  
 *<!-- Logo -->* <**a class="navbar-brand" href="**{% **url 'home'** %}**"**>  
 <**img src="/media/image/WIT-Logo.png" alt="logo" width="60" height="60"**>  
 </**a**>  
 *<!-- Logo Brand -->* <**a class="navbar-brand font-weight-bold" href="**{% **url 'home'** %}**"**>Bulletin Board</**a**>  
 *<!-- Hamburger menu icon -->* <**button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarResponsive" aria-controls="navbarResponsive" aria-expanded="false" aria-label="Toggle navigation"**>  
 <**span class="navbar-toggler-icon"**></**span**>  
 </**button**>  
 <**div class="collapse navbar-collapse" id="navbarResponsive"**>  
 <**ul class="navbar-nav ml-auto"**>  
 *<!-- Home -->* <**li class="nav-item active"**>  
 <**a class="nav-link" href="**{% **url 'home'** %}**"**>Home</**a**>  
 </**li**>  
 *<!-- User -->* <**li class="nav-item"**>  
 <**a class="nav-link" href="**#**"**>Sign In</**a**>  
 </**li**>  
 </**ul**>  
 </**div**>  
 </**nav**>  
{% **endblock navbar** %}  
{% **block content** %}  
 *<!--- Welcome -->* <**section class="pt-4 pt-md-11"**>  
 <**div class="row align-items-center"**>  
 <**div class="col-12 col-md-5 col-lg-6 order-md-2"**>  
 <**img src="/media/image/data-management.png" class="mx-auto d-block" width="400" height="400" alt="welcome"**>  
 </**div**>  
 <**div class="col-12 col-md-7 col-lg-6 order-md-1 aos-init aos-animate" data-aos="fade-up"**>  
 *<!-- Heading -->* <**h1 class="display-6 text-center text-md-left"**>  
 Welcome to <**span class="bg-warning"**>Bulletin Board</**span**>  
 </**h1**>  
 *<!-- Text -->* <**p class="lead text-center text-md-left text-muted mb-6 mb-lg-8"**>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</**p**>  
 *<!-- Buttons -->* <**button class="btn btn-outline-dark" type="button">**Sign Up</**button**>  
 </**div**>  
 </**div**>  
 </**section**>  
 <**hr**>  
  
 *<!-- Features -->* <**section class="py-8 py-md-11"**>  
 <**div class="container"**>  
 <**h1 class="text-center mt-2 pd-2"**>Features</**h1**>  
 <**div class="row text-center mt-5"**>  
 <**div class="col-12 col-md-4 aos-init aos-animate"**>  
 *<!-- Icon -->* <**div class="icon text-primary mb-3"**>  
 <**img src="/media/image/cloud-storage.png" width="55" height="55"**>  
 </**div**>  
 *<!-- Heading -->* <**h2**>Cloud Storage</**h2**>  
 *<!-- Text -->* <**p class="text-muted mb-6 mb-md-0"**>Lorem Ipsum is simply dummy text of the printing and typesetting industry</**p**>  
 </**div**>  
 <**div class="col-12 col-md-4 aos-init aos-animate"**>  
 *<!-- Icon -->* <**div class="icon text-primary mb-3"**>  
 <**img src="/media/image/share-file.png" width="55" height="55"**>  
 </**div**>  
 *<!-- Heading -->* <**h2**>Share Files</**h2**>  
 *<!-- Text -->* <**p class="text-muted mb-6 mb-md-0"**>Lorem Ipsum is simply dummy text of the printing and typesetting industry</**p**>  
 </**div**>  
 <**div class="col-12 col-md-4 aos-init aos-animate"**>  
 *<!-- Icon -->* <**div class="icon text-primary mb-3"**>  
 <**img src="/media/image/schedule.png" width="55" height="55"**>  
 </**div**>  
 *<!-- Heading -->* <**h2**>Event</**h2**>  
 *<!-- Text -->* <**p class="text-muted mb-6 mb-md-0"**>Lorem Ipsum is simply dummy text of the printing and typesetting industry</**p**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**section**>  
{% **endblock content** %}

Start the development server

Command line

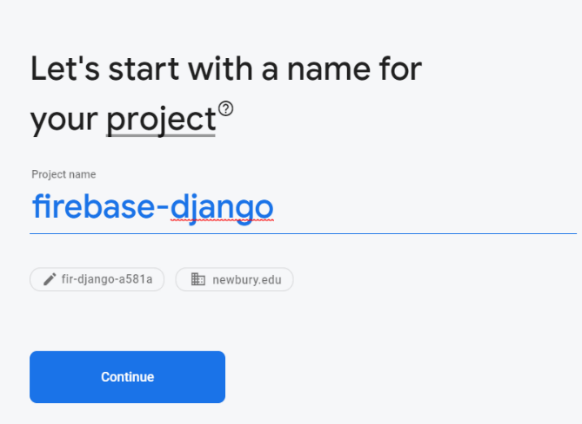
$ pipenv run python manage.py runserver

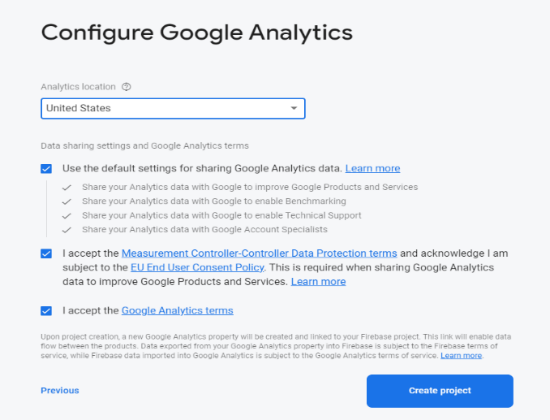
\*\*\* If your setup is correct, you should see the page looks like this when visit <http://127.0.0.1:8000/>



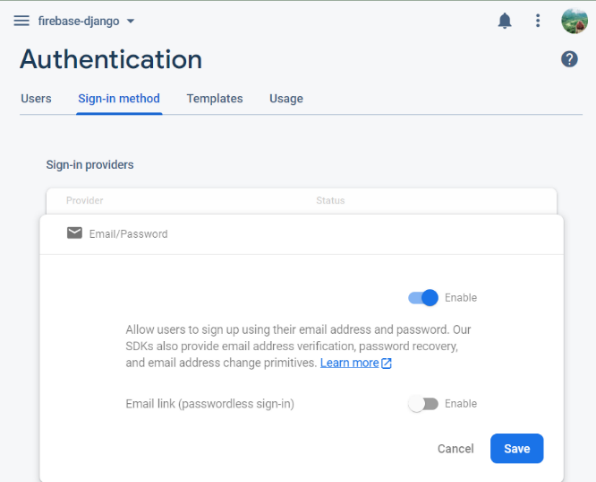
# **IV. Initial set up of Firebase**

## Create a project on Firebase

Step 1: Go to <https://firebase.google.com/> > Login to Google Account > Create a project > Enter project name > Continue

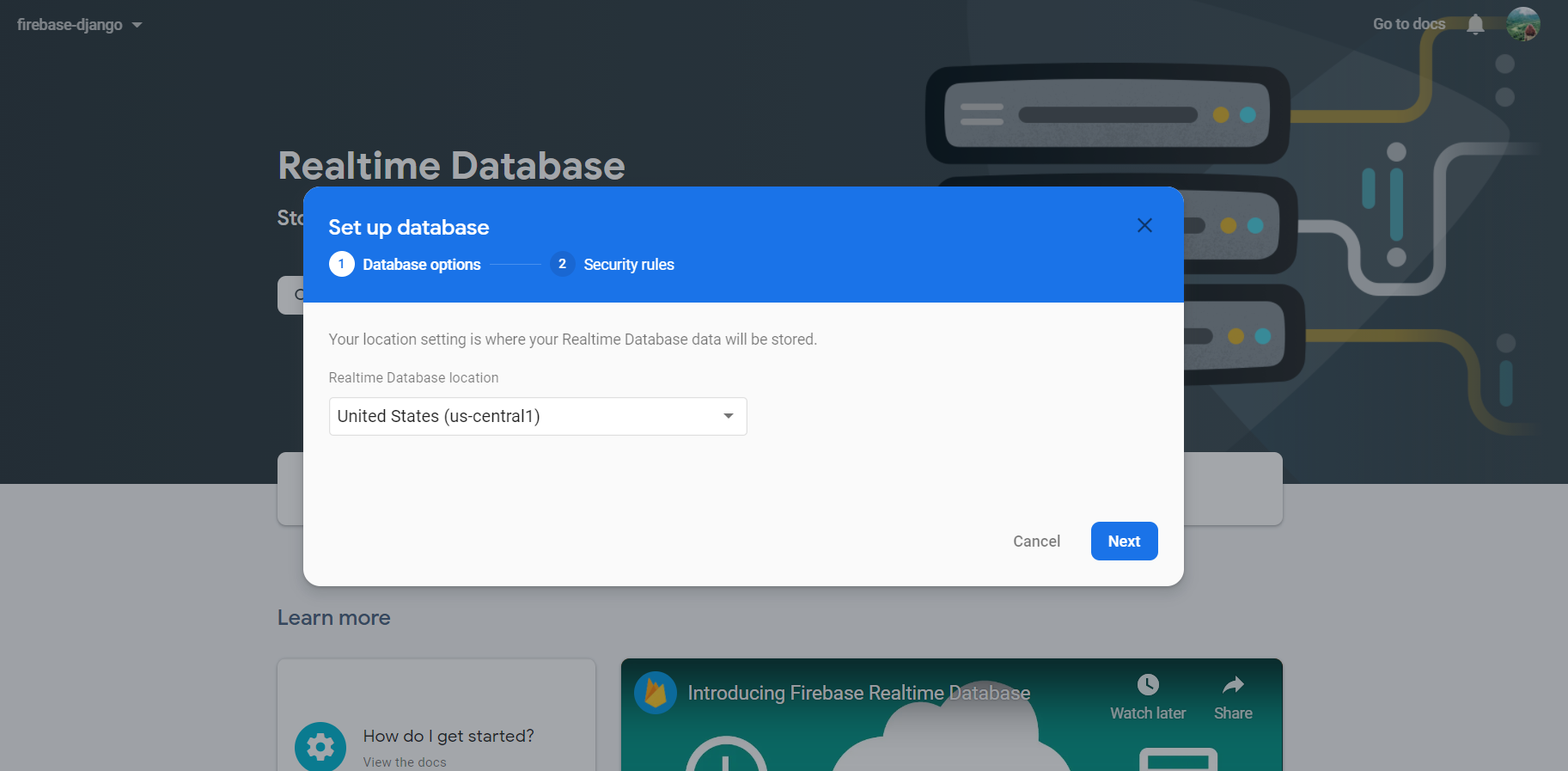
Step 2: Enable Google Analytics for this project > Continue > Choose location and check all the checkboxes > Create project

## Setup Firebase Authentication

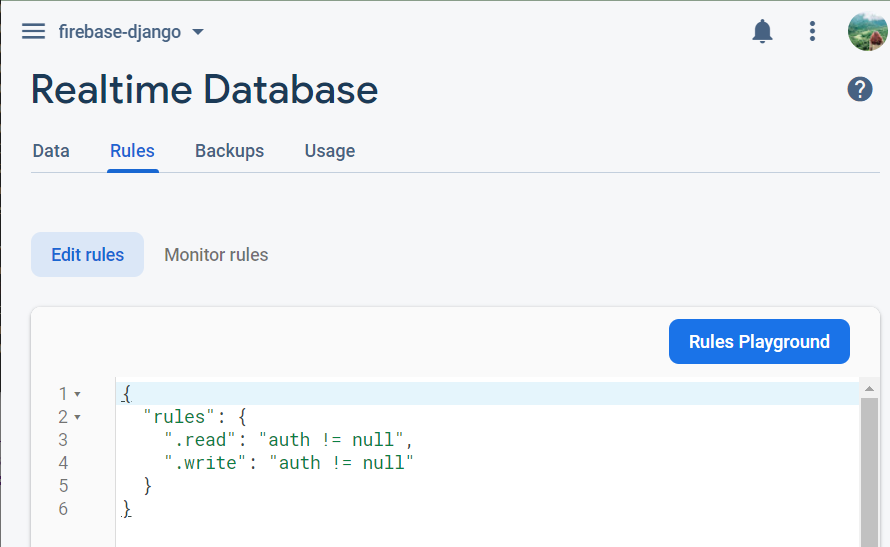
Go to Authentication > Get started > Sign-in-method > Enable Email/Password > Save

## Setup Firebase Realtime Database

Step 1: Go to Realtime Database > Create Database > Choose database location > Next > Enable



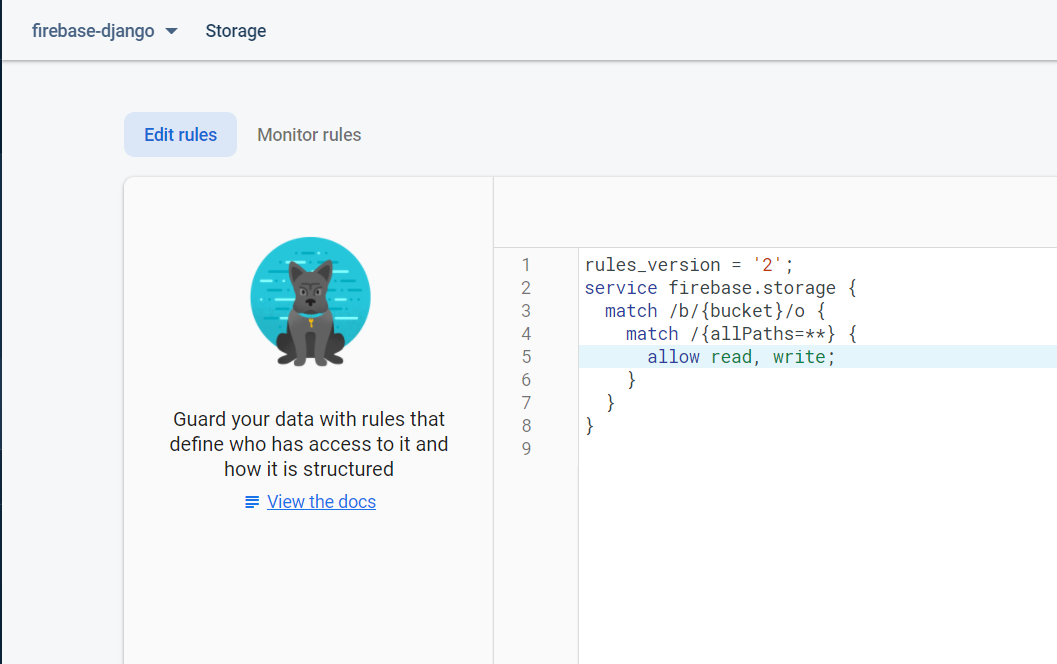
Step 2: Click on Rules > Edit rule > Publish

\*\*\* This rule will set the database allow only authenticated users in Firebase Authentication to access and write data.

