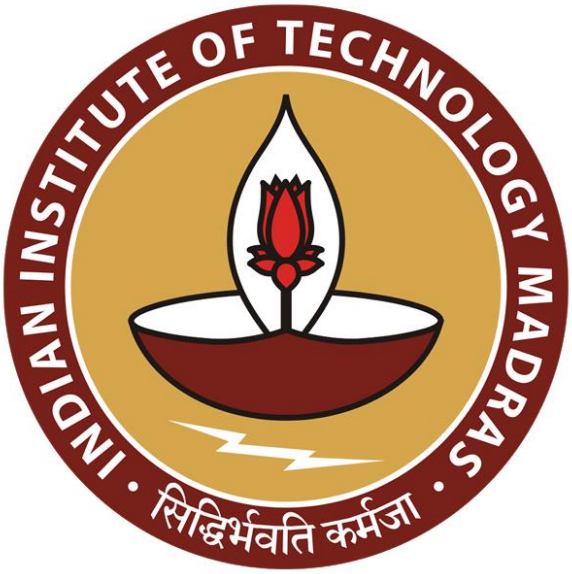
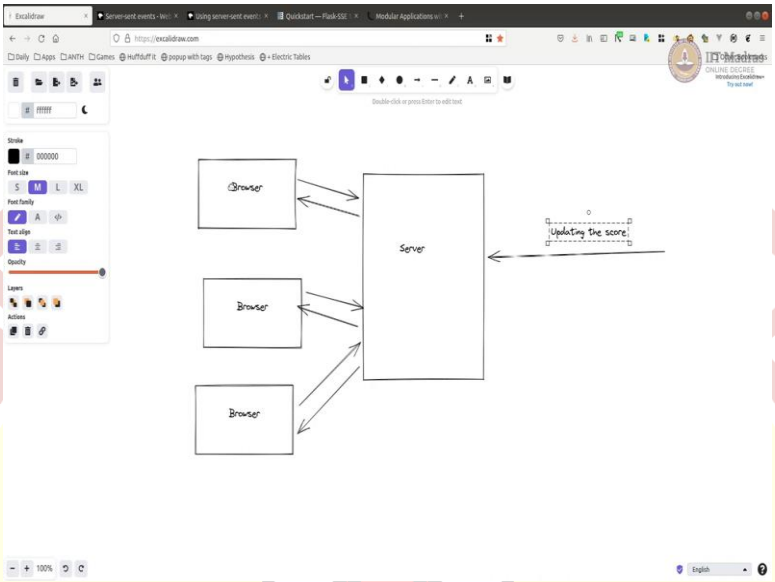


IIT Madras

ONLINE DEGREE

Modern Application Development
Professor. Thejesh G N
Software Consultant
Indian Institute of Technology, Madras
Server Sent Events

(Refer Slide Time: 00:14)



Welcome to the Modern Application Development to Screencast. In this screencast, we will see how to push events from server. For this, you will need a Linux development environment. And we will need editor and you will have to install Redis and Celery. So, we can experiment. Now, let us see why do we need server sent event? Let us assume you are building a cricket score application, there is a match going on, somebody is going to update the score. And then you will have to broadcast that score to multiple users.

Let us make it simple. Assume this was a server. And this is a browser. One of your sports enthusiasts is keeping track of score. There could be more than one. And there is a scope page here, which they are accessing, and then they are getting response from it. Each one is a request response. And let us see this. I assume there are 300 or 500, how many of our users are watching the score.

Now there is somebody updating the scores from the backend, you know. Tomorrow, somebody is the field, updating the match scores as the match goes on. And the same scores gets updated into database or some storage. And then whenever somebody visits the page, it is shown to the user.

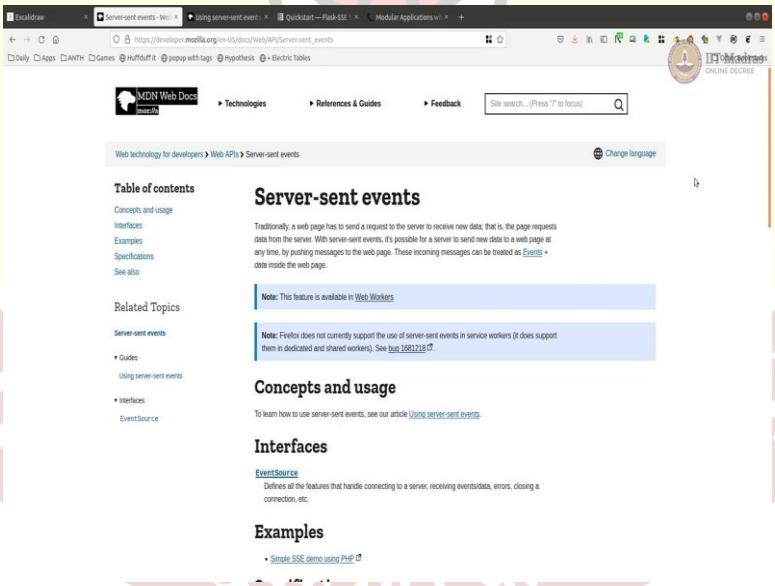
Now, what is the problem here? The match can go at its own speed. Sometimes the score does not change match. Sometimes the score changes, sometimes you add commentary,

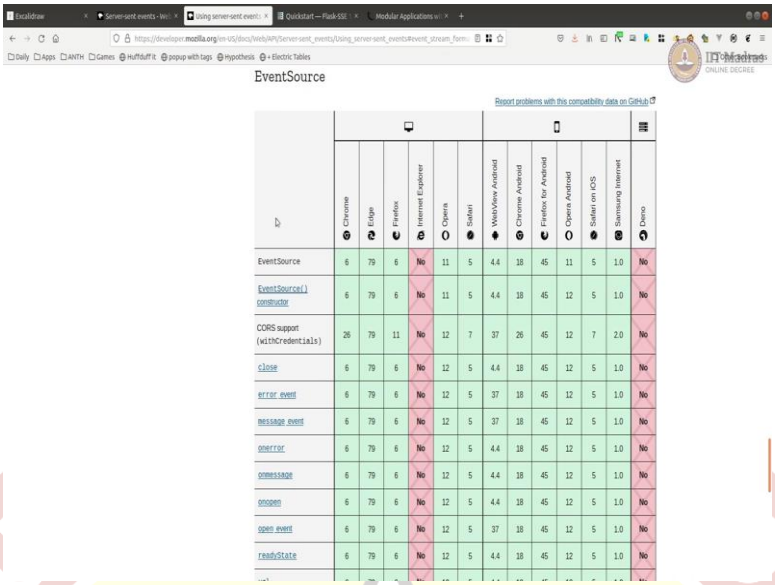
sometimes you do not add commentary. So, for the users to get the latest, he has to keep refreshing the page, Otherwise, you will not get the latest, everything is a request response, you will not get any score if it does not refresh.

So, one of the ways of solving is to make the page auto refresh. You kind of have a some kind of JavaScript timeout, once it times out you refresh maybe every 2 seconds, maybe every 4 seconds or something, you refresh, or you manually click on the refresh button. It is a problem still, because now you are doing refresh for no reason, sometimes refresh might have some data, sometimes it may not.

And sometimes, even before you hit the refresh, data are changed multiple times. So, your user would miss out some update. Now, instead of a browser asking for the score every time, what if it has one time asking it like send me scores, whenever there is an update, and the server is keep sending it, keep sending the data. And that is called Server Sent Events where server keeps sending data to the browser, as long as the browser is interested. Now, this is a standard feature in the browser. Almost all the browsers support it now.

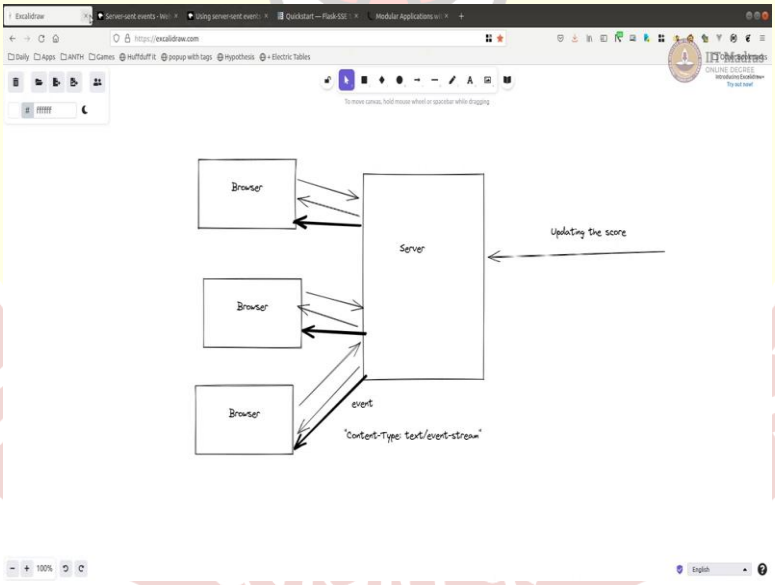
(Refer Slide Time: 03:38)





You can go to MDN web docs page, to see what all the browsers that support this. There is a page, yeah, here. So, almost all except Internet Explorer, and Dino. Dino is like a node like environment. So, it does not matter that much. Internet Explorer does not support. Everyone else actually support. Now, there are two parts to it, like we saw.

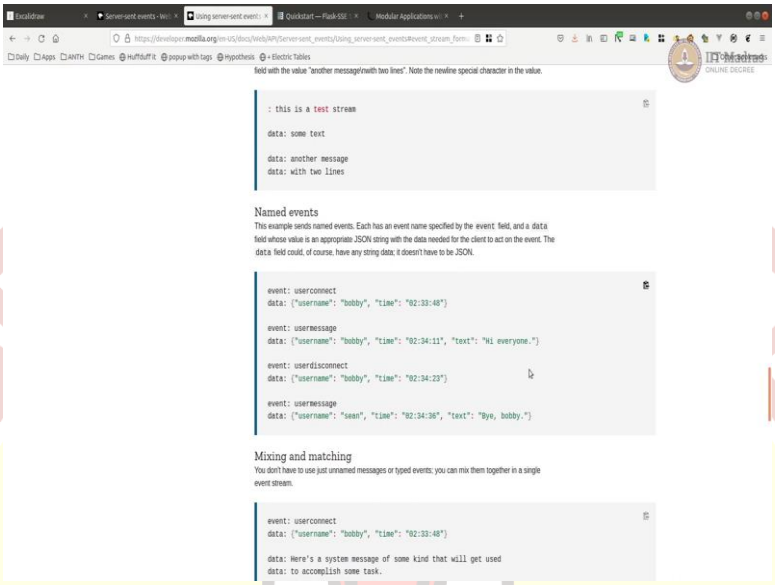
(Refer Slide Time: 04:14)

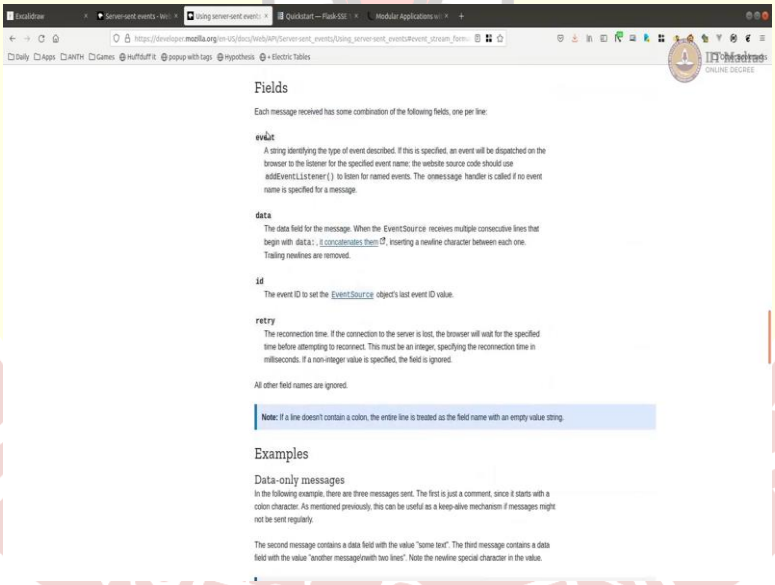


One is how does the server send keep sending the event to the browser? And how does the browser receive it? Now, it uses the same HTTP protocol. So, it is not a new protocol, you extends current HTTP protocol with content type call event slash text event stream text slash event stream. Usually, you will have HTML. So here, if you are actually let us say, we are pushing the data now, this will be a push just marketers, some other colour or bigger pipe. So, this is the event, this the event.