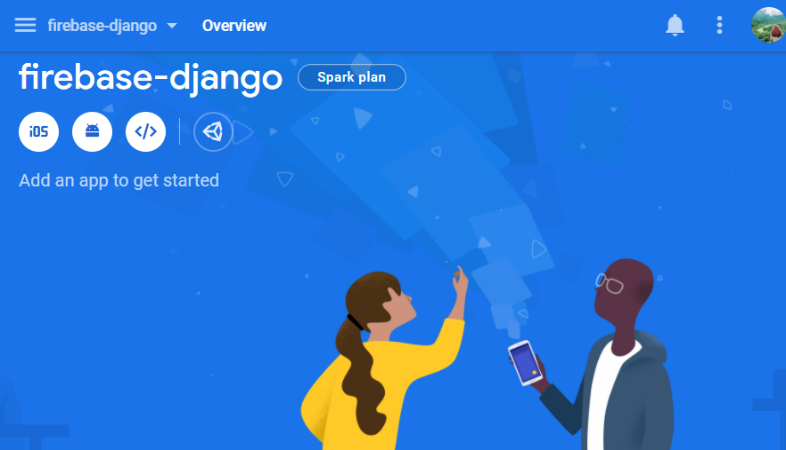
## Setup Firebase Storage

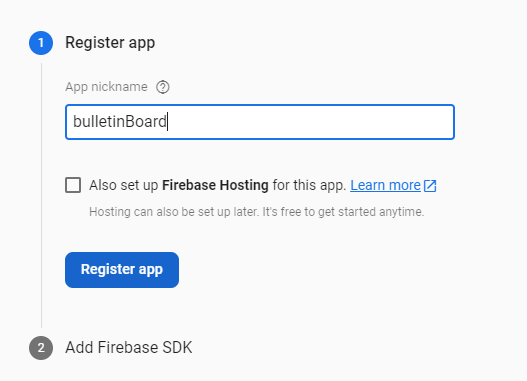
Machine generated alternative text:
firebase-django 
Storage 
Set up Cloud Storage 
Store and ret' 
Secure rules for Cloud Storage 
images, audic 
side code 
Go to docs 
Set Cloud Storage location 
Your location setting is where your default Cloud Storage bucket and its data will be stored. 
Get started 
A After you set this location, you cannot change it later. This location setting will 
also be the default location for Cloud Firestore. 
Cloud Storage location 
nam5 (us-central) 
Learn more 
Blaze Plan customers can choose other locations for additional buckets 
How 
View the docs 
How does Storage work? 
Learn more 
Cancel 
Done 
• Watch latéK 
Share Step 1: Go to Storage > Get started > Next > Choose storage location > Done

Step 2: Click on Rules > Edit rule > Publish

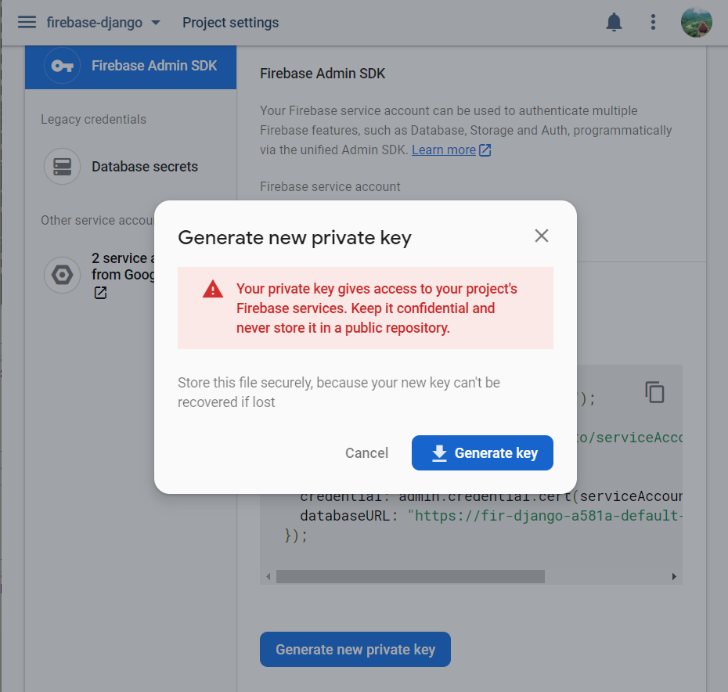


## Add a new app

Step 1: Click on </> icon to create a new web app

Step 2: Enter the app nickname > Register app > Continue to console

## Configure the Connection to Firebase

Step 1: Go to Project settings > Service accounts > Generate new private key > Generate key > Move your service account key to project-level folder

\*\*\* Project folder structure

├── bulletinBoard

│   ├── \_\_init\_\_.py

│   ├── asgi.py

│   ├── settings.py

│   ├── urls.py

│   ├── views.py

│   └── wsgi.py

├── db.sqlite3

├──<SERVICE-ACCOUNT-KEY>.json

├── manage.py

├── media

│   └── image

│   ├── WIT-Logo.png

│   ├── cloud-storage.png

│   ├── data-management.png

│   ├── schedule.png

│   └── share-file.png

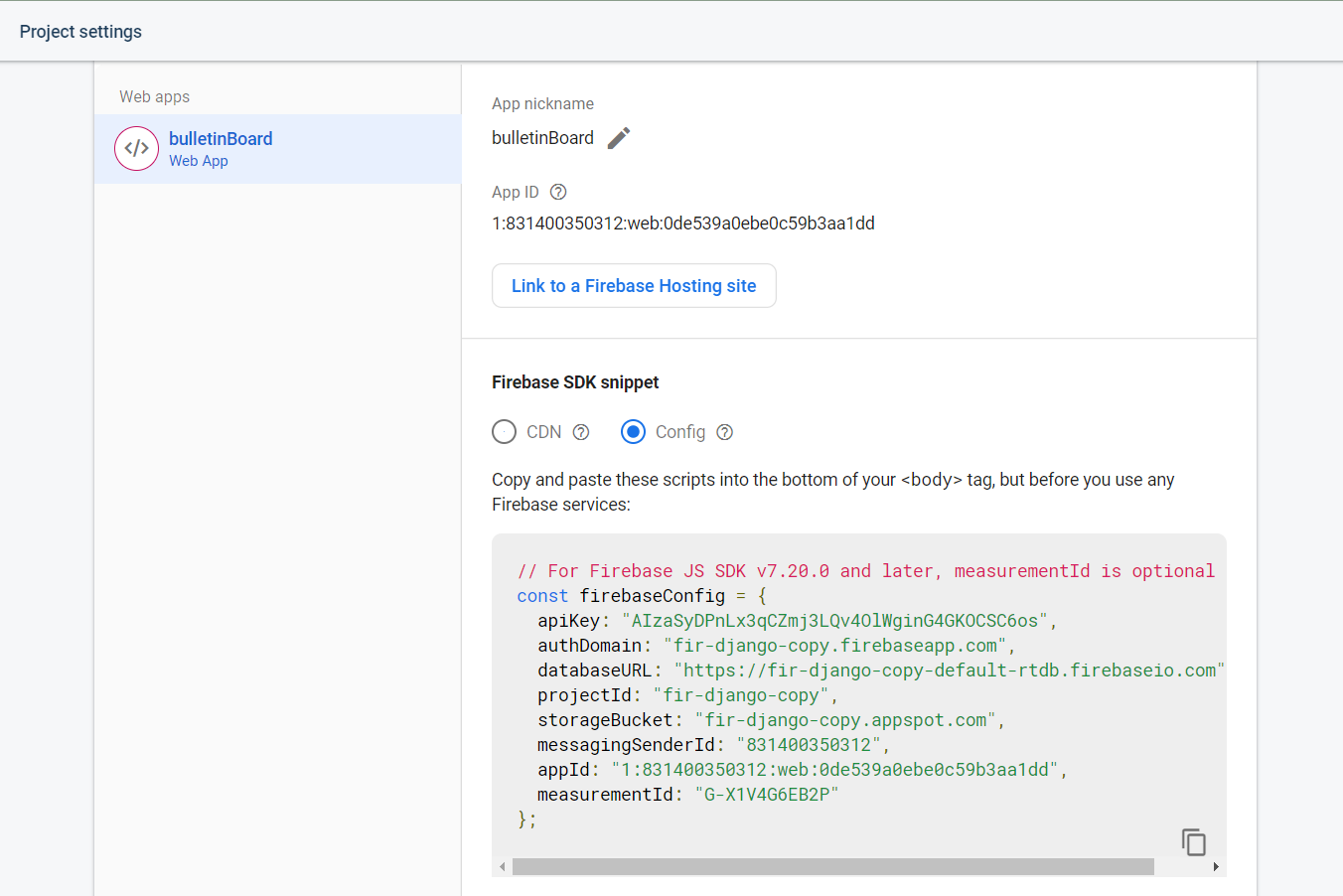
├── static

├── templates

├── base.html

└── index.html

Step 2: Go to Project settings > General > Your apps > Firebase SDK snippet > Check on Config > Copy the script



Step 3: Update views.py in project-level

* Paste the script to views.py and make some edit on the code so that your version will look like the following
* Copy your service account key json file name and paste it in **<YOUR-SERVICE-ACCOUNT-KEY.json>**

Code

# bulletinBoard/views.py

**from** django.shortcuts **import** render

**import** pyrebase #new

config = {

**"apiKey"**: **"<YOUR-API-KEY>"**,

**"authDomain"**: **"<YOUR-AUTH-DOMAIN>"**,

**"databaseURL"**: **"<YOUR-DATABASE-URL>"**,

**"projectId"**: **"<YOUR-PROJECTID>"**,

**"storageBucket"**: **"<YOUR-STORAGE-BUCKET>"**,

**"messagingSenderId"**: **"<YOUR-MESSAGING-SENDERID>"**,

**"appId"**: **"<YOUR-APPID>"**,

**"measurementId"**: **"<YOUR-MEASUREMENTID>"**,

**"serviceAccount"**: **"<YOUR-SERVICE-ACCOUNT-KEY>.json"**,

}

*#Initializingdatabase,authandfirebaseforfurtheruse*

firebase=pyrebase.initialize\_app(config)

authe=firebase.auth()

database=firebase.database()

storage=firebase.storage()

…

# **V. Create MVC for User Authentication**

## We have built an interface for the homepage, but we need to create the most major web applications: user authentication (signing in, signing out, signing up, and reset password). This section will show you how to add a new account to Firebase Authentication and store account information in Firebase Realtime Database whenever users sign up.

## Views

Update views.py

Code

# bulletinBoard/views.py

**from** django.shortcuts **import** render, redirect #new  
**from** django.contrib **import** auth #new  
**import** pyrebase  
*…***def** signIn(request):  
 **return** render(request, **"signIn.html"**)  
  
**def** postsignIn(request):  
 email = request.POST.get(**'email'**)  
 pw = request.POST.get(**'pass'**)  
 **try**:  
 *# if there is no error then signin the user with given email and password* user = authe.sign\_in\_with\_email\_and\_password(email, pw)  
 **except**:  
 message = **"Your account or password is incorrect. If you don't remember your password, reset it now."  
 return** render(request, **"signIn.html"**, {**"message"**: message})  
 session\_id = user[**'idToken'**]  
 request.session[**'uid'**] = str(session\_id)  
 **return** render(request, **"welcome.html"**, {**"email"**: email})  
  
**def** signOut(request):  
 **try**:  
 **del** request.session[**'uid'**]  
 **except**:  
 **pass  
 return** render(request, **"index.html"**)  
  
**def** signUp(request):  
 **return** render(request, **"signUp.html"**)  
  
**def** postsignUp(request):  
 username = request.POST.get(**'username'**)  
 email = request.POST.get(**'email'**)  
 pw = request.POST.get(**'pass'**)  
 **try**:  
 *# creating a user with the given email and password* user = authe.create\_user\_with\_email\_and\_password(email, pw)  
 uid = user[**"localId"**]  
 data = {**"username"**: username, **"email"**: email, **"pass"**: pw}  
 database.child(**"users"**).child(uid).set(data)  
 message = **"Successfully create new account"  
 return** render(request, **"signIn.html"**, {**"message"**: message})  
 **except**:  
 message = **"Unable to create account"  
 return** render(request, **"signUp.html"**, {**"message"**: message})  
  
**def** reset(request):  
 **return** render(request, **"reset.html"**)  
  
**def** postReset(request):  
 email = request.POST.get(**'email'**)  
 **try**:  
 authe.send\_password\_reset\_email(email)  
 message = **"A email to reset password is successfully sent"  
 return** render(request, **"reset.html"**, {**"message"**: message})  
 **except**:  
 message = **"Something went wrong, Please check the email you provided is registered or not"  
 return** render(request, **"reset.html"**, {**"message"**: message})

## URLs

Update urls.py

Code

# bulletinBoard/urls.py  
…  
urlpatterns = [  
 …path(**'signIn/'**, views.signIn, name=**"signin"**),  
 path(**'signUp/'**, views.signUp, name=**"signup"**),  
 path(**'signOut/'**, views.signOut, name=**"signout"**),  
 path(**'reset/'**, views.reset, name=**"reset"**),  
 path(**'postReset/'**, views.postReset),  
 path(**'postsignIn/'**, views.postsignIn),  
 path(**'postsignUp/'**, views.postsignUp),  
] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