So, this is how content type text slash event stream. And that is how the server sends it, just like in this format, but it is a text that it sent. And then the browser knows how to understand this event stream and calls respect to JavaScript methods, so that you can handle them.

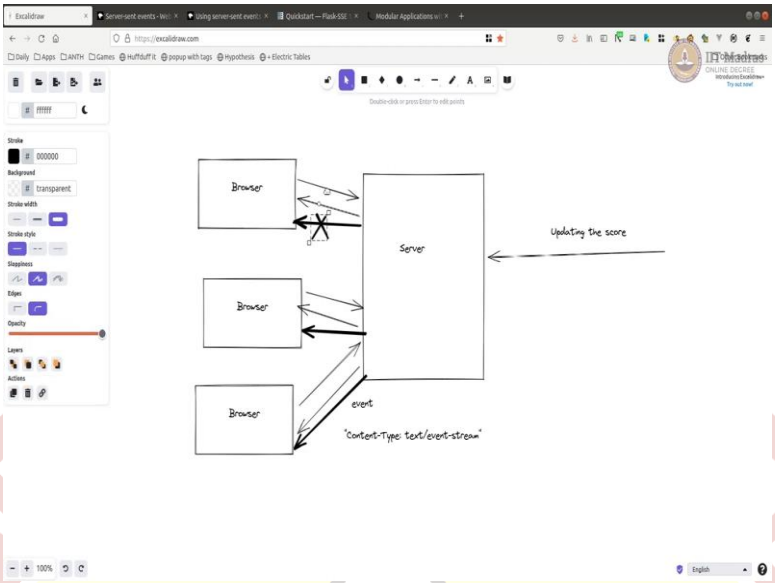
(Refer Slide Time: 05:32)





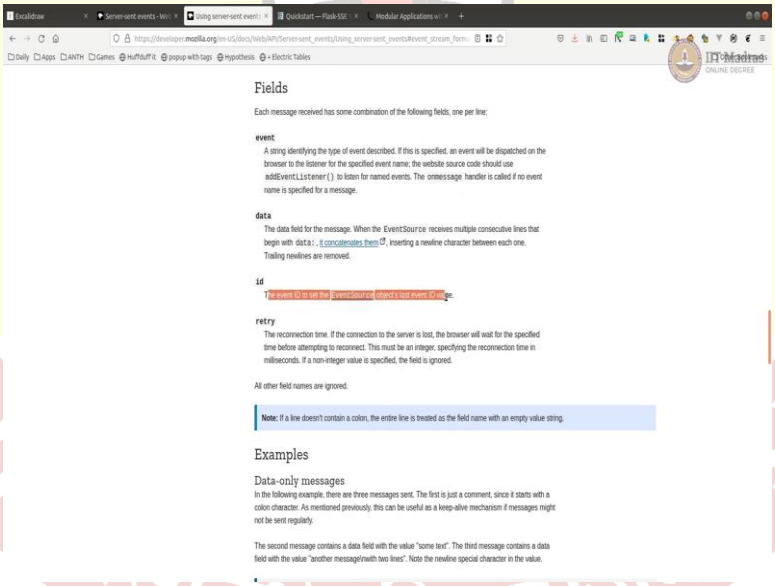
Now, let us see on the server side, how it sent. So, on the server side, it sends like this, there are actually 3 items that it can send. One is event, one is data, and other one is ID, and retry. And I think they can send all 4, event, data, ID, and retry. ID and retry are useful.

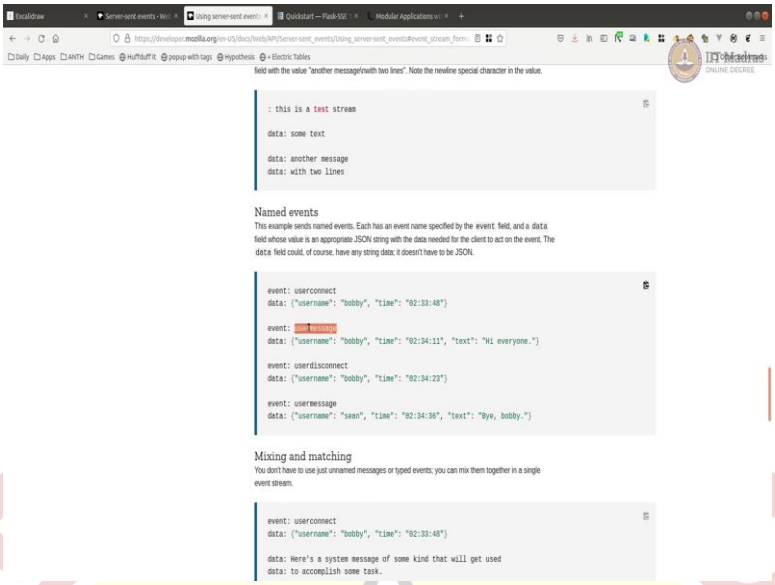
(Refer Slide Time: 06:00)



When actually, for some reason, this becomes disconnected. If this becomes disconnected, then the kind of tells browser how long to wait, before you retry.

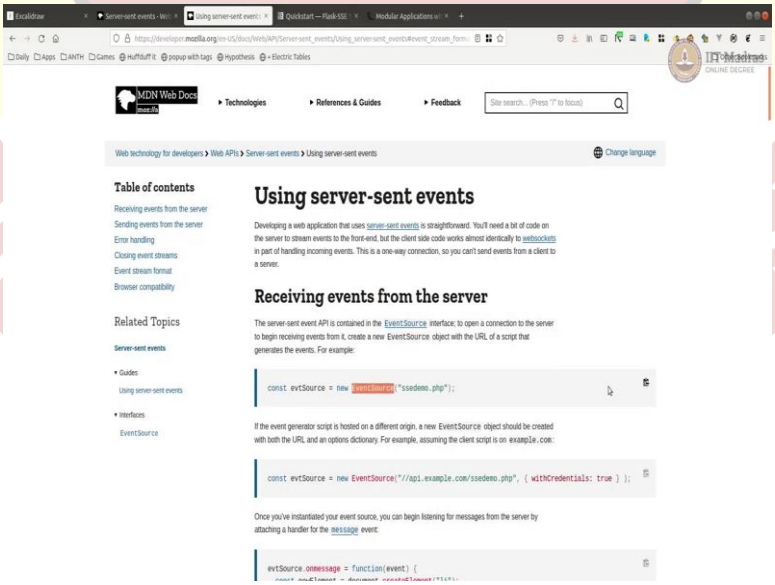
(Refer Slide Time: 06:12)





And what was the last event that sent from where you can pull the next set of events. Event is actually type of the event because you could send many types of data. For example, in this case, we could name the event a score. And the data can have any json text, saying time, for example, commentary, score, etcetera, and your score in the application. And then Id is a unique ID for each of the message. So, if you miss some message, you can pull it from the last thing, last message. Now, that is on the server side, that is how we send but on the client side, how are we going to receive.

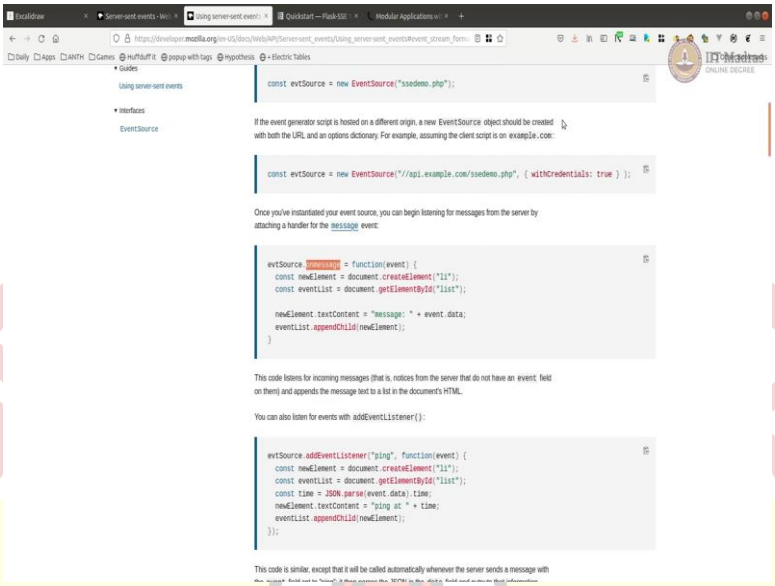
(Refer Slide Time: 06:55)



On the client side, we are going to use this, use something called EventSource. You can go read about this API EventSource it is defined in JavaScript, it gives you a path API, or wherever you implemented your controller endpoint, to the EventSource, and then you can

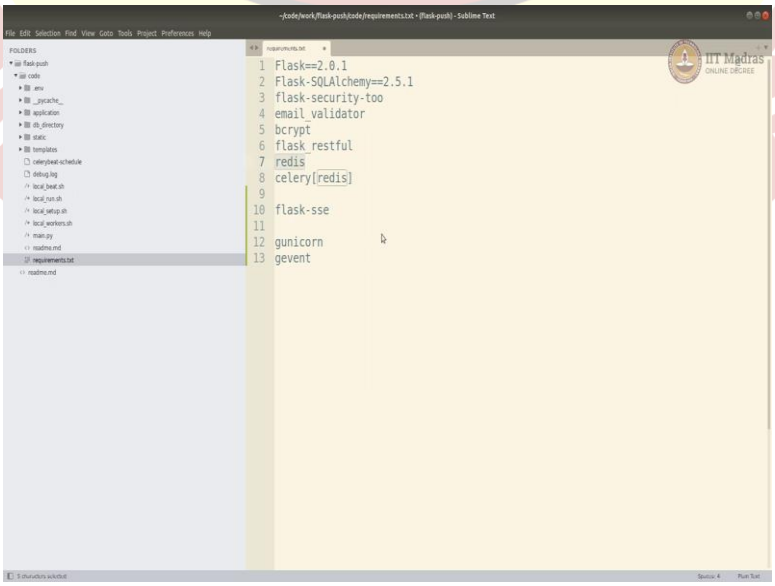
pass the credential because some of these pushes could be credential controlled, then you say with credentials equal to true, then it will send all the cookies and stuff.

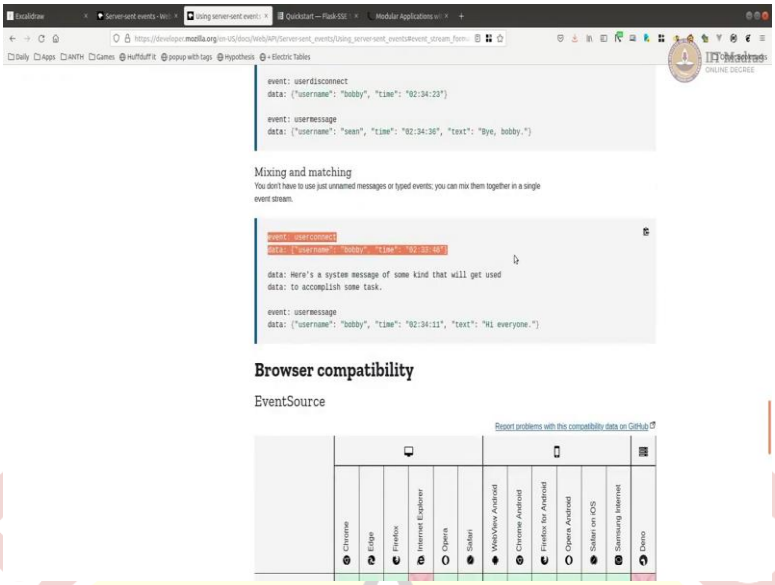
(Refer Slide Time: 07:25)



Now, this EventSource returns, makes a call back on message whenever there is a new message sent through the server. And then you can handle the message in whatever way you want to handel. And this event itself will contain the data like, it will contain all of that, those four things that I told you, even data, event type, and things like that. So, you can take the event data and handle it. Now let us see how to implement this in our flask application. We will also look at how to implement in view, and without view.

(Refer Slide Time: 08:03)

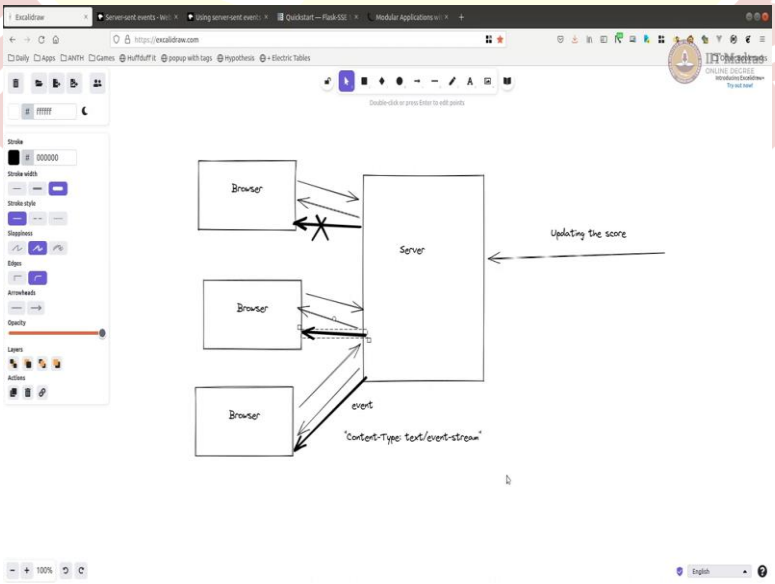




To start with, we need to install three more packages, one is called flask-sse, other one is gunicorn, and gevent. Now, flask sse gives the functionality that you do not need to build these EventStream raw messages by yourself. Instead, it can handle it can abstract all of that so that you can do it very easily. So, that, you do not have to build these messages and push it using raw. If you want to experiment, you can write that too.

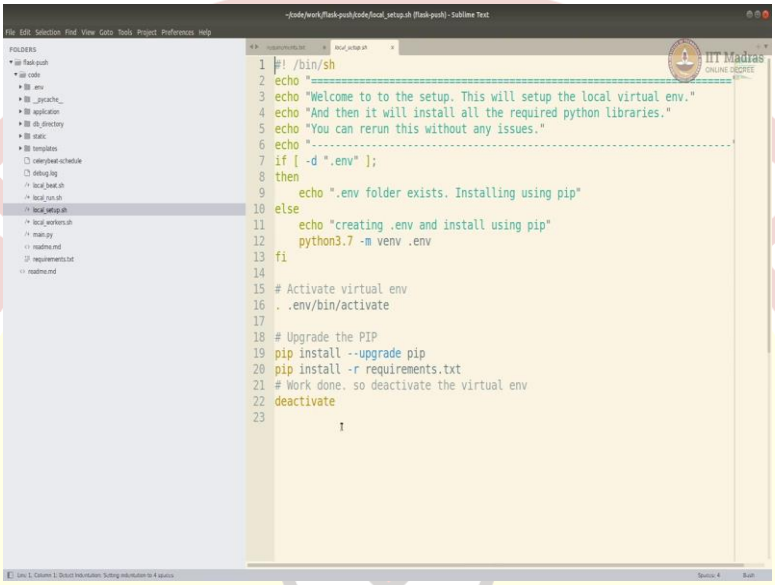
But to make it easy, flask-sse wraps around it. And it is easy to use. And it also uses Redis internally. So, you should have Redis installed already. So, I think since we are already installed Celery Redis is already installed. So, we are going to use the same package on same server as well. Why does flask-sse uses Redis, it kind of gives an additional functionality where each message queue on the server is also maintained in Redis.

(Refer Slide Time: 09:19)



And every customer or every user on the browser is subscribed through that and hence kind of gives you a scalability Pub/Sub feature through only pushed through the HTTP transport layer. So, just to be sure, this is still an HTTP call, HTTP protocol. It is not anything different from your regular protocol. So, it kind of uses all of the HTTP features and all of your regular servers should work.

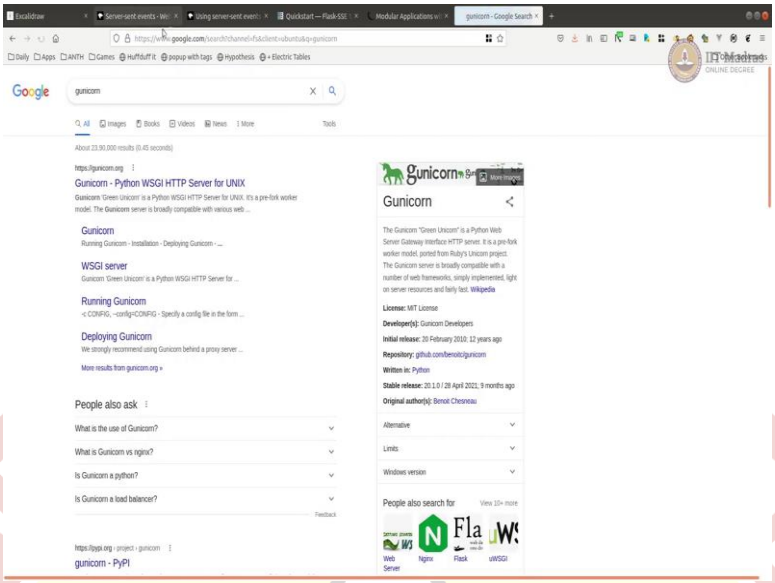
(Refer Slide Time: 09:52)

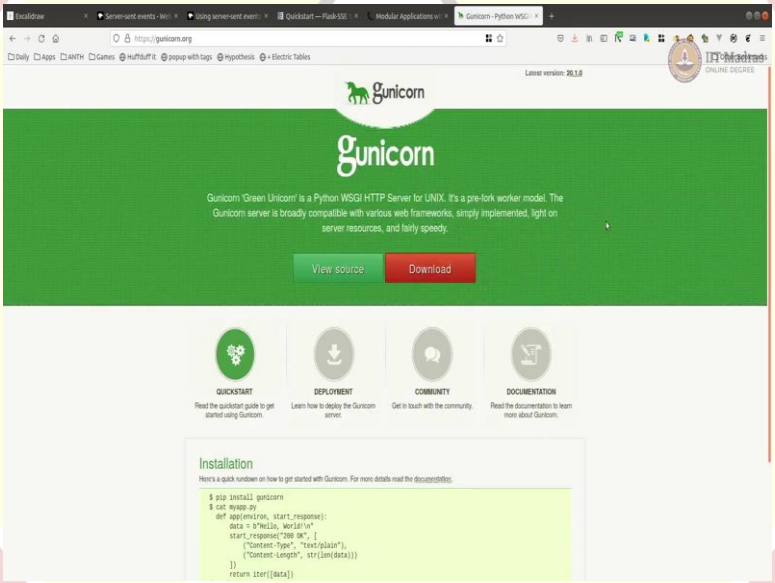


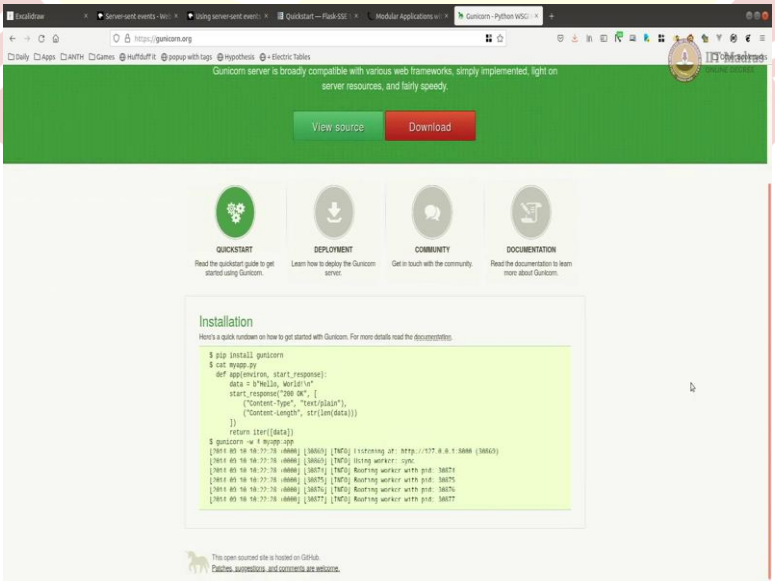
Now to run locally. Until now, we were using, if you look at our local setup, we are using develop server, local run, develop server and running the python main.py. Now, this is a single thread and single process. And that can handle it. Because now, at any point, you could have multiple browsers connecting, and that might need multiple process to run on the server.

So, we have to use the proper application server and hence we are going to use gunicorn that is what we are going to install it and we will have a look at it what it means. And then we are going to use gevent to give like an asynchronous feature on the server side, so that when it is sent, each one does not keep waiting in loop, but it can send them then go to the next one, etcetera, etcetera.

(Refer Slide Time: 10:53)