## Templates

Create signIn.html in templates and populate it with HTML code

Command line

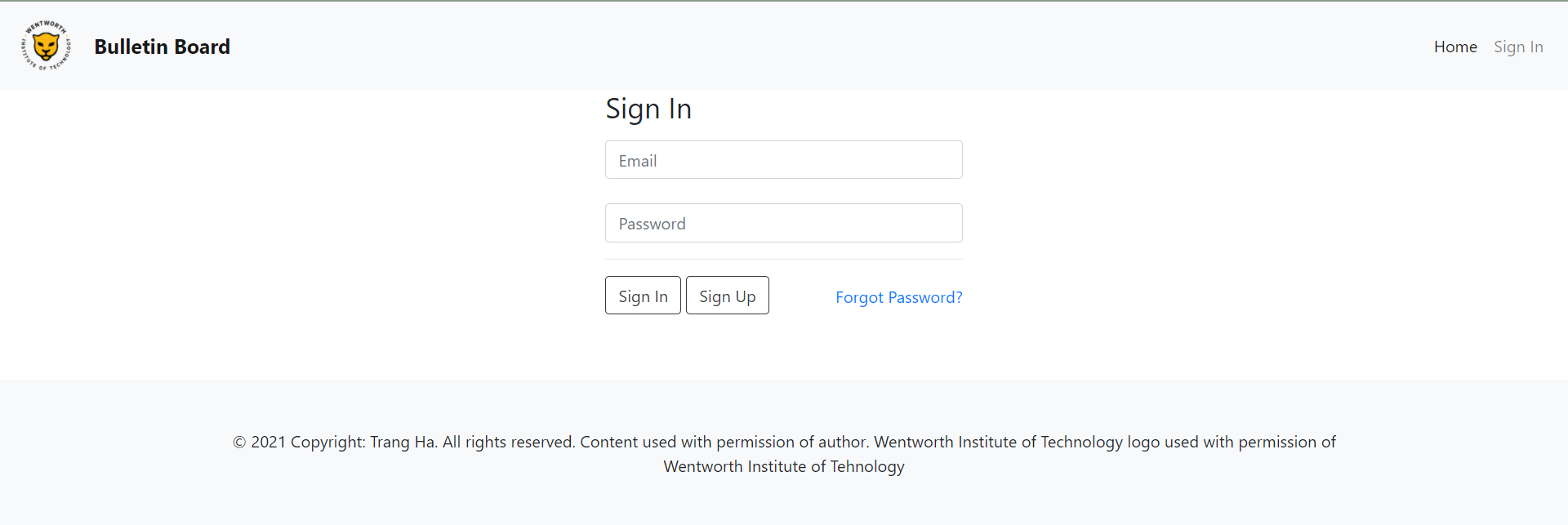
$ nano templates/signIn.html

Code

# templates/signIn.html

{% **extends 'base.html'** %}  
{% **block navbar** %}  
 *<!-- Navigation Bar -->* <**nav class="navbar navbar-expand-lg navbar-light bg-light static-top"**>  
 *<!-- Logo -->* <**a class="navbar-brand" href="**{% **url 'home'** %}**"**>  
 <**img src="/media/image/WIT-Logo.png" alt="logo" width="60" height="60"**>  
 </**a**>  
 *<!-- Logo Brand -->* <**a class="navbar-brand font-weight-bold" href="**{% **url 'home'** %}**"**>Bulletin Board</**a**>  
 *<!-- Hamburger menu icon -->* <**button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarResponsive" aria-controls="navbarResponsive" aria-expanded="false" aria-label="Toggle navigation"**>  
 <**span class="navbar-toggler-icon"**></**span**>  
 </**button**>  
 <**div class="collapse navbar-collapse" id="navbarResponsive"**>  
 <**ul class="navbar-nav ml-auto"**>  
 *<!-- Home -->* <**li class="nav-item active"**>  
 <**a class="nav-link" href="**{% **url 'home'** %}**"**>Home</**a**>  
 </**li**>  
 *<!-- User -->* <**li class="nav-item"**>  
 <**a class="nav-link" href="**{% **url 'signin'** %}**"**>Sign In</**a**>  
 </**li**>  
 </**ul**>  
 </**div**>  
 </**nav**>  
{% **endblock navbar** %}  
{% **block content** %}  
 {% **if message** %}  
 <**div class="alert alert-warning" role="alert"**><**p class="text-center"**>{{ **message** }}</**p**></**div**>  
 {% **endif** %}  
 <**div class="row"**>  
 <**div class="col-md-4 offset-md-4"**>  
 <**form class="form-signin" action="/postsignIn/" method="post"**>  
 {% **csrf\_token** %}  
 <**h1 class="h3 mb-3 font-weight-normal"**>Sign In</**h1**>  
 *<!-- Enter email -->* <**input type="email" id="Email" name="email" class="form-control" placeholder="Email"**><**br**>  
 *<!-- Enter password -->* <**input type="password" id="Password" name="pass" class="form-control" placeholder="Password"**><**hr**>  
  
 <**button class="btn btn-outline-dark" type="submit" onclick="*location***.href=**'**{% **url 'signin'** %}**'"**>Sign In</**button**>  
 <**button class="btn btn-outline-dark" type="button" onclick="**location.href=**'**{% **url 'signup'** %}**'"**>Sign  
 Up</**button**>  
 <**a class="pt-2 float-right" href="/reset/"**>Forgot Password?</**a**>  
 </**form**>  
 </**div**>  
 </**div**>  
{% **endblock content** %}

\*\*\* signIn.html will look like this



Create welcome.html in templates and populate it with HTML code

Command line

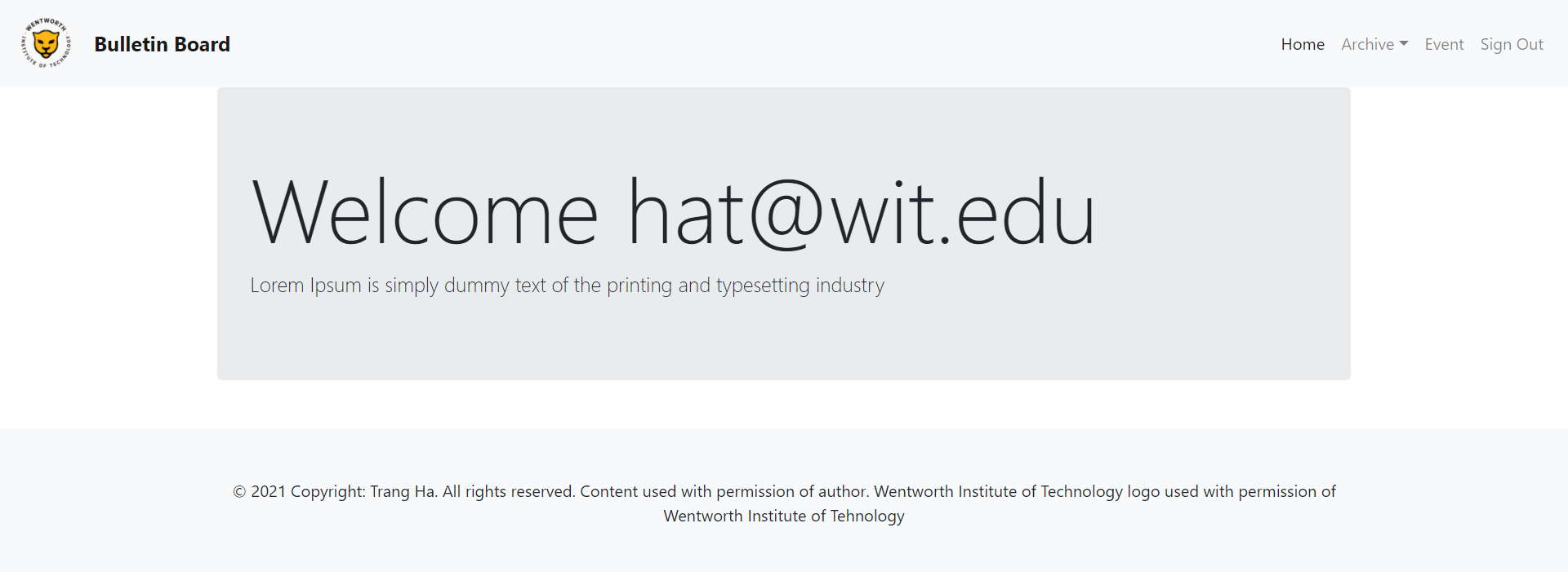
$ nano templates/welcome.html

Code

# templates/signIn.html

{% **extends 'base.html'** %}  
{% **block content** %}  
 <**div class="jumbotron"**>  
 <**h5 class="display-2"**>Welcome {{ **email** }}</**h5**>  
 <**p class="lead"**>Lorem Ipsum is simply dummy text of the printing and typesetting industry</**p**>  
 </**div**>  
{% **endblock content** %}

\*\*\* welcome.html will look like this after signing in



Create signUp.html in templates and populate it with HTML code

Command line

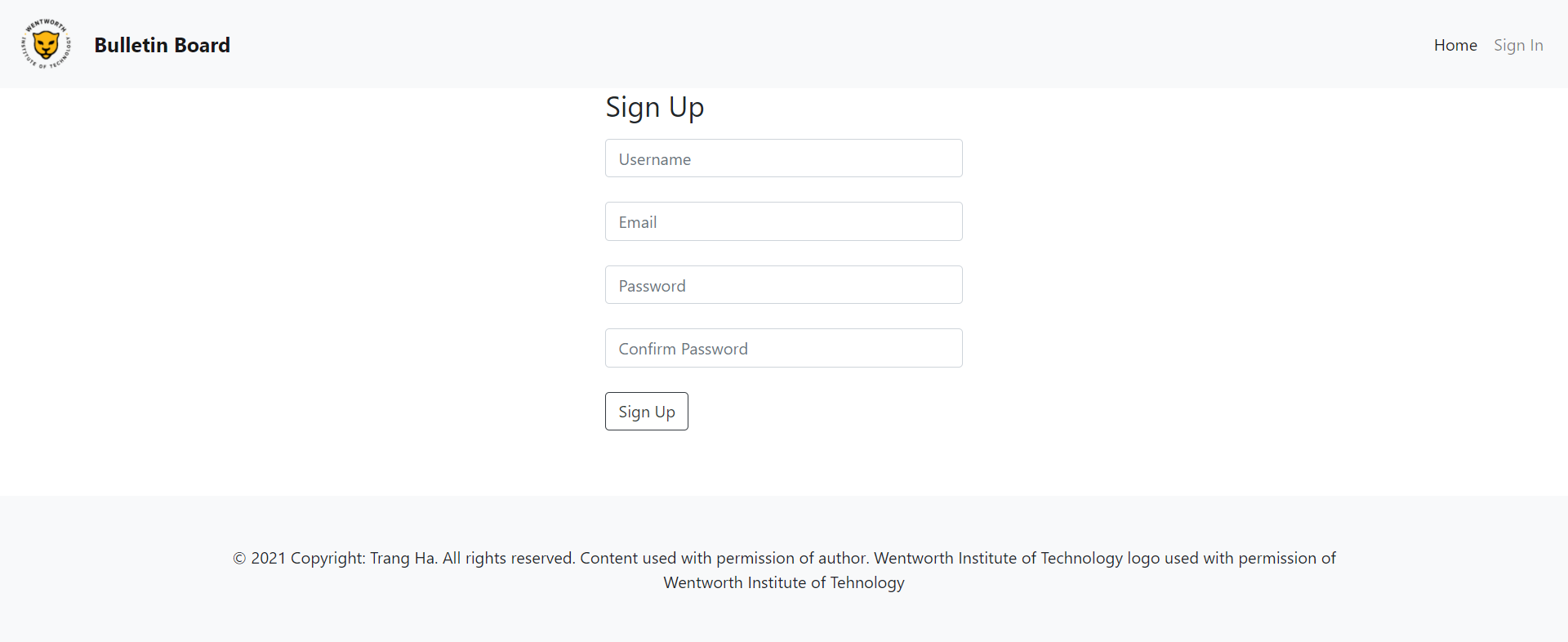
$ nano templates/signUp.html

Code

# templates/signUp.html

{% **extends 'base.html'** %}  
{% **block navbar** %}  
 *<!-- Navigation Bar -->* <**nav class="navbar navbar-expand-lg navbar-light bg-light static-top"**>  
 *<!-- Logo -->* <**a class="navbar-brand" href="**{% **url 'home'** %}**"**>  
 <**img src="/media/image/WIT-Logo.png" alt="logo" width="60" height="60"**>  
 </**a**>  
 *<!-- Logo Brand -->* <**a class="navbar-brand font-weight-bold" href="**{% **url 'home'** %}**"**>Bulletin Board</**a**>  
 *<!-- Hamburger menu icon -->* <**button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarResponsive" aria-controls="navbarResponsive" aria-expanded="false" aria-label="Toggle navigation"**>  
 <**span class="navbar-toggler-icon"**></**span**>  
 </**button**>  
  
 <**div class="collapse navbar-collapse" id="navbarResponsive"**>  
 <**ul class="navbar-nav ml-auto"**>  
 *<!-- Home -->* <**li class="nav-item active"**>  
 <**a class="nav-link" href="**{% **url 'home'** %}**"**>Home</**a**>  
 </**li**>  
 *<!-- User -->* <**li class="nav-item"**>  
 <**a class="nav-link" href="**{% **url 'signin'** %}**"**>Sign In</**a**>  
 </**li**>  
 </**ul**>  
 </**div**>  
 </**nav**>  
{% **endblock navbar** %}  
{% **block content** %}  
 {% **if message** %}  
 <**div class="alert alert-warning" role="alert"**><**p class="text-center"**>{{ **message** }}</**p**></**div**>  
 {% **endif** %}  
 <**div class="row"**>  
 <**div class="col-md-4 offset-md-4"**>  
 <**form class="form-signup" action="/postsignUp/" method="post"**>  
 {% **csrf\_token** %}  
 <**h1 class="h3 mb-3 font-weight-normal"**>Sign Up</**h1**>  
 *<!-- Enter username -->* <**input type="username" id="Username" name="username" class="form-control"  
 placeholder="Username"**><**br**>  
 *<!-- Enter email -->* <**input type="email" id="Email" name="email" class="form-control" placeholder="Email"**><**br**>  
 *<!-- Enter password -->* <**input type="password" id="Password" name="pass" class="form-control" placeholder="Password"**><**br**>  
 *<!-- Repeat password -->* <**input type="password" id="confirm\_password" name="pass-repeat" class="form-control" placeholder="Confirm Password"**><**br**>  
 <**button class="btn btn-outline-dark" type="submit" onclick="**location.href=**'**{% **url 'signup'** %}**"**>Sign Up</**button**>  
 </**form**>  
 </**div**>  
 </**div**>  
{% **endblock content** %}