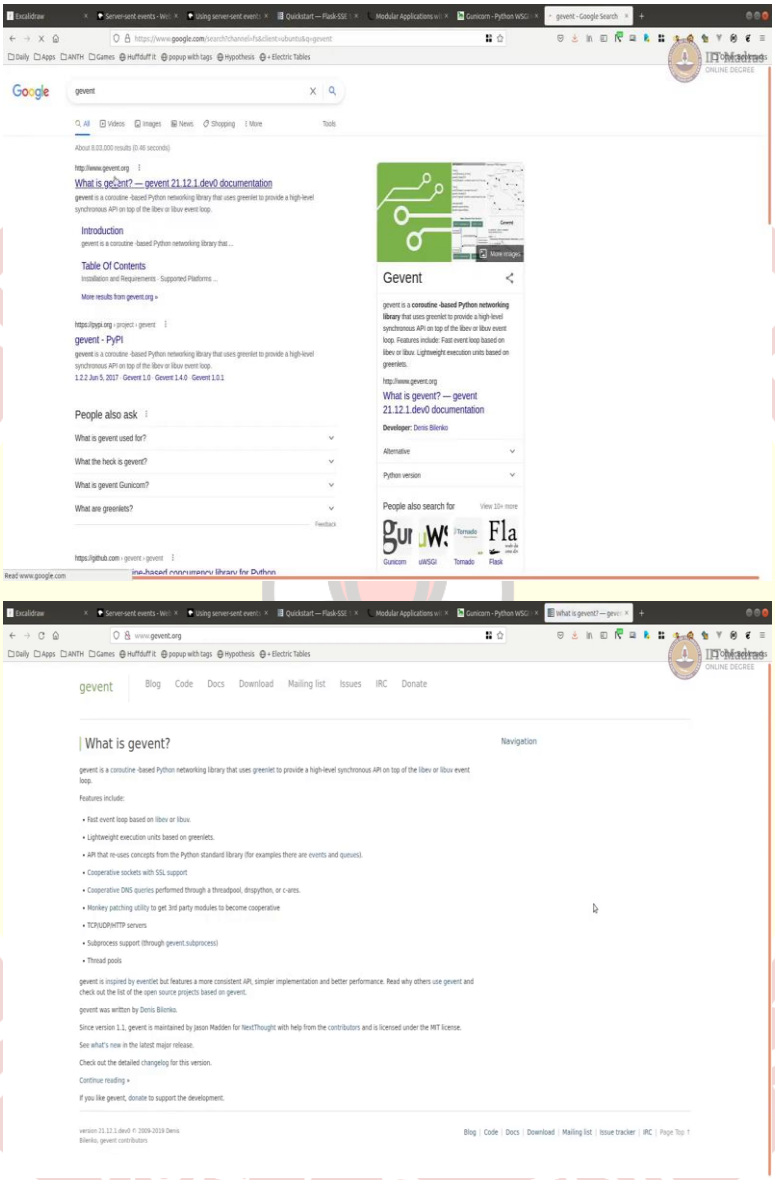






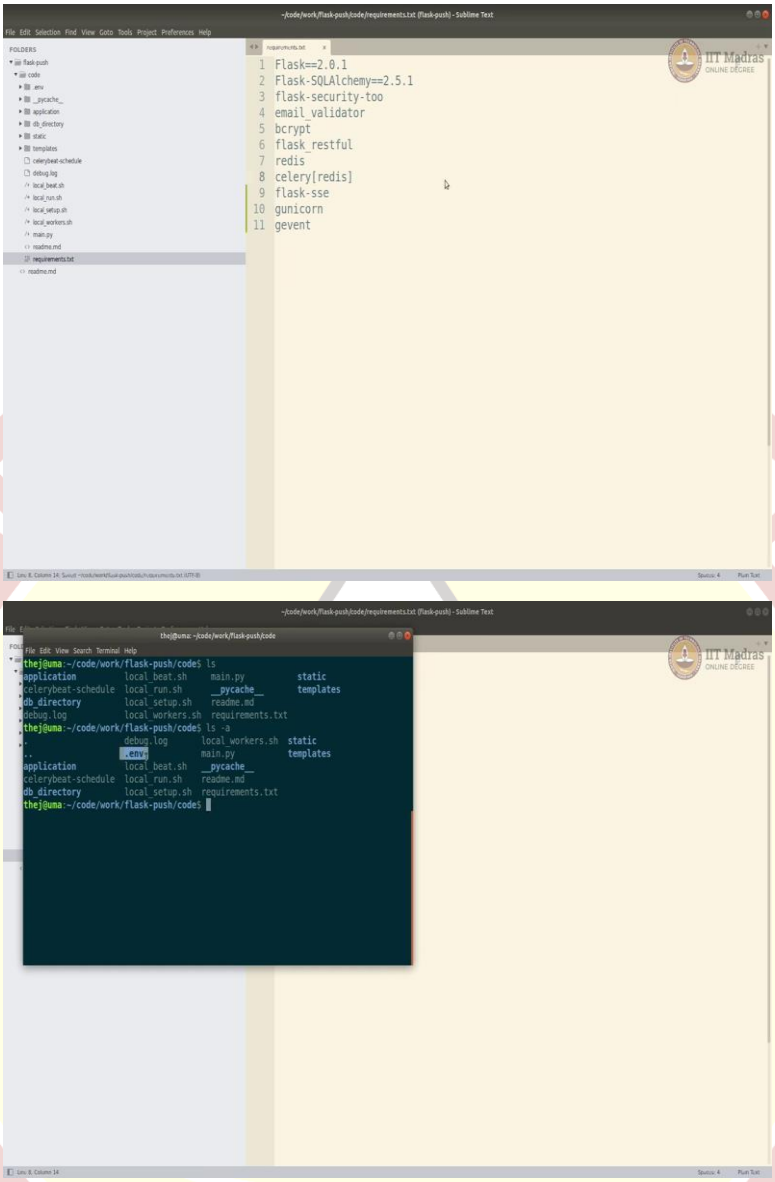
You can learn more about gunicorn here, on the gunicorn console. I think in the last term, we did look into how to install and use gunicorn. So, it is here you can try it out. Similarly, gevent also you can have a look at it.

(Refer Slide Time: 11:07)



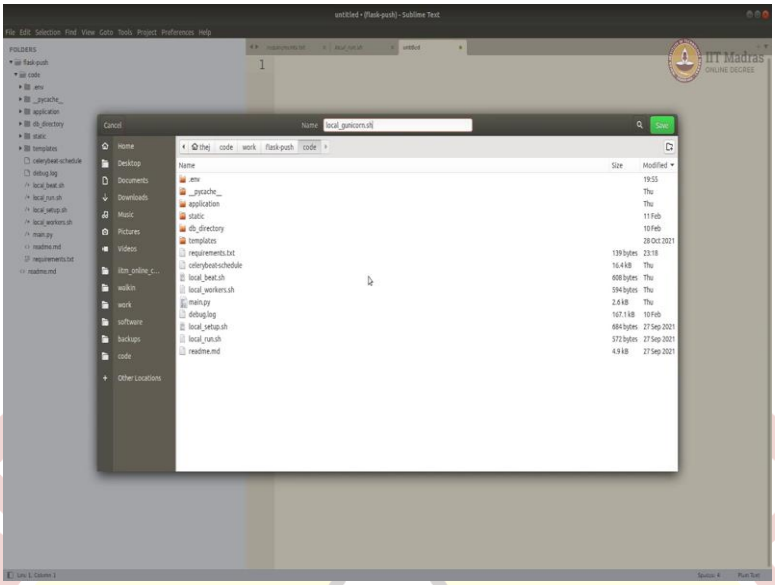
Both are packages, Python packages are available, you can go into details of them.

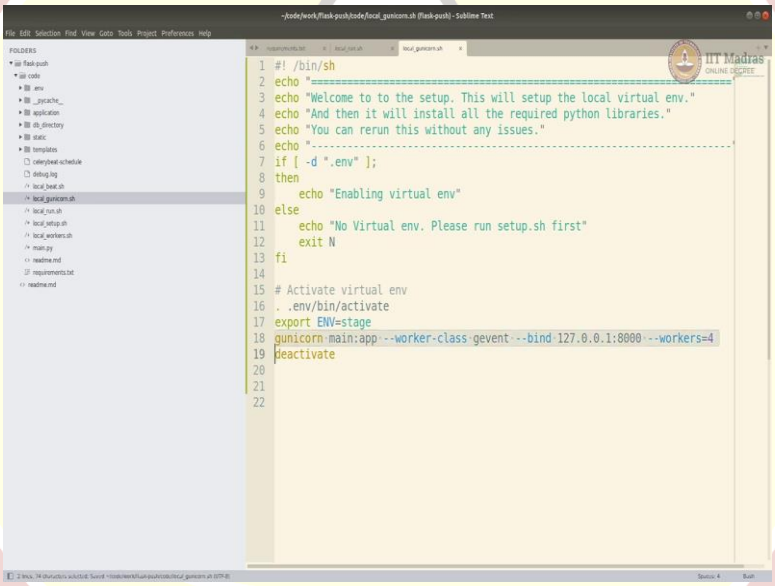
(Refer Slide Time: 11:24)

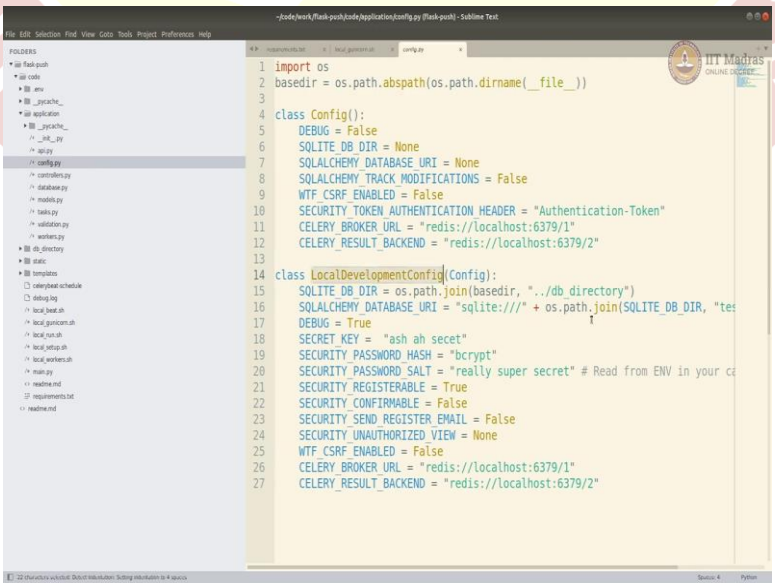


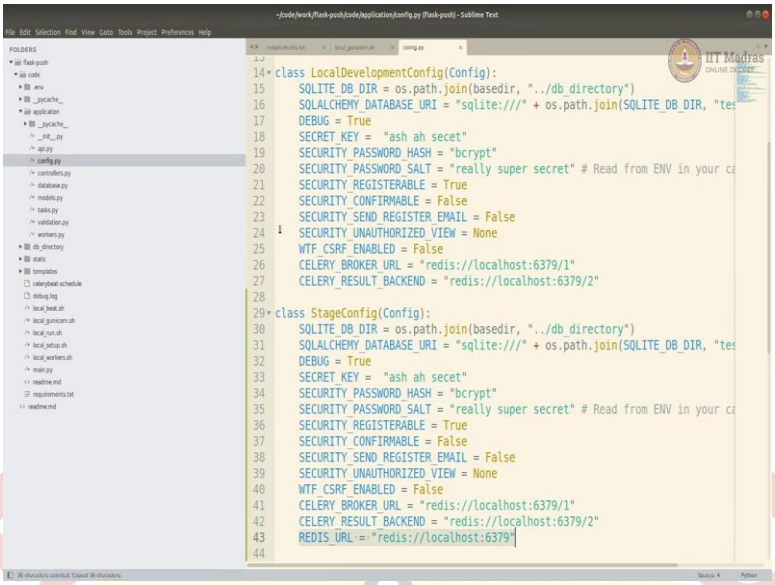
Now, once you put them into requirements dot txt, you need to install them, I have already installed them. So, my environment is set up in dot env. So, I do not have to do anymore.

(Refer Slide Time: 11:39)









```
class LocalDevelopmentConfig(Config):
    SQLITE_DB_DIR = os.path.join(basedir, "../db_directory")
    SQLALCHEMY_DATABASE_URI = "sqlite:///" + os.path.join(SQLITE_DB_DIR, "tes
    DEBUG = True
    SECRET_KEY =  "ash ah secet"
    SECURITY_PASSWORD_HASH = "bcrypt"
    SECURITY_PASSWORD_SALT = "really super secret" # Read from ENV in your ca
    SECURITY_REGISTERABLE = True
    SECURITY_CONFIRMABLE = False
    SECURITY_SEND_REGISTER_EMAIL = False
    SECURITY_UNAUTHORIZED_VIEW = None
    WTF_CSRF_ENABLED = False
    CELERY_BROKER_URL = "redis://localhost:6379/1"
    CELERY_RESULT_BACKEND = "redis://localhost:6379/2"

class StageConfig(Config):
    SQLITE_DB_DIR = os.path.join(basedir, "../db_directory")
    SQLALCHEMY_DATABASE_URI = "sqlite:///" + os.path.join(SQLITE_DB_DIR, "tes
    DEBUG = True
    SECRET_KEY =  "ash ah secet"
    SECURITY_PASSWORD_HASH = "bcrypt"
    SECURITY_PASSWORD_SALT = "really super secret" # Read from ENV in your ca
    SECURITY_REGISTERABLE = True
    SECURITY_CONFIRMABLE = False
    SECURITY_SEND_REGISTER_EMAIL = False
    SECURITY_UNAUTHORIZED_VIEW = None
    WTF_CSRF_ENABLED = False
    CELERY_BROKER_URL = "redis://localhost:6379/1"
    CELERY_RESULT_BACKEND = "redis://localhost:6379/2"
    REDIS_URL = "redis://localhost:6379"
```

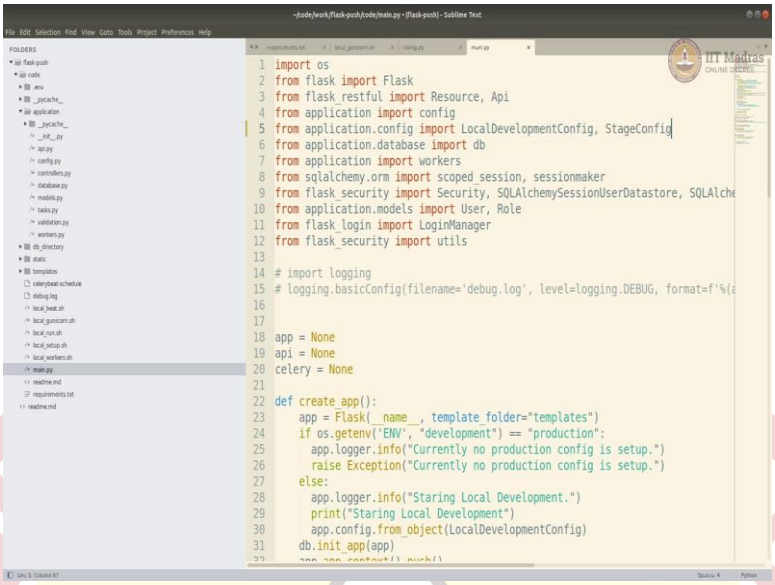
Now, like I said, instead of local run, we are going to use gunicorn. So, I am going to make a new file. And I am not going to update a local run, I am going to keep it like that just for us to know what is the difference. And I will call local gunicorn dot sh. Now, instead gunicorn run, local gunicorn run. This is how it looks. Though, you can see that only difference is gunicorn.

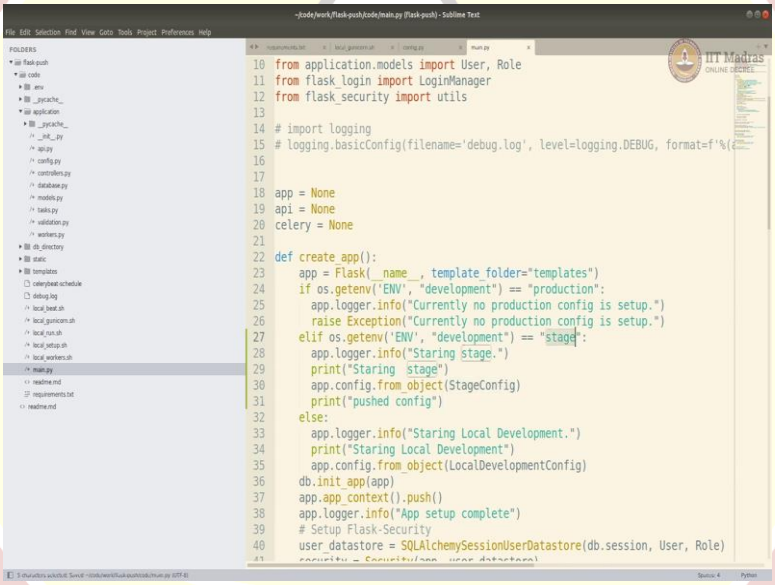
And it takes the flask app that needs to run and a worker class is gevent and I am running it on port 8000. Am I going to have at least four workers, you can have many workers based on the CPU, your machine has, etcetera, etcetera. And I am going to run it in a staging like a define a new environment called stage. And use that because I do not want to modify my local config yet.

But if you want, if you are going to do that you can also update, you could have updated this. But I am going to create a new config, just to make sure it is all clean and separate for you to experiment. Let me add them, I am going to keep most of the things same. everything is same, except there is Redis URL here. This is the Redis URL that sse is going to use in a new environment.

So, I am going to add across all three. That is the only extra thing that we are going to add here, Redis here and we have defined a stage config. And since we are defined a stage config, we are going to import that in our main.

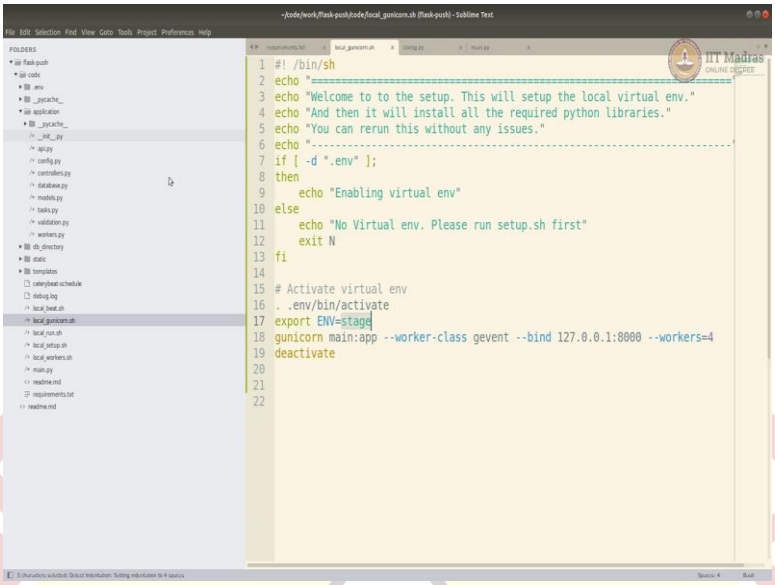
(Refer Slide Time: 13:32)



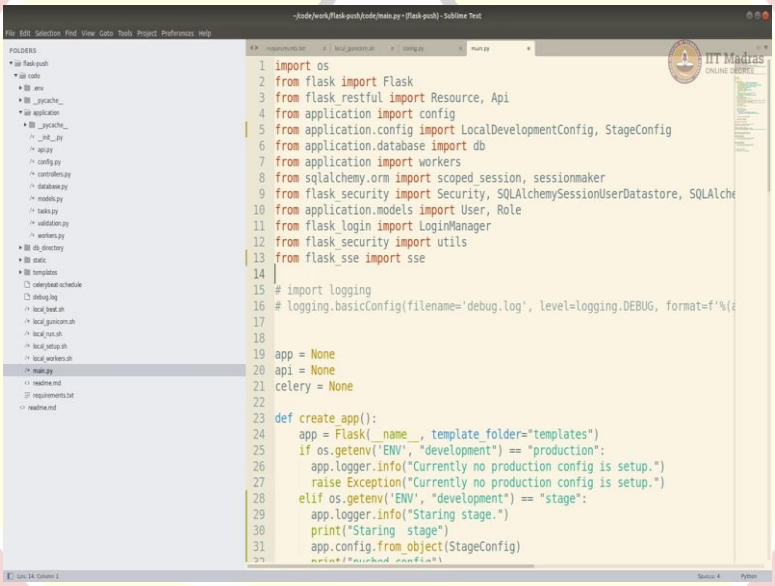


We have local config here, we have to import the stage conflict and use it to, got it? Now, here we had one production. Now, we will also add like a stage if else. Let us add that here, you get nothing special. It is the same thing. But just in quotes on stage configure the environment is stage. So, that part is done.

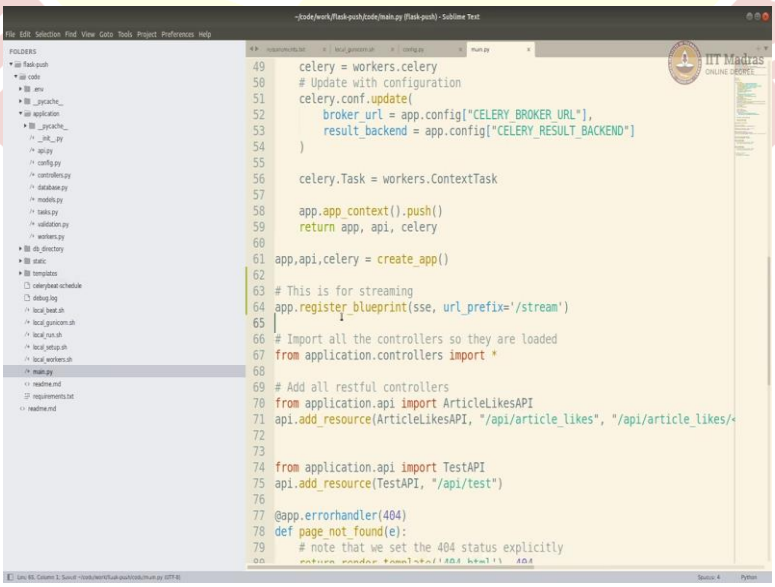
(Refer Slide Time: 14:11)



```
#! /bin/sh
echo "============================================================"
echo "Welcome to to the setup. This will setup the local virtual env."
echo "And then it will install all the required python libraries."
echo "You can rerun this without any issues."
echo "------------------------------------------------------------"
if [ -d ".env" ];
then
    echo "Enabling virtual env"
else
    echo "No Virtual env. Please run setup.sh first"
    exit N
fi

# Activate virtual env
. .env/bin/activate
export ENV=stage
gunicorn main:app --worker-class gevent --bind 127.0.0.1:8000 --workers=4
deactivate
```



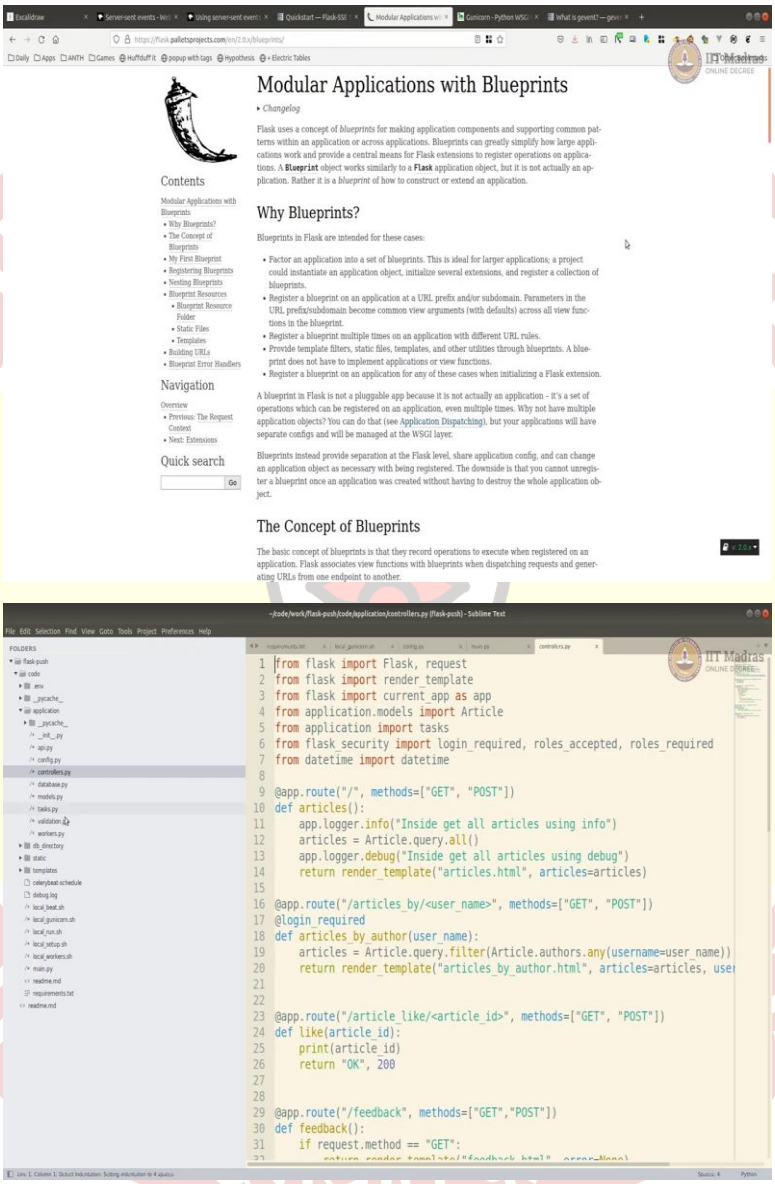
```
import os
from flask import Flask
from flask_restful import Resource, Api
from application import config
from application.config import LocalDevelopmentConfig, StageConfig
from application.database import db
from application import workers
from sqlalchemy.orm import scoped_session, sessionmaker
from flask_security import Security, SQLAlchemySessionUserDatastore, SQLAlche
from application.models import User, Role
from flask_login import LoginManager
from flask_security import utils
from flask_sse import sse

# import logging
# logging.basicConfig(filename='debug.log', level=logging.DEBUG, format=f'%(a


app = None
api = None
celery = None

def create_app():
    app = Flask(__name__, template_folder="templates")
    if os.getenv('ENV', "development") == "production":
      app.logger.info("Currently no production config is setup.")
      raise Exception("Currently no production config is setup.")
    elif os.getenv('ENV', "development") == "stage":
      app.logger.info("Staring stage.")
      print("Staring  stage")
      app.config.from_object(StageConfig)
```