\*\*\* signUp.html will look like this



Create reset.html in templates and populate it with HTML code

Command line

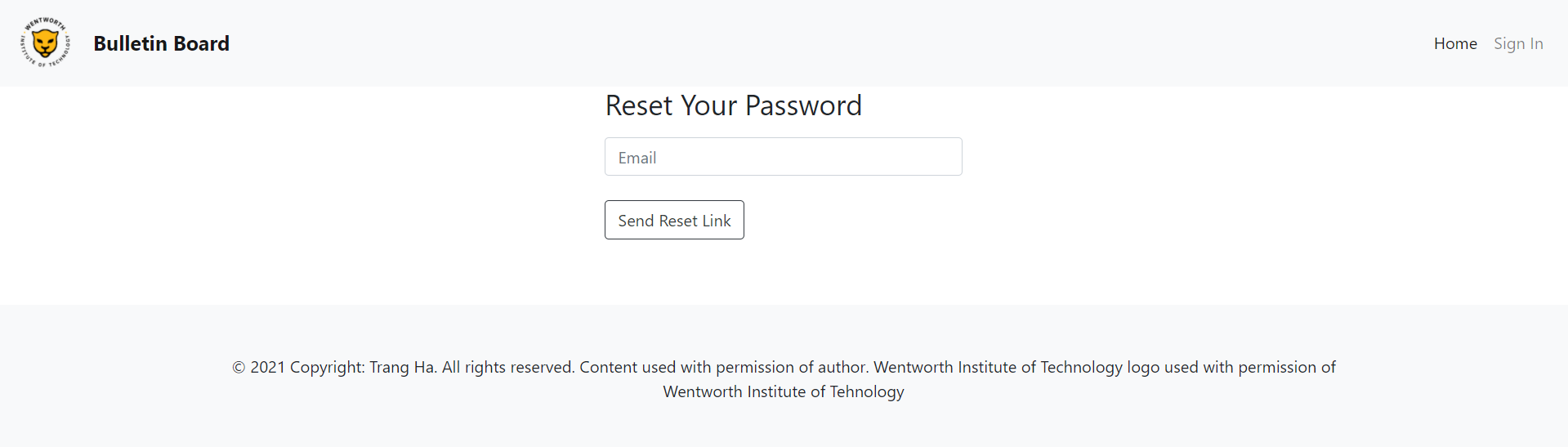
$ nano templates/reset.html

Code

# templates/signIn.html

{% **extends 'base.html'** %}  
{% **block navbar** %}  
 *<!-- Navigation Bar -->* <**nav class="navbar navbar-expand-lg navbar-light bg-light static-top"**>  
 *<!-- Logo -->* <**a class="navbar-brand" href="**{% **url 'home'** %}**"**>  
 <**img src="/media/image/WIT-Logo.png" alt="logo" width="60" height="60"**>  
 </**a**>  
 *<!-- Logo Brand -->* <**a class="navbar-brand font-weight-bold" href="**{% **url 'home'** %}**"**>Bulletin Board</**a**>  
 *<!-- Hamburger menu icon -->* <**button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarResponsive" aria-controls="navbarResponsive" aria-expanded="false" aria-label="Toggle navigation"**>  
 <**span class="navbar-toggler-icon"**></**span**>  
 </**button**>  
 <**div class="collapse navbar-collapse" id="navbarResponsive"**>  
 <**ul class="navbar-nav ml-auto"**>  
 *<!-- Home -->* <**li class="nav-item active"**>  
 <**a class="nav-link" href="**{% **url 'home'** %}**"**>Home</**a**>  
 </**li**>  
 *<!-- User -->* <**li class="nav-item"**>  
 <**a class="nav-link" href="**{% **url 'signin'** %}**"**>Sign In</**a**>  
 </**li**>  
 </**ul**>  
 </**div**>  
 </**nav**>  
{% **endblock navbar** %}  
{% **block content** %}  
 {% **if message** %}  
 <**div class="alert alert-warning" role="alert"**><**p class="text-center"**>{{ **message** }}</**p**></**div**>  
 {% **endif** %}  
 <**div class="row"**>  
 <**div class="col-md-4 offset-md-4"**>  
 <**form class="form-signup" action="/postReset/" method="post"**>  
 {% **csrf\_token** %}  
 <**h1 class="h3 mb-3 font-weight-normal"**>Reset Your Password</**h1**>  
 *<!-- Enter email -->* <**input type="email" id="Email" name="email" class="form-control" placeholder="Email"**><**br**>  
 <**button class="btn btn-outline-dark" type="submit" onclick="*location***.**href**=**'**{% **url 'reset'** %}**'"**>Send Reset Link</**button**>  
 </**form**>  
 </**div**>  
 </**div**>  
{% **endblock content** %}

\*\*\* reset.html will look like this



Update link in navbar in base.html and index.html

Code

# templates/base.html

…

*<!-- User -->*<**li class="nav-item"**>  
 <**a class="nav-link" href="**{% **url 'signout'** %}**"**>Sign Out</**a**>  
</**li**>

# templates/index.html

{% **block navbar** %}

…  
 *<!-- User -->* <**li class="nav-item"**>  
 <**a class="nav-link" href="**{% **url 'signin'** %}**"**>Sign In</**a**>  
 </**li**>

*…*  
{% **endblock navbar** %}  
  
{% **block content** %}

…  
 *<!-- Buttons -->*

<**button class="btn btn-outline-dark" type="button" onclick="*location***.**href**=**'**{% **url 'signup'** %}**'"**>Sign Up</**button**>

…

{% **endblock content** %}

# **VI. Create MVC for File Management**

So far, we have the user authentication set up, and we are going to build an application called "Document" that allows users to upload, delete and download files. The file is upload by HTML form and stored in Firebase Storage. Firebase Realtime Database will then contain the file URL so that the file can be retrieved and accessed by users directly on the website.

## Views

Update views.py

Code

# bulletinBoard/views.py

**from** django.shortcuts **import** render, redirect  
**from** django.contrib **import** auth  
**import** pyrebase  
**import** datetime #new  
**import** re #new  
…  
**def** document(request):  
 get\_time = database.child(**'archive'**).child(**'documents'**).shallow().get().val()  
 **try**:  
 docId = []  
 **for** i **in** get\_time:  
 docId.append(i)  
 date = []  
 title = []  
 url = []  
 **for** i **in** docId:  
 get\_date = database.child(**"archive"**).child(**"documents"**).child(i).child(**"date"**).get().val()  
 get\_title = database.child(**"archive"**).child(**"documents"**).child(i).child(**"title"**).get().val()  
 get\_url = database.child(**"archive"**).child(**"documents"**).child(i).child(**"url"**).get().val()  
 title.append(get\_title)  
 url.append(get\_url)  
 date.append(get\_date)  
 documentLst = [(title[i], date[i], docId[i], url[i]) **for** i **in** range(0, len(docId))]  
 **return** render(request, **'document.html'**, {**"documentLst"**: documentLst})  
 **except**:  
 **return** render(request, **"document.html"**)  
  
**def** upload(request):  
 **return** render(request, **"upload.html"**)  
  
**def** postUpload(request):  
 current\_time = datetime.datetime.now()  
 timestamp = current\_time.timestamp()  
 date = **"{0}-{1}-{2}"**.format(current\_time.month, current\_time.day, current\_time.year)  
 title = request.POST.get(**'title'**)  
 url = request.POST.get(**'url'**)  
 **try**:  
 data = {**"title"**: title, **"date"**: date, **"url"**: url}  
 database.child(**'archive'**).child(**'documents'**).child(str(int(timestamp))).set(data)  
 **return** redirect(document)  
 **except**:  
 message = **"Unable to upload file"  
 return** render(request, **"upload.html"**, {**"message"**: message})  
  
**def** delete(request):  
 docId = request.GET.get(**'docId'**)  
 get\_url = database.child(**"archive"**).child(**'documents'**).child(docId).child(**"url"**).get().val()  
 **try**:  
 *# Delele data on Firebase database* database.child(**"archive"**).child(**'documents'**).child(docId).remove()  
 extractFilename = re.search(**'/o/(.\*?)\?alt'**, get\_url)  
 storage.delete(extractFilename.group(1))  
 **return** redirect(document)  
 **except**:  
 message = **"Unable to delete file"  
 return** render(request, **'document.html'**, {**"message"**: message})

## URLs

Update urls.py

Code

# bulletinBoard/urls.py  
…  
urlpatterns = [  
 …path(**'doc/'**, views.document, name=**"doc"**),  
 path(**'upload/'**, views.upload, name=**"upload"**),  
 path(**'postUpload/'**, views.postUpload),  
 path(**'delete/'**, views.delete, name=**"delete"**),  
] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

## Templates

Create document.html in templates and populate it with HTML code

Command line

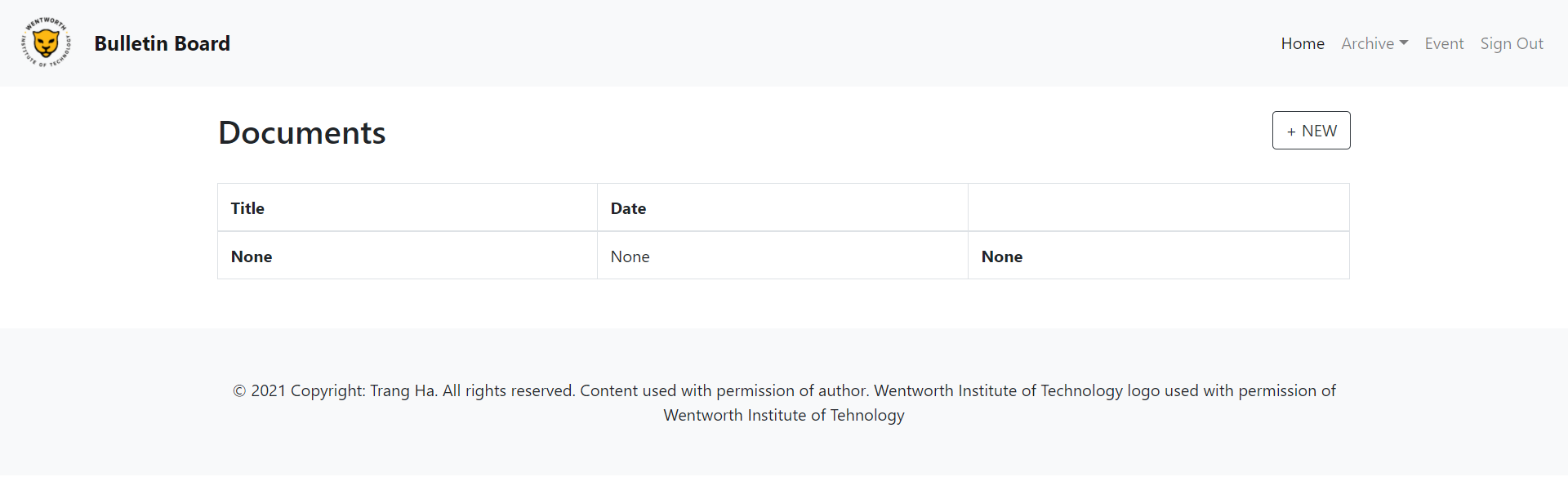
$ nano templates/document.html

Code

# templates/document.html

{% **extends 'base.html'** %}  
{% **block content** %}  
 {% **if message** %}  
 <**div class="alert alert-warning" role="alert"**><**p class="text-center"**>{{ **message** }}</**p**></**div**>  
 {% **endif** %}  
 <**br**>  
 <**button type="button" class="btn btn-outline-dark float-right" onclick="*location***.**href**={% **url 'upload'** %}**"**>+ NEW</**button**>  
 <**h2**>Documents</**h2**>  
 <**br**>  
 <**table class="table table-hover table-bordered"**>  
 <**thead**>  
 <**tr**>  
 <**th**>Title</**th**>  
 <**th**>Date</**th**>  
 <**th**></**th**>  
 </**tr**>  
 </**thead**>  
 <**tbody**>  
 {% **if documentLst** %}  
 {% **for title**, **date**, **docId**, **url in documentLst** %}  
 <**tr**>  
 <**th scope="row"**><**a href="**{{ **url** }}**" class="text-muted"**>{{ **title** }}</**a**></**th**>  
 <**td**>{{ **date** }}</**td**>  
 <**th**>  
 <**a href="/delete/?docId=**{{ **docId** }}**" class="text-muted"**>Delete</**a**>  
 </**th**>  
 </**tr**>  
 {% **endfor** %}  
 {% **else** %}  
 <**tr**>  
 <**th scope="row"**>None</**th**>  
 <**td**>None</**td**>  
 <**th**>None</**th**>  
 </**tr**>  
 {% **endif** %}  
 </**tbody**>  
 </**table**>  
{% **endblock content** %}

\*\*\* document.html will look like this



Create upload.html in templates and populate it with HTML code

Command line

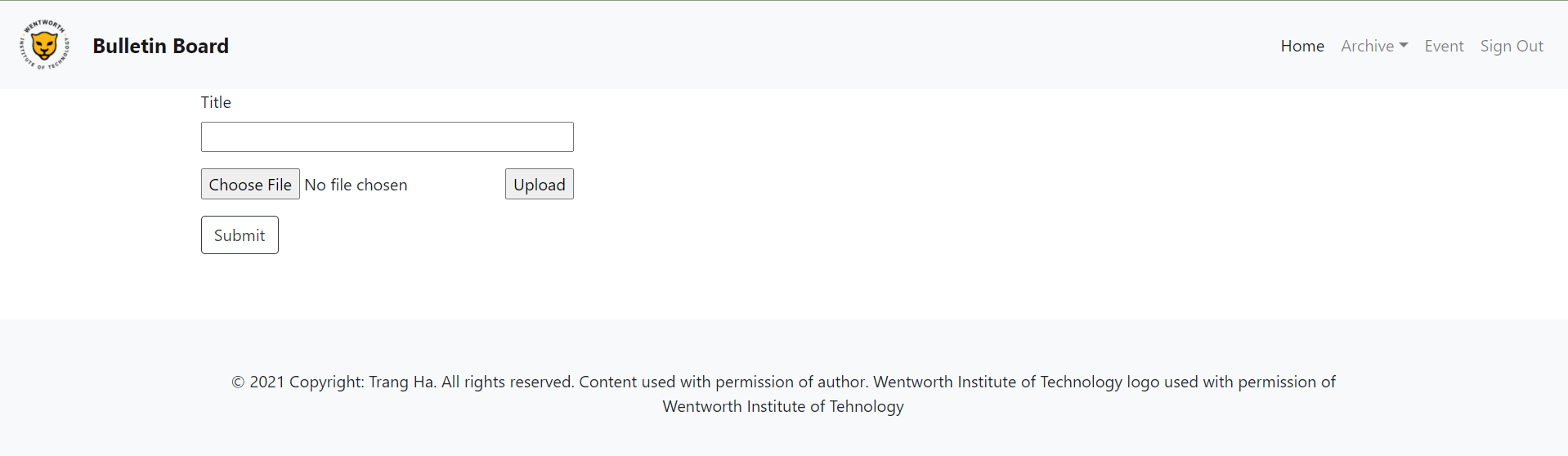
$ nano templates/upload.html

Code

# templates/upload.html

{% **extends 'base.html'** %}  
{% **block content** %}  
 {% **if message** %}  
 <**div class="alert alert-warning" role="alert"**><**p class="text-center"**>{{ **message** }}</**p**></**div**>  
 {% **endif** %}  
 <**div class="row "**>  
 <**div class=".col-md-6 .offset-md-3"**>  
 <**form action="/postUpload/" method="post" enctype="multipart/form-data"**>  
 {% **csrf\_token** %}  
 <**div class="form-group"**>  
 <**label for="formGroupExampleInput"**>Title</**label**>  
 <**input type="text" name="title" class="form-control-file" required**>  
 </**div**>  
 <**div class="form-group"**>  
 <**input type="file" name="files[]" id="files" required**>  
 <**input type="hidden" name="url" id="url"**>  
 <**input type="button" onclick="***uploadFile*()**" value="Upload"**>  
 </**div**>  
 <**input type="submit" class="btn btn-outline-dark" value="Submit"**>  
 </**form**>  
 </**div**>  
 </**div**>  
 <**script src="https://www.gstatic.com/firebasejs/3.7.4/firebase.js"**></**script**>  
 <**script**>  
 **var *config*** = {  
 **apiKey**: **"<YOUR-API-KEY>"**,  
 **authDomain**: **"<YOUR-AUTH-DOMAIN>"**,  
 **databaseURL**: **"<YOUR-DATABASE-URL>"**,  
 **storageBucket**: **"<YOUR-STORAGE-BUCKET>"**,  
 **measurementId**: **"<YOUR-MEASUREMENTID>"**,  
 };  
 firebase.initializeApp(***config***);  
 *// Get a reference to the storage service, which is used to create references in your storage bucket* **function** *uploadFile*(){  
 **var** storage = firebase.**storage**();  
 **var** file = ***document***.getElementById(**"files"**).**files**[0];  
 **var** storageRef = storage.ref();  
 **var** uploadFile = storageRef.**child**(file.**name**).put(file);  
 uploadFile.on(**'state\_changed'**,**function**(snapshot){  
 ***console***.log(**"file upload succesfully"**);  
 },  
 **function**(error) {  
 },  
 **function**() {  
 *// Upload completed successfully, now we can get the download URL* **var** downloadURL = uploadFile.snapshot.downloadURL;  
 ***document***.getElementById(**"url"**).**value** = downloadURL;  
 *alert*(**"File uploaded successfully"**);  
 });  
 }  
 </**script**>  
{% **endblock content** %}

\*\*\* upload.html will look like this



Update link in navbar in base.html

Code

# templates/base.html

{% **block navbar** %}

… *<!-- Archive (dropdown) -->* <**li class="nav-item dropdown"**>  
 <**a class="nav-link dropdown-toggle" id="navbarDropdown" data-toggle="dropdown" aria-expanded="false"**>Archive</**a**>  
 <**div class="dropdown-menu"**>  
 <**a class="dropdown-item" href="**{% **url 'doc'** %}**"**>Document</**a**>

…

{% **endblock navbar** %}

# **VII. Create MVC for Post**

In the previous session, we have learned how to push data to Firebase Realtime Database by HTML forms. For this session, we will create an application called "Post" that allows users to post any announcements with similar steps in the previous session to practice and be more familiar with Firebase Realtime Database.

## Views

Update views.py

Code

# bulletinBoard/views.py

…**def** post(request):  
 get\_postId = database.child(**"archive"**).child(**"posts"**).shallow().get().val()  
 **try**:  
 postId = []  
 **for** i **in** get\_postId:  
 postId.append(i)  
 postId.sort(reverse=**True**)  
 date = []  
 title = []  
 content = []  
 **for** i **in** postId:  
 get\_date = database.child(**"archive"**).child(**"posts"**).child(i).child(**"date"**).get().val()  
 get\_title = database.child(**"archive"**).child(**"posts"**).child(i).child(**"title"**).get().val()  
 get\_content = database.child(**"archive"**).child(**"posts"**).child(i).child(**"content"**).get().val()  
 date.append(get\_date)  
 title.append(get\_title)  
 content.append(get\_content)  
 postLst = [(title[i], content[i], date[i], postId[i]) **for** i **in** range(0, len(postId))]  
 **return** render(request, **"post.html"**, {**"postLst"**: postLst})  
 **except**:  
 **return** render(request, **"post.html"**,)  
  
**def** add(request):  
 **return** render(request, **"add.html"**)  
  
**def** postAdd(request):  
 current\_time = datetime.datetime.now()  
 timestamp = current\_time.timestamp()  
 date = current\_time.strftime(**"%m/%d/%Y %H:%M:%S"**)  
 title = request.POST.get(**'title'**)  
 content = request.POST.get(**'content'**)  
 **try**:  
 data = {**"title"**: title, **"content"**: content, **"date"**: date}  
 database.child(**"archive"**).child(**"posts"**).child(str(int(timestamp))).set(data)  
 **return** redirect(post)  
 **except**:  
 message = **" Unable to add post"  
 return** render(request, **"add.html"**, {**"message"**: message})  
  
**def** remove(request):  
 postId = request.GET.get(**'postId'**)  
 **try**:  
 *# Delele data on Firebase database* database.child(**"archive"**).child(**"posts"**).child(postId).remove()  
 **return** redirect(post)  
 **except**:  
 message = **"Unable to delete post"  
 return** render(request, **'post.html'**, {**"message"**: message})

## URLs

Update urls.py

Code

# bulletinBoard/urls.py  
…  
urlpatterns = [

…path(**'post/'**, views.post, name=**"post"**),  
 path(**'add/'**, views.add, name=**"add"**),  
 path(**'postAdd/'**, views.postAdd),  
 path(**'remove/'**, views.remove, name=**"remove"**),  
] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

## Templates

Create post.html in templates and populate it with HTML code

Command line

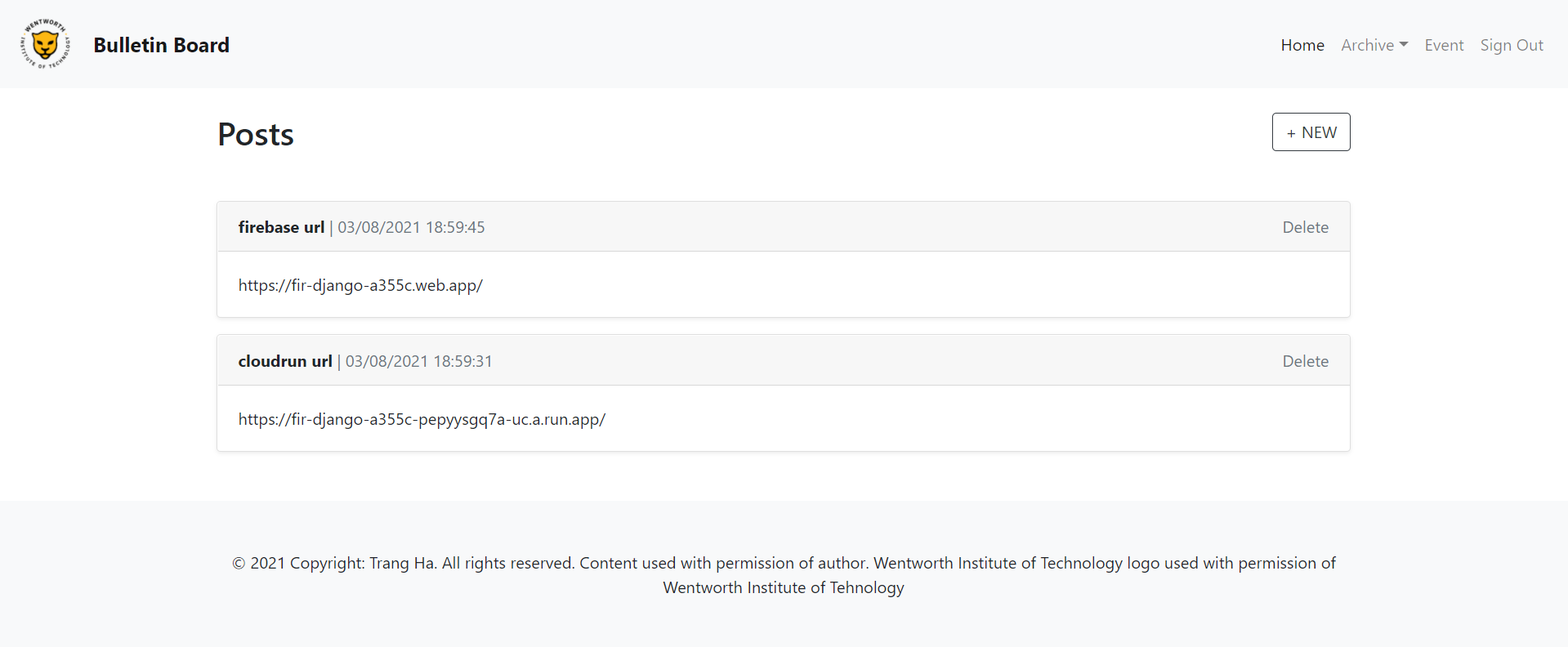
$ nano templates/post.html

Code

# templates/post.html

{% **extends 'base.html'** %}  
{% **block content** %}  
 {% **if message** %}  
 <**div class="alert alert-warning" role="alert"**><**p class="text-center"**>{{ **message** }}</**p**></**div**>  
 {% **endif** %}  
 <**br**>  
 <**button type="button" class="btn btn-outline-dark float-right" onclick="*location***.href={% **url 'add'** %}**"**>+  
 NEW</**button**>  
 <**h2**>Posts</**h2**>  
 <**br**>  
 {% **if postLst** %}  
 {% **for title**, **question**, **date**, **postId in postLst** %}  
 <**div class="card shadow-sm mt-3"**>  
 <**div class="card-header"**>  
 <**span class="font-weight-bold"**>{{ **title** }}</**span**><**span class="text-muted"**> | {{ **date** }}</**span**>  
 <**span class="float-right"**>  
 <**a href="/remove/?postId=**{{ **postId** }}**" class="text-muted"**>Delete</**a**>  
 </**span**>  
 </**div**>  
 <**div class="card-body"**>  
 {{ **content** }}  
 </**div**>  
 </**div**>  
 {% **endfor** %}  
 {% **else** %}  
 <**div class="card shadow-sm mt-3"**>  
 <**div class="card-header"**>  
 <**span class="font-weight-bold"**>None</**span**>  
 </**div**>  
 <**div class="card-body"**>None</**div**>  
 </**div**>  
 {% **endif** %}  
{% **endblock content** %}

\*\*\* post.html will look like this



Create add.html in templates and populate it with HTML code

Command line

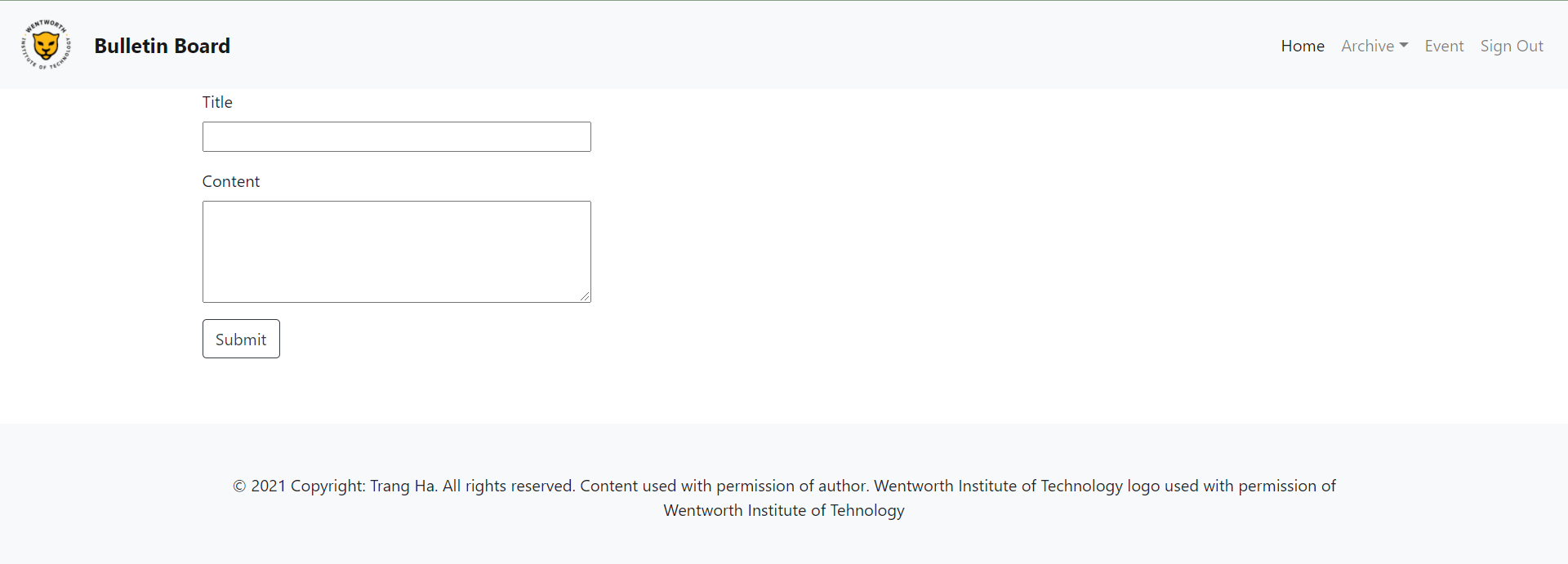
$ nano templates/add.html

Code

# templates/add.html

{% **extends 'base.html'** %}  
{% **block content** %}  
 {% **if message** %}  
 <**div class="alert alert-warning" role="alert"**><**p class="text-center"**>{{ **message** }}</**p**></**div**>  
 {% **endif** %}  
 <**div class="row "**>  
 <**div class=".col-md-6 .offset-md-3"**>  
 <**form action="/postAdd/" method="post"**>  
 {% **csrf\_token** %}  
 <**div class="form-group"**>  
 <**label**>Title</**label**><**br**>  
 <**input type="text" name="title" class="form-control-file" required**>  
 </**div**>  
 <**div class="form-group"**>  
 <**label**>Content</**label**><**br**>  
 <**textarea rows="4" cols="50" name="content" required**></**textarea**>  
 </**div**>  
 <**input type="submit" class="btn btn-outline-dark" value="Submit"**>  
 </**form**>  
 </**div**>  
 </**div**>  
{% **endblock content** %}

\*\*\* add.html will look like this



Update link in navbar in base.html

Code

# templates/base.html

{% **block navbar** %}

…  
 *<!-- Archive (dropdown) -->* <**li class="nav-item dropdown"**>  
 <**a class="nav-link dropdown-toggle" id="navbarDropdown" data-toggle="dropdown" aria-expanded="false"**>Archive</**a**>  
 <**div class="dropdown-menu"**>  
 <**a class="dropdown-item" href="**{% **url 'post'** %}**"**>Post</**a**>

…  
{% **endblock navbar** %}

# **VIII. Create MVC for Event**

We will build an application called “Event” that allows users to create a schedule with the same technique in the previous section.