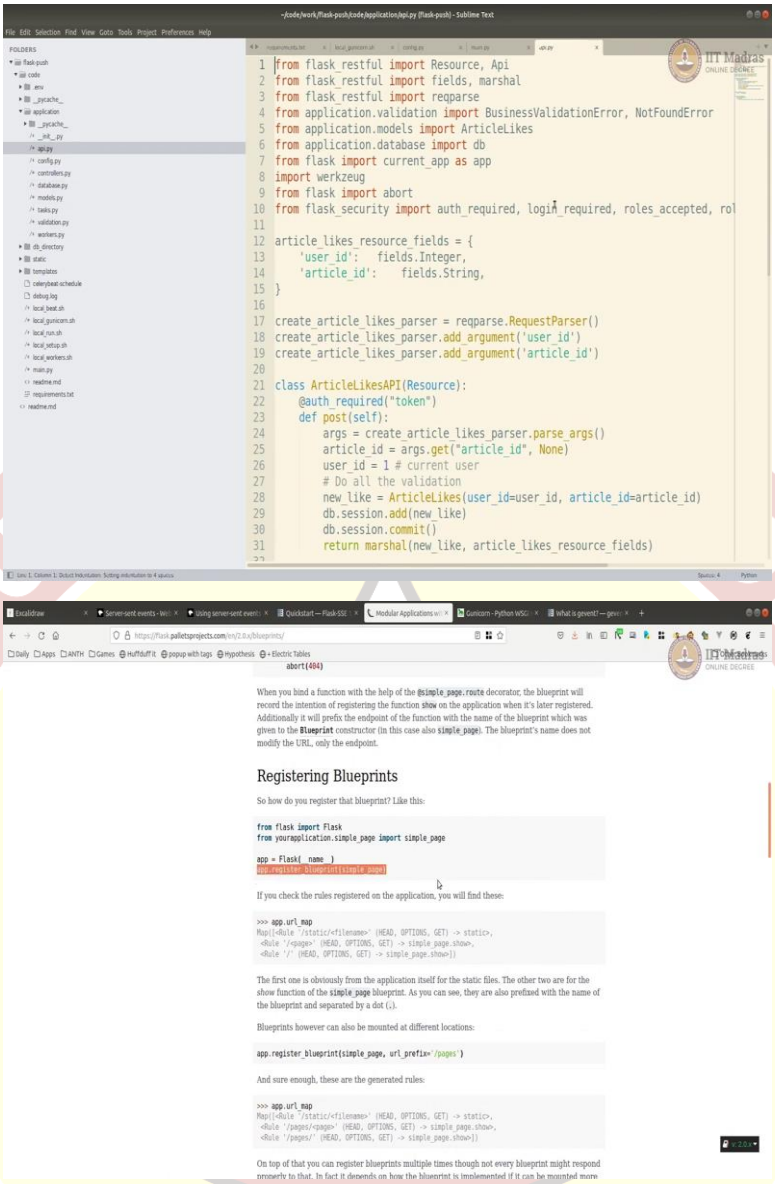


```
    celery = workers.celery
    # Update with configuration
    celery.conf.update(
        broker_url = app.config["CELERY_BROKER_URL"],
        result_backend = app.config["CELERY_RESULT_BACKEND"]
    )

    celery.Task = workers.ContextTask

    app.app_context().push()
    return app, api, celery

app,api,celery = create_app()

# This is for streaming
app.register_blueprint(sse, url_prefix='/stream')

# Import all the controllers so they are loaded
from application.controllers import *

# Add all restful controllers
from application.api import ArticleLikesAPI
api.add_resource(ArticleLikesAPI, "/api/article_likes", "/api/article_likes/<

from application.api import TestAPI
api.add_resource(TestAPI, "/api/test")

@app.errorhandler(404)
def page_not_found(e):
    # note that we set the 404 status explicitly
```

Now, we will have to set up the a flask-sse. It is quite straightforward. The first thing is to actually import flask-sse. And then you have to set up the flask-sse that we are going to do here. So, now, actually here using we are using something called blueprint, passing the sse and URL prefix to that.

(Refer Slide Time: 14:47)

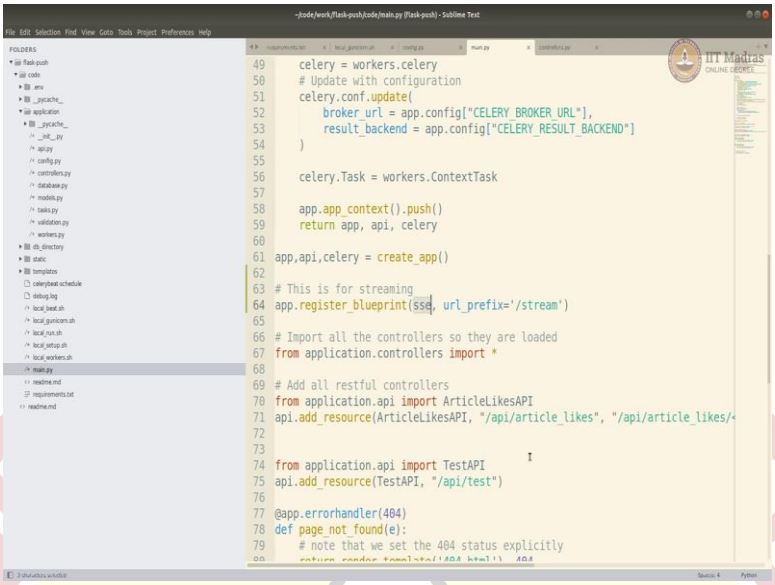


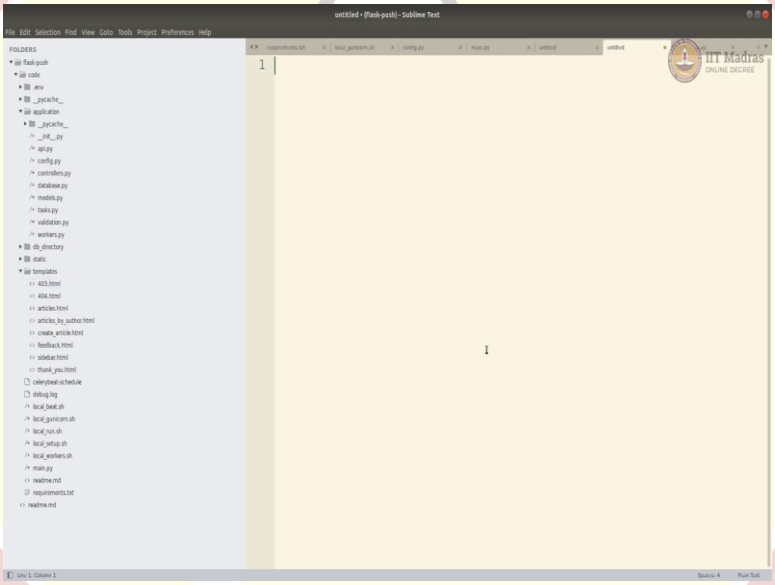


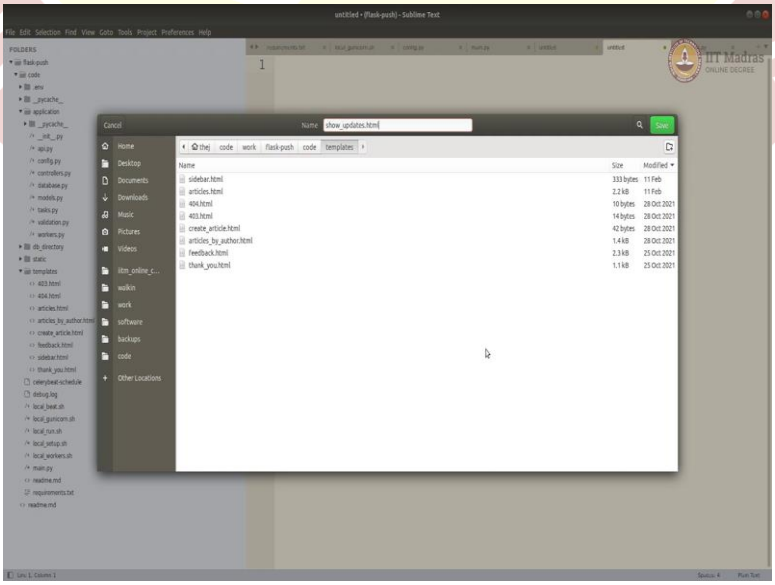
Blueprints are a feature of flask. We can, okay, feature of our flask. You can go learn about it. It kind of makes it easy to manage controllers for example, if you have or separate out the controllers, we currently have all controllers in the controllers dot py, and all APIs in api dot py. If you want to separate the controller controllers and make it modular and have their own related stuff in that folder, then you could use blueprints, makes it easy to isolate and keep them separate and clean.

It is like an advanced feature of flask. But it is easy to use. And if you want to make it modular, so this uses blueprint. In blueprint, you define the app and you define the blueprint. And then you register the blueprint with a flask, either with a URL endpoint that it should be enabled for or without it.

(Refer Slide Time: 15:47)

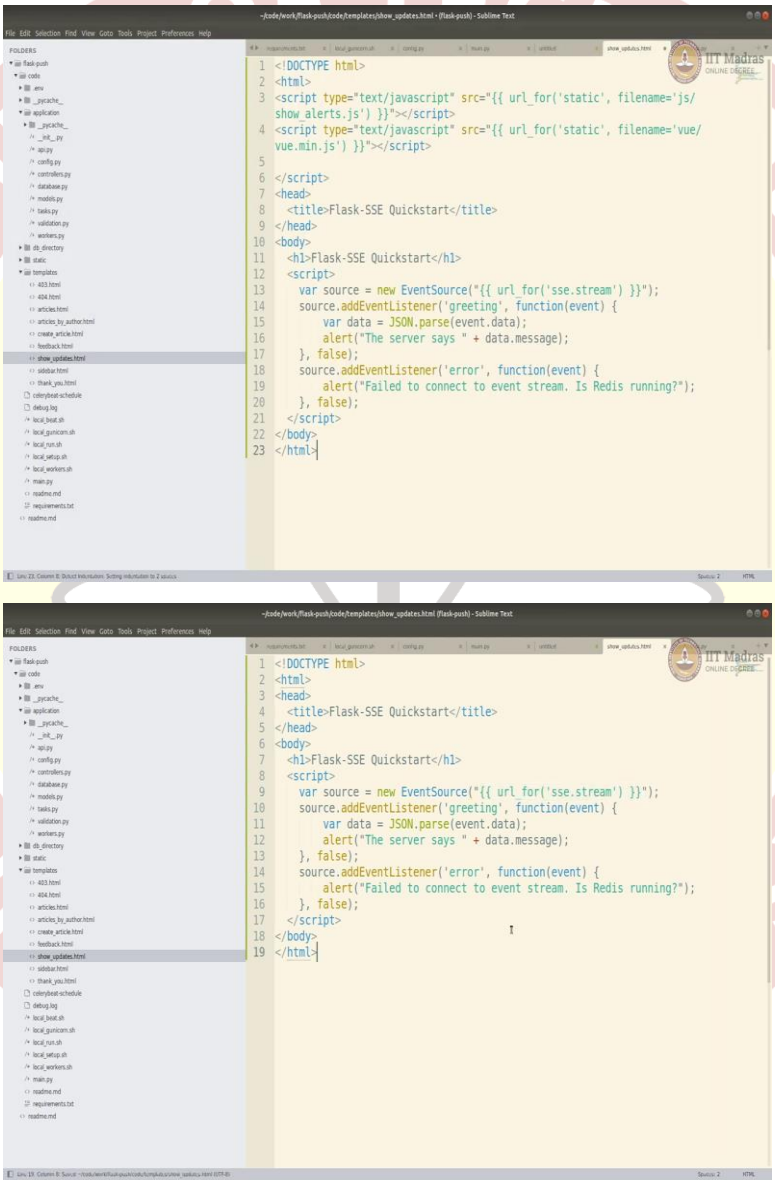






Here, we are going to define one endpoint called stream that gets used by sse to stream the data. And we can have other sub streams like say slash stream slash something, but we are going to have your prefix all of that stream, which will send the sse input. So, this is just the one. Now, on the client side, we need a template right to receive the event and display it somehow. So, I am going to create a template called show update. Let me just call it show update dot html.