## Views

Update views.py

Code

# bulletinBoard/views.py  
  
*…***def** agenda(request):  
 get\_date = database.child(**"events"**).shallow().get().val()  
 **try**:  
 eventId = []  
 **for** i **in** get\_date:  
 eventId.append(i)  
 eventId.sort(reverse=**True**)  
 event = []  
 date = []  
 start\_time = []  
 end\_time = []  
 content = []  
 **for** i **in** eventId:  
 get\_event = database.child(**"events"**).child(i).child(**"event"**).get().val()  
 get\_date = database.child(**"events"**).child(i).child(**"date"**).get().val()  
 get\_start\_time = database.child(**"events"**).child(i).child(**"start\_time"**).get().val()  
 get\_end\_time = database.child(**"events"**).child(i).child(**"end\_time"**).get().val()  
 get\_content = database.child(**"events"**).child(i).child(**"content"**).get().val()  
 event.append(get\_event)  
 date.append(get\_date)  
 start\_time.append(get\_start\_time)  
 end\_time.append(get\_end\_time)  
 content.append(get\_content)

eventLst = [(event[i], date[i], start\_time[i], end\_time[i], content[i], eventId[i]) **for** i **in** range(0, len(eventId))]

**return** render(request, **'agenda.html'**, {**"eventLst"**: eventLst})  
 **except**:  
 **return** render(request, **"agenda.html"**)  
  
**def** create(request):  
 **return** render(request, **"create.html"**)  
  
**def** postCreate(request):  
 current\_time = datetime.datetime.now()  
 timestamp = current\_time.timestamp()  
 event = request.POST.get(**'event'**)  
 content = request.POST.get(**'content'**)  
 start\_time = request.POST.get(**'start\_time'**)  
 end\_time = request.POST.get(**'end\_time'**)  
 date = request.POST.get(**'date'**)  
 **try**:  
 data = {**"date"**: date,**"event"**: event,**"start\_time"**: start\_time, **"end\_time"**: end\_time, **"content"**: content}  
 database.child(**"events"**).child(str(int(timestamp))).set(data)  
 **return** redirect(agenda)  
 **except**:  
 message = **" Unable to create event"  
 return** render(request, **"create.html"**, {**"message"**: message})  
  
**def** cancel(request):  
 eventId = request.GET.get(**'eventId'**)  
 **try**:  
 *# Delele data on Firebase database* database.child(**"events"**).child(eventId).remove()  
 **return** redirect(agenda)  
 **except**:  
 message = **"Unable to delete event"  
 return** render(request, **'agenda.html'**, {**"message"**: message})

## URLs

Update urls.py

Code

# bulletinBoard/urls.py

urlpatterns = [  
 …path(**'event/'**, views.agenda, name=**"event"**),  
 path(**'create/'**, views.create, name=**"create"**),  
 path(**'postCreate/'**, views.postCreate),  
 path(**'cancel/'**, views.cancel, name=**"cancel"**),  
] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

## Templates

Create agenda.html in templates and populate it with HTML code

Command line

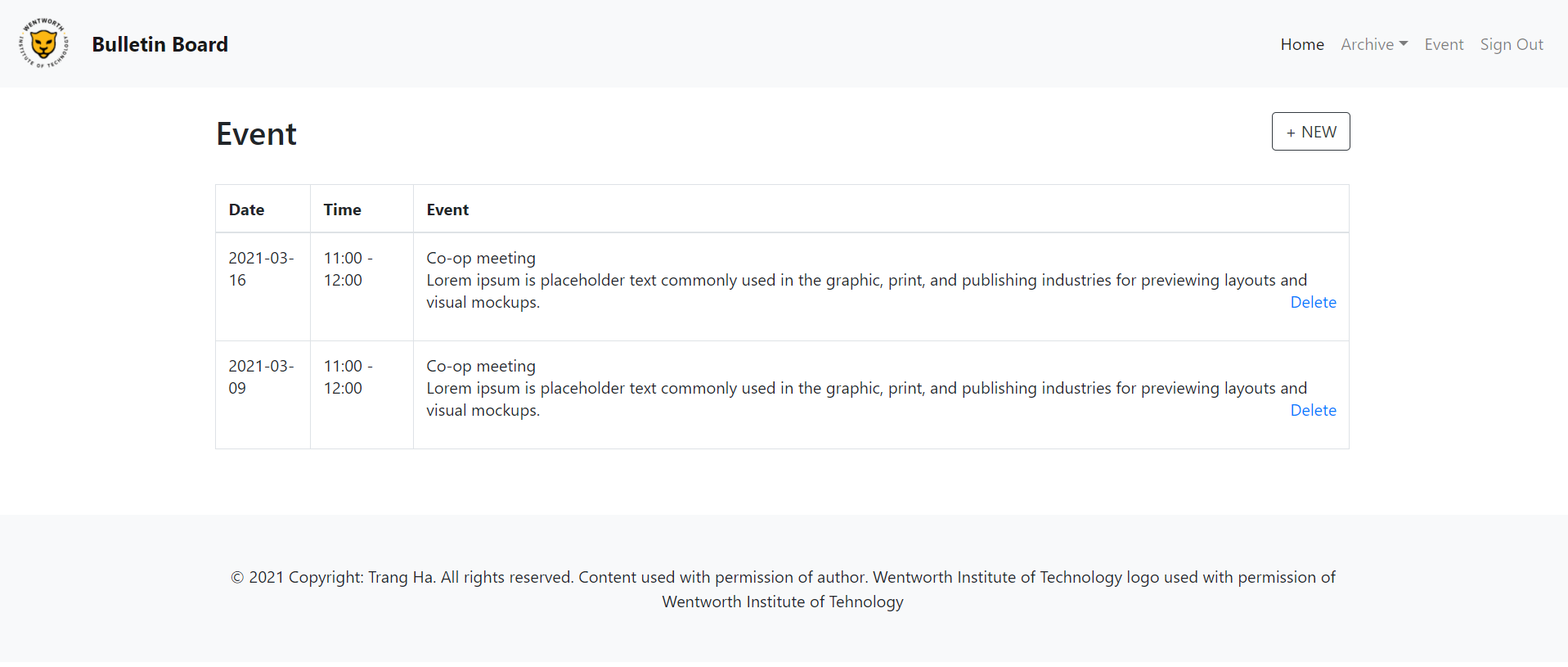
$ nano templates/agenda.html

Code

# templates/agenda.html

{% **extends 'base.html'** %}  
{% **block content** %}  
 {% **if message** %}  
 <**div class="alert alert-warning" role="alert"**><**p class="text-center"**>{{ **message** }}</**p**></**div**>  
 {% **endif** %}  
 <**br**>  
 <**button type="button" class="btn btn-outline-dark float-right" onclick="*location***.**href**={% **url 'create'** %}**"**>+  
 NEW</**button**>  
 <**h2**>Event</**h2**>  
 <**br**>  
 <**div class="agenda"**>  
 <**div class="table-responsive"**>  
 <**table class="table table-condensed table-bordered"**>  
 <**thead**>  
 <**tr**>  
 <**th**>Date</**th**>  
 <**th**>Time</**th**>  
 <**th**>Event</**th**>  
 </**tr**>  
 </**thead**>  
 <**tbody**>  
 *<!-- Single event in a single day -->* {% **if eventLst** %}  
 {% **for event**, **date**, **start\_time**, **end\_time**, **content**, **eventId in eventLst** %}  
 <**tr**>  
 <**td**>{{ **date** }}</**td**>  
 <**td**>{{ **start\_time** }} - {{ **end\_time** }}</**td**>  
 <**td**>  
 <**p**>{{ **event** }}  
 <**br**>{{ **content** }}  
 <**a href="/cancel/?eventId=**{{ **eventId** }}**" class="float-right"**>Delete</**a**>  
 </**p**>  
 </**td**>  
 </**tr**>  
 {% **endfor** %}  
 {% **else** %}  
 <**tr**>  
 <**td**>None</**td**>  
 <**td**>None</**td**>  
 <**td**>None</**td**>  
 </**tr**>  
 {% **endif** %}  
 </**tbody**>  
 </**table**>  
 </**div**>  
 </**div**>  
{% **endblock content** %}

\*\*\* agenda.html will look like this



Create create.html in templates and populate it with HTML code

Command line

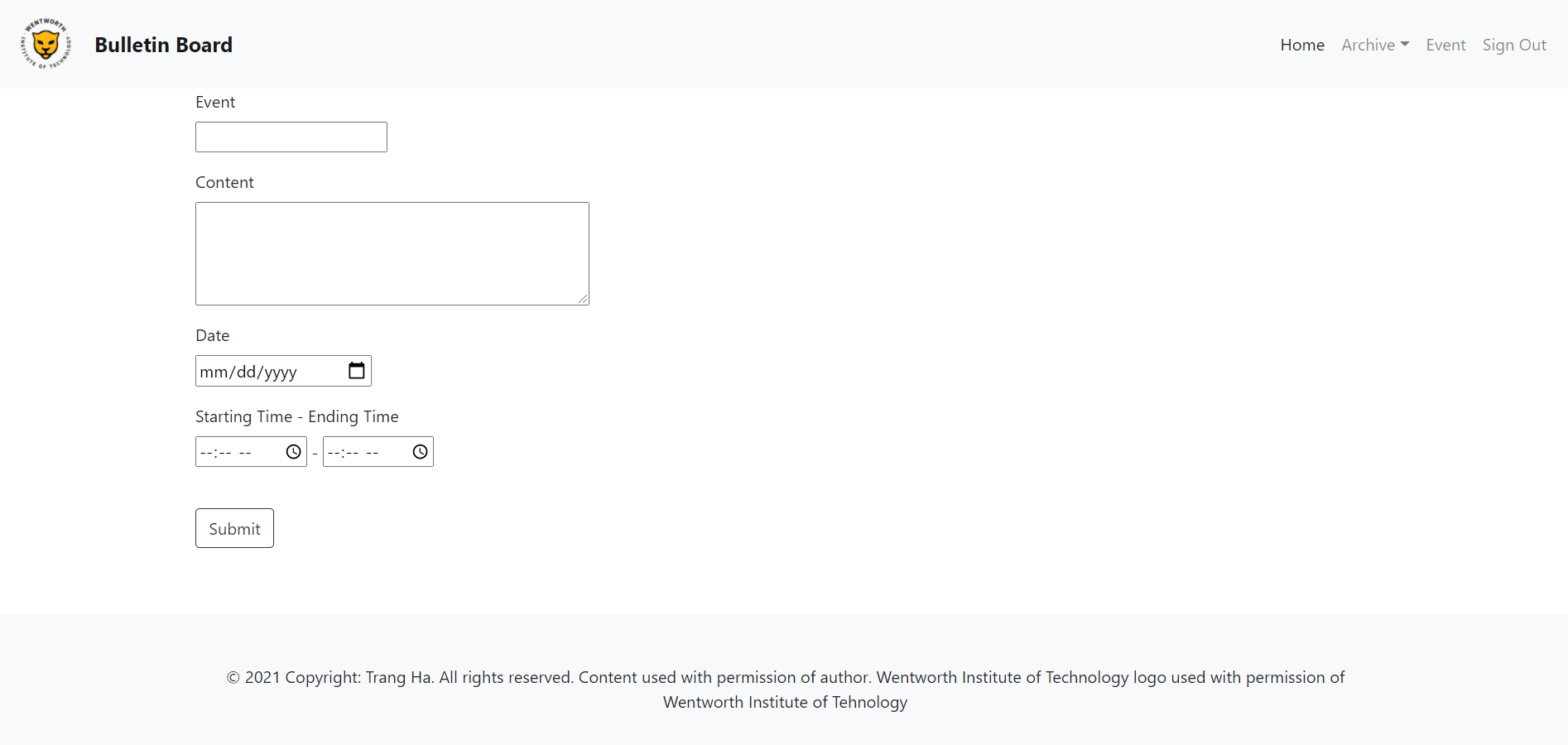
$ nano templates/create.html

Code

# templates/create.html

{% **extends 'base.html'** %}  
{% **block content** %}  
 {% **if message** %}  
 <**div class="alert alert-warning" role="alert"**><**p class="text-center"**>{{ **message** }}</**p**></**div**>  
 {% **endif** %}  
 <**div class="row "**>  
 <**div class=".col-md-6 .offset-md-3"**>  
 <**form action="/postCreate/" method="post"**>  
 {% **csrf\_token** %}  
 <**div class="form-group"**>  
 <**label**>Event</**label**><**br**>  
 <**input type="text" name="event" required**>  
 </**div**>  
 <**div class="form-group"**>  
 <**label**>Content</**label**><**br**>  
 <**textarea rows="4" cols="50" name="content" required**></**textarea**>  
 </**div**>  
 <**div class="form-group"**>  
 <**label**>Date</**label**><**br**>  
 <**input type="date" name="date" required**>  
 </**div**>  
 <**div class="form-group"**>  
 <**label**>Starting Time - Ending Time</**label**><**br**>  
 <**input type="time" name="start\_time" required**> - <**input type="time" name="end\_time" required**>  
 </**div**><**br**>  
 <**input type="submit" class="btn btn-outline-dark" value="Submit"**>  
 </**form**>  
 </**div**>  
 </**div**>  
{% **endblock content** %}