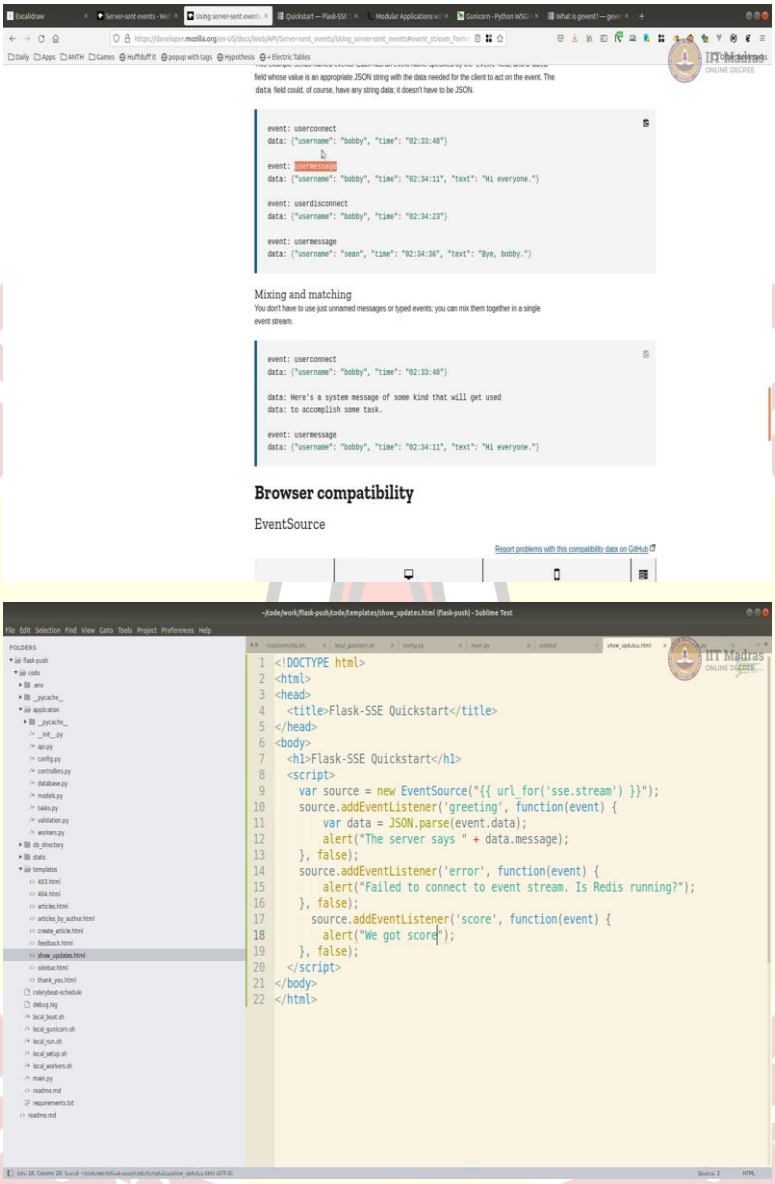
(Refer Slide Time: 16:39)



So, I am just going to paste the whole thing. Currently, you do not need vue, so I am just get rid of this, and of this. So, it is just a simple flask-sse, where I am just subscribing to the sse stream, either you can use this, or even if you want to make it really simple. You could use slash stream. But we can just use that I think it is fine. It kind of makes it easier. Now I am

listening to an event called greeting. And then whatever greeting the server sends, I am pushing it. Now greeting is an event type.

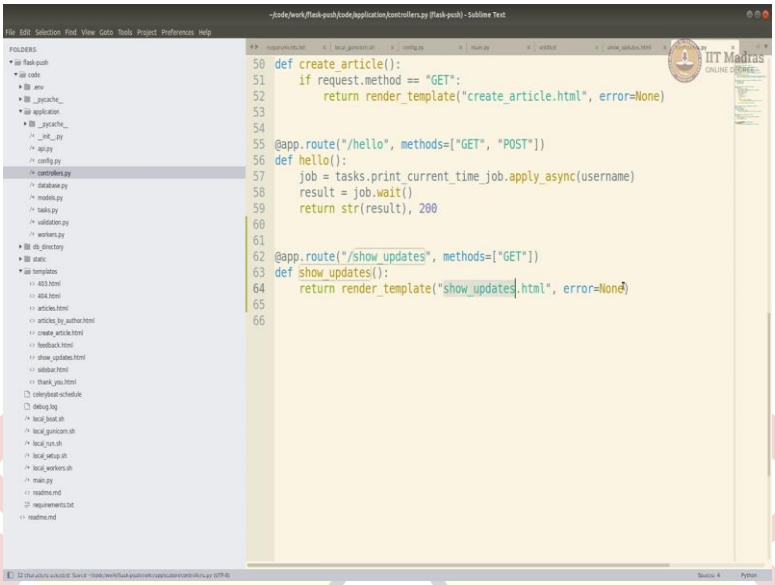
(Refer Slide Time: 17:33)



If you see here, we said there, there can be many event types. Here, we have an event type called greeting. But if you are sending scores, you could have had an event type called scores. And if you are sending event types called update, you can send event type updates, and you can handle all of them differently, you can subscribe to each one of them separately.

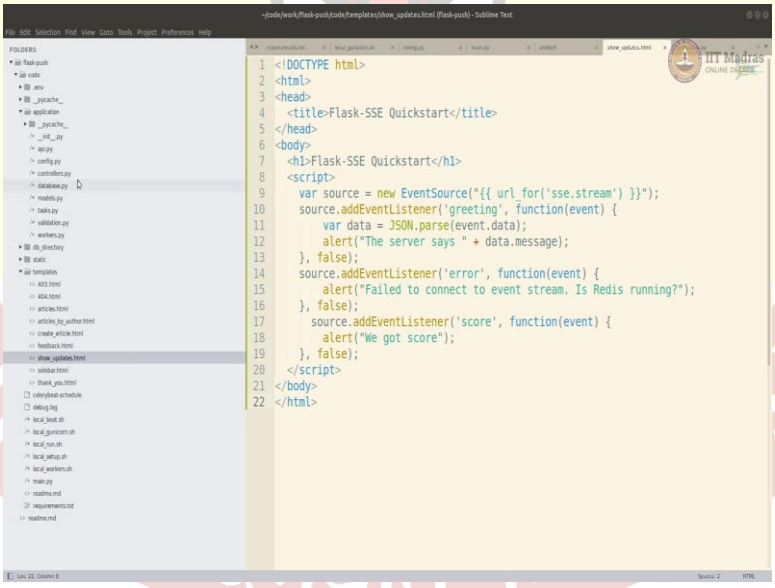
And there is a standard one called error. If there is an error. You could have many, for example, forget scores, you could do something else with that, like for example score, as an event type, you can handle that separately, like we got score, for example. So, now we have show update.

(Refer Slide Time: 18:22)



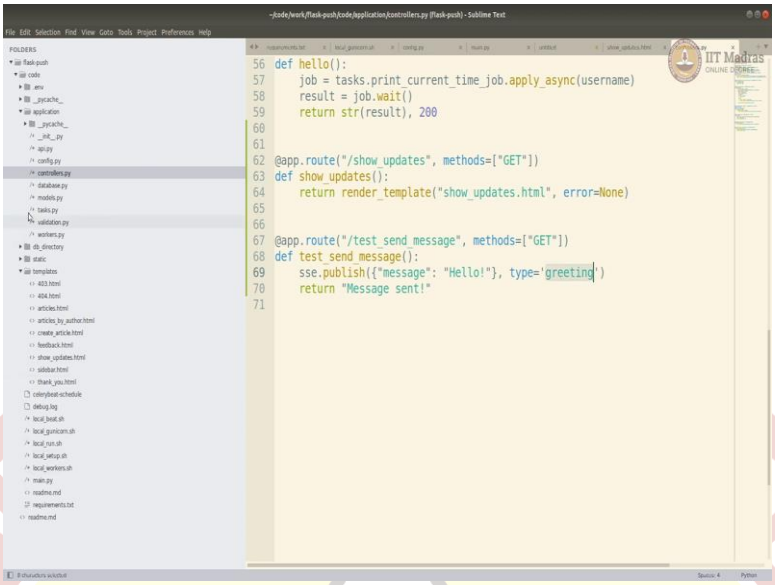
Now I have to modify my controller to go to that page, show updates page. So, I am just going to add, show update controller and to show update. I am just going to replace this. Just to get and it kind of enter show update dot html and says.

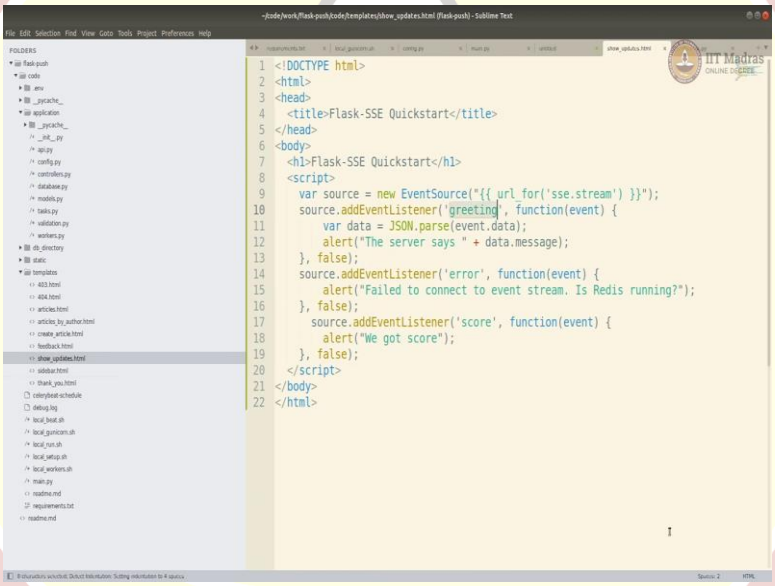
(Refer Slide Time: 18:49)



In show update, you are actually connecting to the stream, you could have used local slash stream, but I am going to do a dot sse stream. Because this is built in template, jinja template. And this is built in flask function to use to view the URL for this. Now, I think we have kind of ready to receive, but who will push the data into this stream. So, we will create a dummy function, dummy controller to send some test message into that stream so that can be shown to the user. So, we will do another controller.

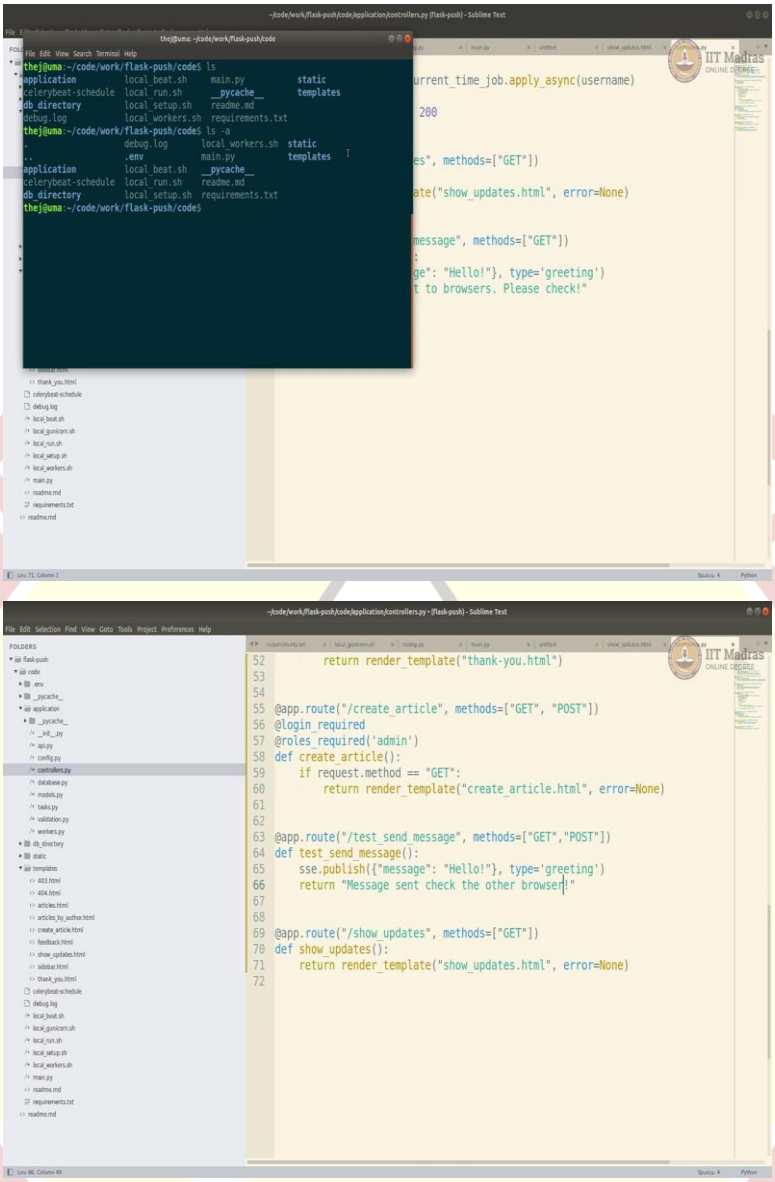
(Refer Slide Time: 19:32)