\*\*\* create.html will look like this



Update link in navbar in base.html

Code

# templates/base.html

{% **block navbar** %}

…  
*<!-- Event -->*<**li class="nav-item"**>  
 <**a class="nav-link" href="**{% **url 'event'** %}**"**>Event</**a**>  
</**li**>

…  
{% **endblock navbar** %}

# **IX. Deploy to Cloud Run**

## Initial Setup for Cloud Run

Step 1: Follow the steps in the following link to install Cloud SDK: <https://cloud.google.com/sdk/docs/quickstart#deb>

Step 2: Go to <https://console.cloud.google.com/> > Login to Google Account > Select organization > Choose the project that you create in Firebase

Machine generated alternative text:
Google Cloud Platform 
Dashboard 
Select a project 
Q 
Search products and resources 
Select from NEWBURY.EDU 
To view this page, select a project. 
Q Search projects and folders 
ID 
844736755468 
fir-django-a581 a 
NEW PROJECT 
CANCEL 
CREATE PROJECT 
RECENT 
Name 
ALL 
newbury.edu 
firebase-django 
OPEN 

Step 3: Enable billing by going to “Billing” on the navigation bar on the left

Machine generated alternative text:
0 
RPI 
fre 
Google Cloud Platform 
Home 
Marketplace 
Billing 
APIs & Services 
Support 
IAM & Admin 
Getting started 
Security 
Compliance 
Anthos 
firebase-django 
MMENDATIONS 
COVID-19. Learn more 
project-fir-django-a581a&supported... 
Search products and resources 
RPI APIs 
Requests (requests/sec) 
7:15 
Requests: 0.033/s 
Go to APIs overview 
CUSTOMIZE 
DISMISS 
7:30 
008/s 
0.04/s 
002Js 
7:45 
COMPUTE 
App Engine 
Compute Engine 
Kubernetes Engine 
@ Google Cloud Platform status 
All services normal 
Go to Cloud status dashboard 
Monitoring 
Set up alerting policies 
Create uptime checks 
View all dashboards 
Go to Monitoring 
o 
Error Reporting 
No sign of any errors. Have you set up Error Reporting? 

\*\*\* If you already set up a billing account, click on "Link a billing account". Otherwise, click on "Manage billing accounts"

\*\*\* You will need a credit card to access this. Don't worry, Google won't charge you anything, you will have $300 credit for free to spend over the next 90 days. After that, you still can access Google Cloud Platform with limited usage.

Step 4: Go to Cloud Build on the navigation bar on the left > Enable

Step 5: Go to Cloud Run on the navigation bar on the left > Create Service

## Create a service account

Step 1: Go to IAM & Admin > Service accounts > Create service account > Fill in service account detail

* Enter “Service account name”
* Add a description is optional
* Click “Create”

Step 2: Grant this service account access to the project

Machine generated alternative text:
Google Cloud Platform 
IAM & Admin 
Create service account 
firebase-django v 
Grant this service account access to project 
(optional) 
Grant this service account access to firebase-django so that it has permission to 
complete specific actions on the resources in your project. Learn more 
Role 
Cloud Run Admin 
Full control over all Cloud Run resources. 
Role 
Cloud Run Service Agent 
Gives Cloud Run service account access to 
managed resources. 
Role 
Cloud Build Service Agent 
Gives Cloud Build service account access to 
managed resources. 
Role 
Viewer 
Read access to all resources. 
+ ADD ANOTHER ROLE 
CONTINUE 
Condition 
Add condition 
Condition 
Add condition 
Condition 
Add condition 
Condition 
Add condition Assign roles for the service account: Cloud Run Admin, Cloud Run Service Agent, Cloud Build Service Agent, Cloud Build Service Agent > Click “Continue” > Skip “Grant users access to this service account” > Click Done

\*\*\* You will see a new account listed in the list

Step 3: Configure the project and account

* Set the configuration to default by entering 1
* Choose the default Gmail account
* Pick the project you are working on

Command line

Machine generated alternative text:
(firebase_django-MsN3TLoV) 
Welcome! This command will 
Settings from your current 
accessibility: 
screen reader: 'False' 
core : 
gcloud init 
take you through the configuration of gcloud . 
configuration [default] are: 
account: hat-4øø6@fir-django-4øø6.iam.gserviceaccount . com 
disable_usage_reporting: 'True ' 
project: fir-django-4ØØ6 
Pick configuration to use: 
[1] Re-initialize this configuration [default] with new settings 
[2] Create a new configuration 
Please enter your numeric choice: 
1 
Your current configuration has been set to: [default] 
You can skip diagnostics next time by using the following flag: 
gcloud init 
-skip-diagnostics 
Network diagnostic detects and fixes local network connection issues. 
Checking network connection.. .done. 
Reachability Check passed. 
Network diagnostic passed (1/1 checks passed) . 
Choose the account you would like to use to perform operations for 
this configuration : 
[1] hat-4øø6@fir-django-4øø6.iam.gserviceaccount . com 
[2] hat@wit.edu 
[3] trangha1999-github@fir-django-a355c . iam.gserviceaccount . com 
[4] Log in with a new account 
Please enter your numeric choice: 
2 
You are logged in as: Chat@wit . edu] . 
Pick cloud project to use: 
[1] fir-django-4ØØ6 
[2] fir-django-a355c 
[3] Create a new project 
Please enter numeric choice or text value (must exactly match list 
item) : 
1 
Your current project has been set to: [fir-django-4øø6] . $ gcloud init

Step 4: Download the service account key

Command line

$ gcloud iam service-accounts keys create ./<NAME-OF-KEY-FILE>.json --iam-account <SERVICE-ACCOUNT-EMAIL>

\*\*\* A new <NAME-OF-KEY-FILE>.json will add to the project-level folder.

Step 5: Authenticate the service account so that you can deploy the project with service account

Command line

$ gcloud auth activate-service-account --key-file=<NAME-OF-KEY-FILE>.json

\*\*\* The service account successfully authenticated if you see the output said: Activated service account credentials for: [<SERVICE-ACCOUNT-EMAIL>]

Step 6: Switch to the service account

Command line

$ gcloud config set account <SERVICE-ACCOUNT-EMAIL>

\*\*\* To display active service account: gcloud auth list

\*\*\* To deactivate or revoke service account: gcloud auth revoke <SERVICE-ACCOUNT-EMAIL>

## Requirement.txt and Dockerfile

Requirements.txt is a text file at project-level folder specifying all the program’s dependencies needed for the project.

Command line

$ pip freeze > requirements.txt

Dockerfile is a text file without an extension needed. It contains specifications and commands to assemble a Docker Image. Syntax of a Dockerfile:

* The FROM command specifies the base image that the code will run on python 3.8
* The RUN command runs the pip command to install all packages listed in requirements.txt
* The COPY command copies the source code into a folder in the container
* The WORKDIR command sets the container source code folder as the working directory.
* The ENV PORT 8080 command sets a variable for the port.
* The ENV PYTHONUNBUFFERED command sets an environmental variable that ensures output from python is sent straight to the terminal without buffering it first.
* The CMD command binds the port and runs the project after the container is launched from the built image.

Command line

$ nano Dockerfile

Code

*# Dockerfile*

*# Use an official lightweight Python image.  
# https://hub.docker.com/\_/python***FROM** python:3.8**-**slim  
*# Install dependencies.***COPY** .**/**requirements.txt .  
**RUN** pip install **-**r requirements.txt  
*# Copy local code to the container image.***COPY** . **/**app

**WORKDIR /**app  
*# Service must listen to $PORT environment variable.  
# This default value facilitates local development.***ENV *PORT*** 8080  
*# Setting this ensures print statements and log messages  
# promptly appear in Cloud Logging.***ENV *PYTHONUNBUFFERED*** TRUE  
  
*# Run the web service on container startup. Here we use the gunicorn  
# webserver, with one worker process and 8 threads.  
# For environments with multiple CPU cores, increase the number of workers  
# to be equal to the cores available.***CMD** exec gunicorn **--**bind 0.0.0.0:$***PORT* --**workers 1 **--**threads 8 **--**timeout 0 bulletinBoard.wsgi:application

## Deploy to Cloud Run

Step 1: Build container with Cloud Build

* <NAME-OF-CONTAINER> can be named anything. For this project, just name it as the <PROJECT-ID>

Command line

$ gcloud builds submit --tag gcr.io/<PROJECT-ID>/<NAME-OF-CONTAINER>

\*\*\* Once you see a success message, you are ready to deploy to Cloud Run

Machine generated alternative text:
ID 
2c695cba 
1 more) 
STATUS 
-baaø-4d2ø 
SUCCESS 
-8557 
-6aa3bde4123c 
CREATE TIME 
2021-03 -Ø2TØ2. 
.54:48+00 
DURATION 
48S 
SOURCE 
gs : / /fir-dj ango-4ØØ6_c10udbui1d/source/1614653687.732303 -7ca78c553cfb49558876bd7acad38472. tgz 
IMAGES 
gcr. io/fir-django-4øø6/fir-django-4øø6 (+ 

Step 2: Deploy the container to Cloud Run

* Choose Cloud Run (full managed)
* Press enter to confirm the service name in the parenthesis
* Choose the region (us-central1)
* Say yes to question about enable APIs and allowing unauthenticated access

Command line

$ gcloud beta run deploy --image gcr.io/<PROJECT-ID>/<NAME-OF-CONTAINER> --platform managed

\*\*\* After the deploy completes, you should see a service URL. Go to the that link, and you see the website is now running on server instead of local webserver.

\*\*\* If you see DisallowedHost error, go to the settings.py and add service URL to ALLOWED\_HOSTS = [“<SERVICE-URL>”]. For example,

. Then commit the change and push to GitHub again.

# **X. Deploy to Firebase Hosting**

So far, you have the website runs on Cloud Run, but our goal is to have the website hosts on Firebase Hosting by hooking up Cloud Run to Firebase Hosting. This section will guide you on how to deploy the website on Firebase Hosting.

## Initial Setup for Firebase

Step 1: Follow the steps in the following link to install npm: <https://linuxconfig.org/install-npm-on-linux>

Step 2: Install Firebase CLI

Command line

$ npm init -y

$ npm i -- D firebase-tools

\*\*\* package.json, package-lock.json and the folder node\_modules will add to the project-level folder

Step 3: Login to Firebase

Command line

$ firebase login

Step 3: Initialize Firebase Hosting

* Choose hosting by pressing space and enter
* Choose existing project > Choose the project you are working on
* Direct the public directory to static/css
* Say no to configure as a single-page app

Command line

$ node\_modules/.bin/firebase init hosting

\*\*\* 404.html and index.html will automatically create in static/css. You don’t need those files, so just delete it.

## Deploy to Firebase Hosting

Step 1: Edit firebase.json in project-level folder to hook up Cloud Run to Firebase Hosting

* "public" tells Firebase Hosting the static part of the website.
* "ignore" tells Firebase Hosting to ignore any file that starts with the dot, firebase.json, and node\_modules folder.
* "rewrite" navigates the path for the dynamic parts of the website to Cloud Run.
* "source" matches everything except if it's a static file.
* "run" contains "serviceId" and "region."
  + "serviceId" is the project Id on Google Cloud Platform and Firebase.
  + "region" is the location of the project hosting.

Code

# firebase.json

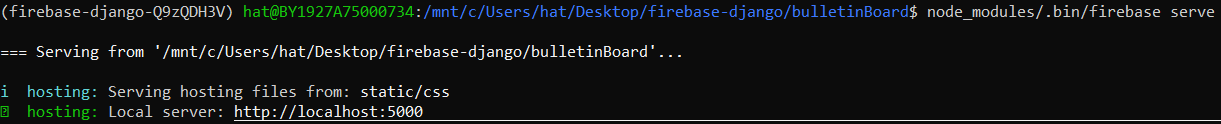
{  
 **"hosting"**: {  
 **"public"**: **"static/css"**,  
 **"ignore"**: [  
 **"firebase.json"**,  
 **"\*\*/.\*"**,  
 **"\*\*/node\_modules/\*\*"** ],  
 **"rewrites"**: [{  
 **"source"**: **"\*\*"**,  
 **"run"**: {  
 **"serviceId"**: **"<PROJECT-ID>"**,  
 **"region"**: **"us-central1"** }  
 }]  
 }  
}

Step 2: Start the deployment in local server

Command line

$ node\_modules/.bin/firebase serve

\*\*\* The project will host on local server. Click on <http://localhost:5000> to see the project



Step 3: Deploy to Firebase Hosting

Command line

$ node\_module/.bin/firebase deploy --only hosting

\*\*\* The project is now hosting on Firebase, if you see this output with a hosting url.

Machine generated alternative text:
(firebase-django-Q9zQDH3V) hat@BY1927A75ØØØ734:/mnt/c/Users/hat/Desktop/firebase-django/bu11etin80ard$ node_modules/ .bin/firebase deploy 
-only hosting 
1 
1 
1 
1 
1 
Deploying to 'fir-django-a355c' 
deploying hosting 
hostingCfir-django 
hostingCfir-django 
hosting [fir-django 
hostingCfir-django 
hosting [fir-django 
hostingCfir-django 
hosting [fir-django 
Deploy complete ! 
-a355c] : 
-a355c] : 
-a355c] : 
-a355c] : 
-a355c] : 
-a355c] : 
-a355c] : 
beginning deploy.. 
found 1 files in static/css 
file upload complete 
finalizing version... 
version finalized 
releasing new version... 
release complete 

# **XI. GitHub Actions**

## We have learn the manually way to deploy to Firebase Hosting and Cloud Run. However, it is an inefficient way to deploy the website since we have to redo every step, and errors might occur whenever we change the code. To automate deployment, we setup GitHub Actions for Cloud Build and Firebase Hosting to connect to the GitHub repository and create a trigger to an event from GitHub, such as push to a branch, push new tag, or pull request. For the tutorial, every time we push the code to the branch master, Cloud Build will follow the instructions we defined in the deploy-cloudrun.yml file to build a container with Cloud Build and then deploy to Cloud Run.

## Setup GitHub Actions to Deploy to Firebase Hosting

Step 1: Create a GitHub repository. For this project, name the repository firebase-django

Step 2: In the project-level folder, set up Firebase Hosting

* Confirm yes to automatic builds and deploys with GitHub
* Enter the project repository on GitHub (<GITHUB-USERNAME>/firebase-django)
* Confirm yes to setup the workflow to run a build script before every deploy
* Enter “npm ci” for the script to deploy before every deploy
* Confirm yes to setup automatic deployment

Command line

$ firebase init hosting:github

\*\*\* The command will create a folder .github/workflow containing two yaml files which take care of setting up GitHub Action

Step 3: Push and commit the project on GitHub repository

Command line

$ git init

$ git status

$ git add .

$ git commit -m "<COMMENT>"

$ git branch -M master

$ git remote add origin <REPOSITORY-HTTP>

$ git push -u origin master

## Setup GitHub Actions to Deploy to Cloud Run

Step 1: Create a file at .github/workflows/deploy-cloudrun.yml. The instructions that Cloud Run follows:

* "on" is to configure the event that is to push the code to GitHub repository on branch master.
* "env" is to set variables for secrets, such as the project Id, project region, and secrete keys for the service account in Google Cloud Platform.
* "jobs" is the set of commands to build and push the image to Container Registry and deploy it to Cloud Run.

Command line

$ nano .github/workflows/deploy-cloudrun.yml

Code

# .github/workflows/deploy-cloudrun.yml

**name**:Deploy to Cloud Run

**on**:

**push**:

**branches**:

- master

**env**:

**PROJECT\_ID**:${{secrets.GCLOUD\_PROJECT}}

**RUN\_REGION**: us-central1

**SA\_KEY\_JSON**:${{secrets.GCLOUD\_AUTH}}

**jobs**:

**deploy**:

**name**: Deploy to Cloud Run

**runs-on**: ubuntu-latest

**steps**:

- **uses**: actions/checkout@v2

*#Setup gcloud CLI*

- **uses**: GoogleCloudPlatform/github-actions/setup-gcloud@master

**with**:

**version**: **"290.0.1"**

**service\_account\_key**: ${{secrets.GCLOUD\_AUTH}}

**project\_id**: ${{secrets.GCLOUD\_PROJECT}}

*# Build and push image to Google Container Registry*

- **name**: Build

**run**: gcloud builds submit --tag gcr.io/$PROJECT\_ID/$PROJECT\_ID:$GITHUB\_SHA

*# Deploy to Cloud Run*

- **name**:Deploy

**run**: gcloud run deploy $PROJECT\_ID --image gcr.io/$PROJECT\_ID/$PROJECT\_ID:$GITHUB\_SHA --platform managed --region $RUN\_REGION

Step 2: Setup app secret in Github

In your Github repository project, click on "Settings" > "Secrets" > New repository secret. Then add these two new secrets:

* GCLOUD\_AUTH: Copy the entire Google Cloud Platform service account key json file and paste in the text area
* GCLOUD\_PROJECT: Your project ID

\*\*\* You should delete the service account key json file on the project-level folder after paste the key into GitHub Secrets so no one could access your service account.

Machine generated alternative text:
GCLOUD AUTH 
GCLOUD PROJECT 
Updated yesterday 
Updated yesterday 
Update 
Update 
Remove 
Remove 

Step 3: Push your project to GitHub master branch and GitHub will trigger an automated deployment.

\*\*\* If you see a green check mark, the project has been successfully deployed on Cloud Run and Firebase Hosting. To see build detail, click on "Actions"

Machine generated alternative text:
trangHa1999 Add firebase hosting url in settings.py 
x/ 6Ø58bbc 9 hours ago 
.github/workflows 
Update firebase.json and deploy-cloudrun.yml 
11 commits 
yesterday 

# **XII. Bonus**

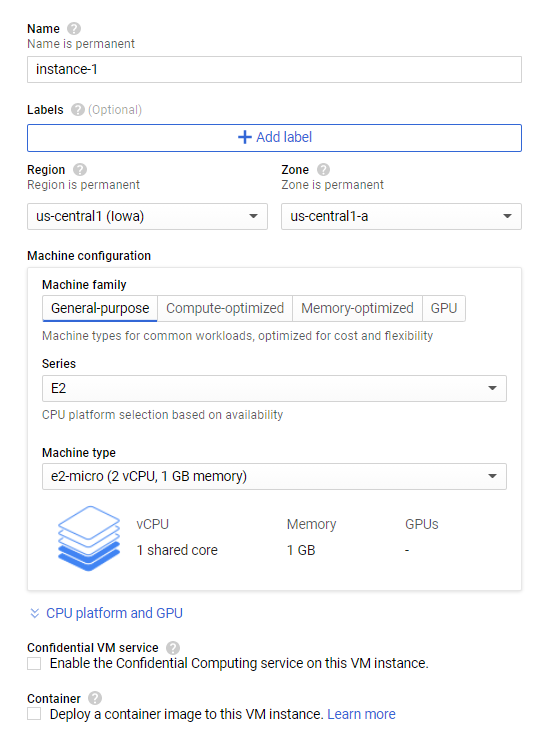
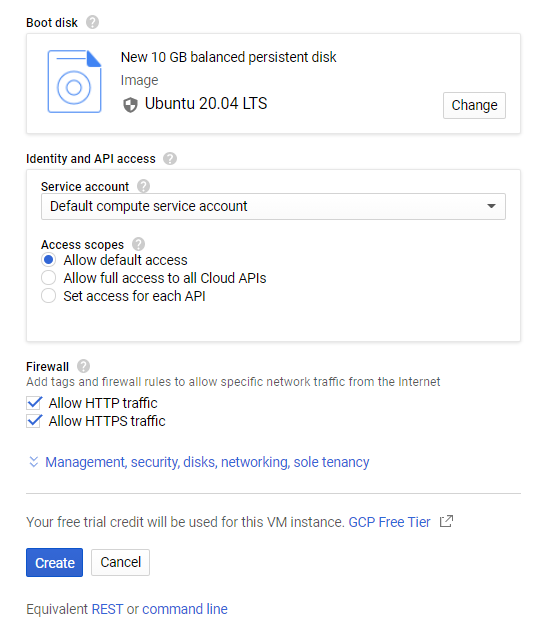
## In addition to deploying a website by integrating Firebase with Google Cloud Run, Google Cloud Platform also provides other ways to support hosting a website, such as deployment on Compute Engine, App Engine, and Kubernetes Engine. This section will guide you on how to do that.

## Deploy to Compute Engine

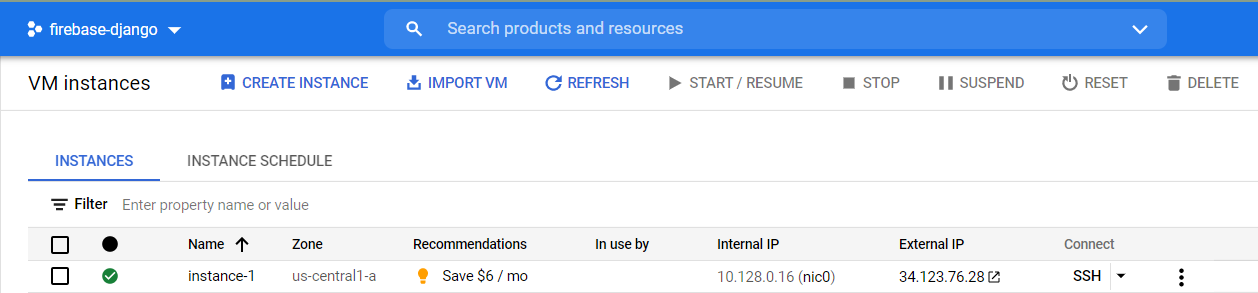
Step 1: Create a virtual machine instance in Google Compute Engine

Go to Compute Engine > Create instance > Configure the machine as following

* Region: us-central1, Zone: us-central1-a
* Machine type: e2-micro(2 vCPU, 1 GB memory)
* Boot Disk:
* Operating system: Ubuntu
* Version: Ubuntu 20.4 LTS
* Firewall: allow HTTP and HTTPS traffics



\*\*\* A new instance will show on VM instances list page.



Step 2: Push source code to GitHub repository and add instance external IP to ALLOWED\_HOSTS. For example,

Step 3: Install Python and Pip for the virtual machine

In the VM instances list page, open the virtual machine terminal by clicking on SSH of instace.

Command line

$ python3 –version

$ sudo apt-get update

$ sudo apt-get install python3-pip

Step 4: Clone GitHub repository to virtual machine

Command line

$ git clone <HTTPS-URL>

Step 5: Install dependencies for the virtual machine

Command line

$ sudo pip3 install -r requirements.txt

Step 6: Start the development server

Command line

$ nohup sudo python3 manage.py runserver 0.0.0.0:80

\*\*\* To vew the website, click on http://<INSTANCE-EXTERNAL-IP-ADDRESS>

## Deploy to App Engine

Step 1: Create a service account and assign these roles: App Engine Admin and App Engine flexible environment Service Agent (Review section IX: Create a Service Account for detail tutorial)

Step 2: Create app.yaml file which contains configuration information for deployment to Google App Engine > Remove Dockerfile so that the two files don’t conflict with each other.

Command line

$ nano app.yaml

Code

# app.yaml

**runtime**: python  
**env**: flex  
**entrypoint**: gunicorn -b :$PORT bulletinBoard.wsgi  
  
**runtime\_config**:  
 **python\_version**: 3

Step 3: Deploy to App Engine

Command line

$ gcloud app deploy

$ gcloud app browse

\*\*\* You should see a link to view your app runs on Google App Engine

\*\*\* If you see DisallowedHost error, go to the settings.py and add service URL to ALLOWED\_HOSTS = [“<URL>”]. For example,

## Deploy to Kubernetes Engine

Step 1: Create Dockerfile (Review section IX: Requirements.txt and Dockerfile for detail tutorial)

Step 2: Build and push container image to Container Registry with Cloud Build

Command line

$ gcloud builds submit --tag gcr.io/<PROJECT-ID>/<NAME-OF-CONTAINER>

Step 3: Install kubectl

Command line

$ gcloud components install kubectl

Step 4: Create the cluster in Google Kubernetes Engine which is a managed set of Compute Engine virtual machines that operate as a single Kubernetes Engine cluster.

Go to Kubernetes Engine > Clusters > Create > Configure standard cluster

* For this project, change the name of cluster to project ID
* Choose us-central1-a for Zone
* Click “Create”

Step 5: Create gke-deploy,yaml to define cluster resources

Command line

$ nano gke-deploy.yaml

Code

# gke-deploy.yaml

**apiVersion**: apps/v1  
**kind**: Deployment  
**metadata**:  
 **name**: <PROJECT-ID>  
 **labels**:  
 **app**: <PROJECT-ID>  
**spec**:  
 **replicas**: 3  
 **selector**:  
 **matchLabels**:  
 **app**: <PROJECT-ID>  
 **template**:  
 **metadata**:  
 **labels**:  
 **app**: <PROJECT-ID>  
 **spec**:  
 **containers**:  
 - **name**: <PROJECT-ID>  
 **image**: gcr.io/<PROJECT-ID>/<PROJECT-ID>:latest  
 **ports**:  
 - **containerPort**: 8080

Step 6: Deploy the resource to the cluster

Command line

$ kubectl apply -f gke-deploy.yaml

Step 7: Track the status of the deployment

Command line

$ kubectl get deployments

\*\*\* The deployment is successful when all the available deployments are ready



Step 8: Create gke-service.yaml

Command line

$ nano gke-service.yaml

Code

# gke-service.yaml

**apiVersion**: v1  
**kind**: Service  
**metadata**:  
 **name**: <PROJECT-ID>  
**spec**:  
 **type**: LoadBalancer  
 **selector**:  
 **app**: <PROJECT-ID>  
 **ports**:  
 - **port**: 80  
 **targetPort**: 8080

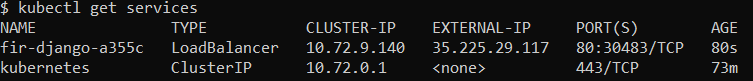
Step 9: Create service and get external IP address

Command line

$ kubectl apply -f gke-service.yaml

$ kubectl get services

$ curl <EXTERNAL-IP>



\*\*\* If you see DisallowedHost error, go to the settings.py and add service URL to ALLOWED\_HOSTS = [“<EXTERNAL-IP>”]. For example, . Click on http://<INSTANCE-EXTERNAL-IP> to review the website.

# **XV. Source**

“Bootstrap 4 Cheat Sheet - The Ultimate List of Bootstrap Classes.” Bootstrap Templates and Themes, hackerthemes.com/bootstrap-cheatsheet/#container.

Craig, Paul. “Quickstart: Continuous Deployment to Google Cloud Run Using Github Actions.” DEV Community, DEV Community, 12 Nov. 2020, dev.to/pcraig3/quickstart-continuous-deployment-to-google-cloud-run-using-github-actions-fna.

“Django Authentication Project with Firebase.” GeeksforGeeks, 15 Jan. 2021, www.geeksforgeeks.org/django-authentication-project-with-firebase/.

East, David. “Hosting Flask Servers on Firebase from Scratch.” Medium, Firebase Developers, 27 Aug. 2019, medium.com/firebase-developers/hosting-flask-servers-on-firebase-from-scratch-c97cfb204579.

“Getting Started with Django.” Google, Google, cloud.google.com/python/django.

Marín, Julio. “10 Firebase Realtime Database Rule Templates.” Medium, Medium, 30 Jan. 2019, medium.com/@juliomacr/10-firebase-realtime-database-rule-templates-d4894a118a98.

Python Advocacy team. “Django on Cloud Run.” Google, Google, codelabs.developers.google.com/codelabs/cloud-run-django#0.