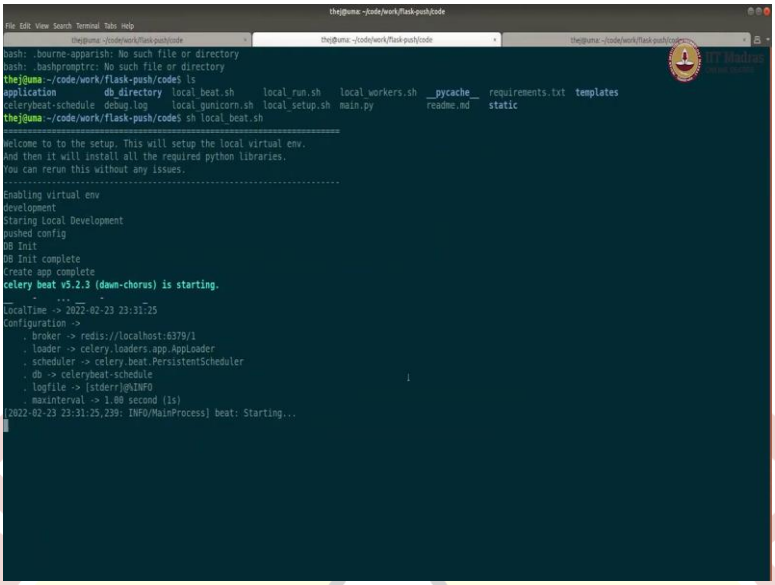
I will say test message. And this test message is also a get, I am just going to access it, which will actually call sse dot publish. This is how you publish to the client. Once this is called all this, this is called one sse publish. Gets called here, everything gets published to all the browsers. And it is of type greeting. You can see here it is of type greeting, which is what we have subscribed to here and the first one greeting. That is it actually just a message sent to browsers. Please check. Let us save this. And let us start the server.

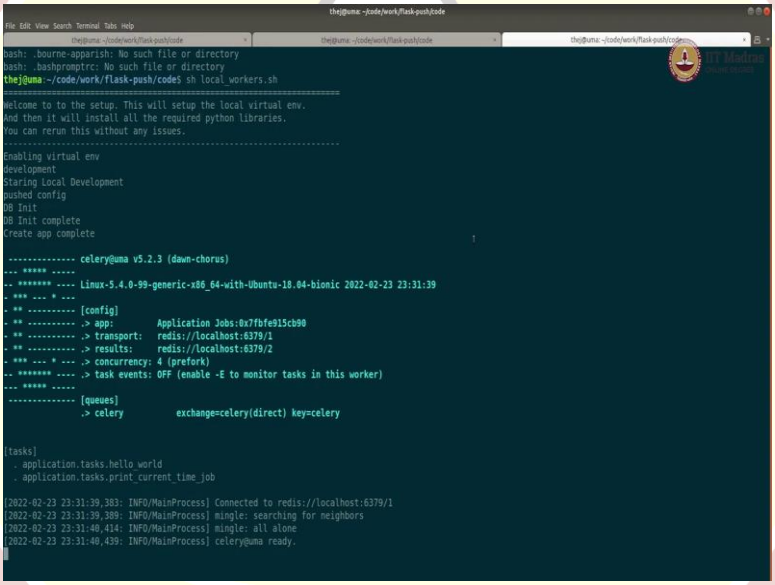
(Refer Slide Time: 20:27)

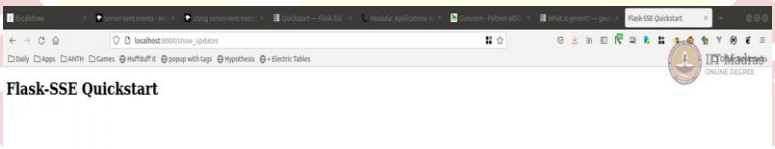


My radius is already running. So, I am not going to start that now. So, we have show dates and then publish. Message Sent. Check the other browser.

(Refer Slide Time: 20:51)

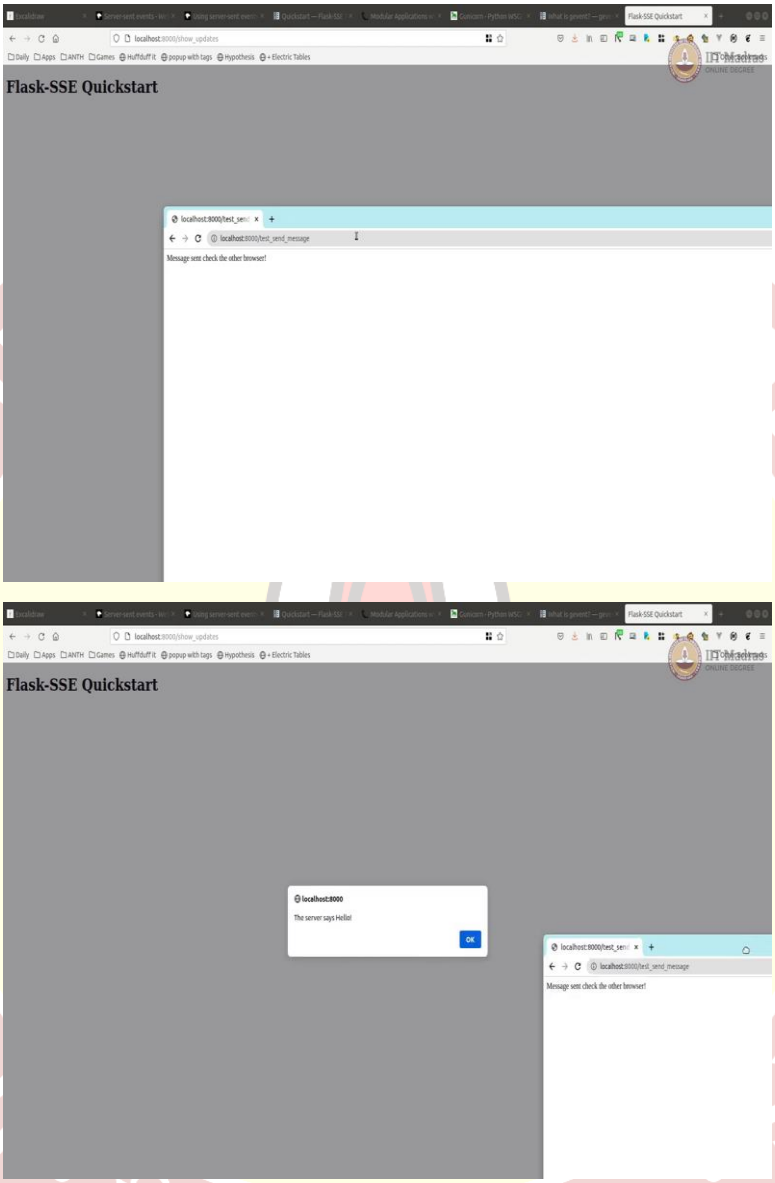






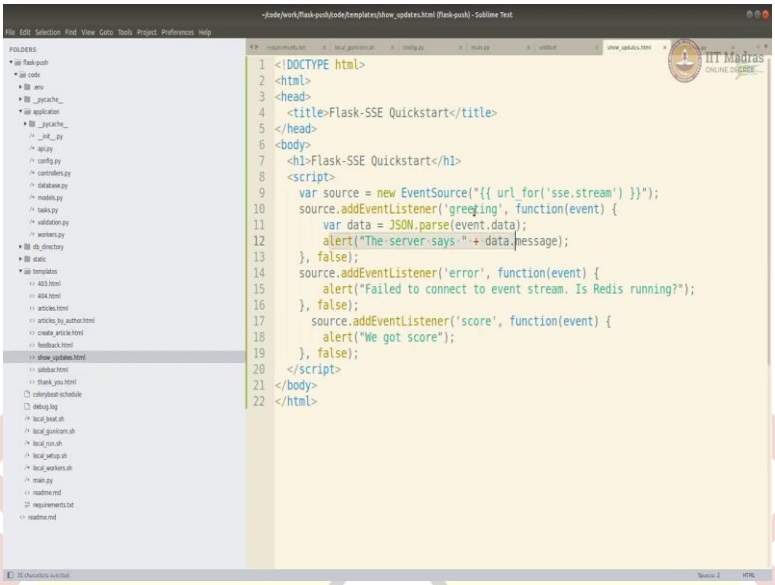
Let us start this. I have also started my celery beat, and celery worker. Now, if I go to this place, it has flask-sse quick start. Now we want to publish something to it.

(Refer Slide Time: 21:13)



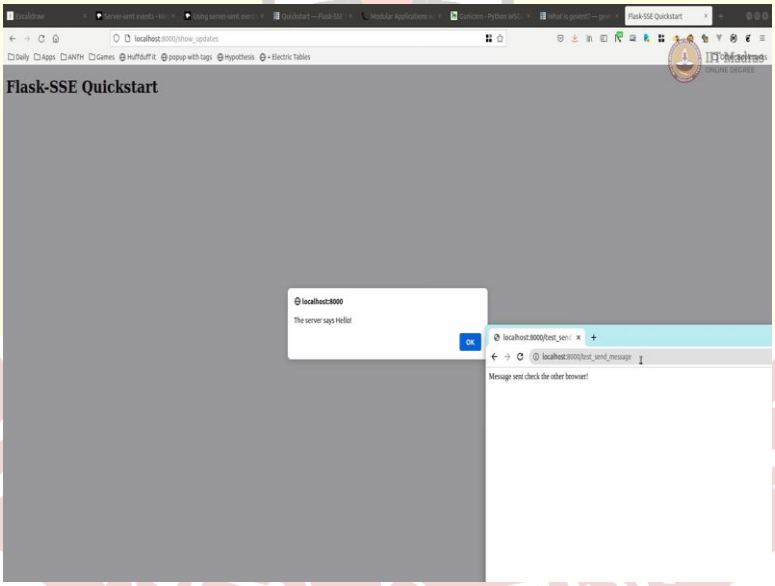
Let us publish from a different browser. It was called send test, send message, localhost colon 8000. So, message sent. And here it says, server says hello. We could do one more small change here.

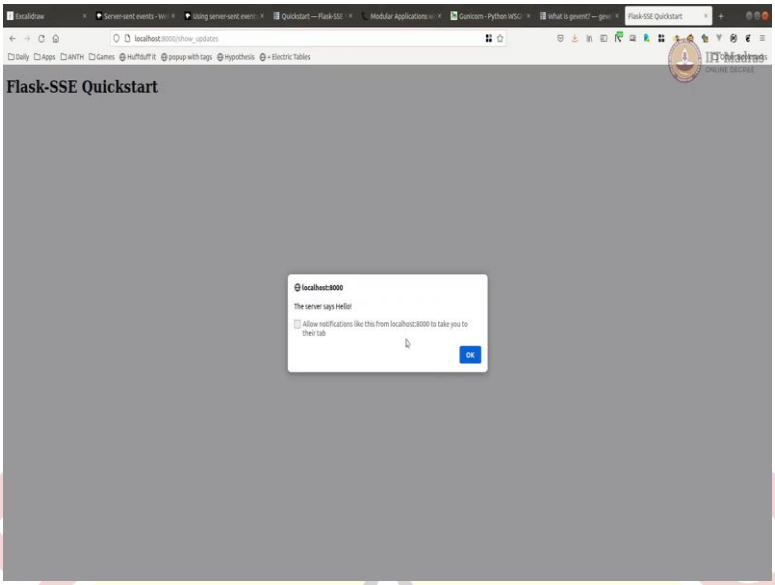
(Refer Slide Time: 21:39)

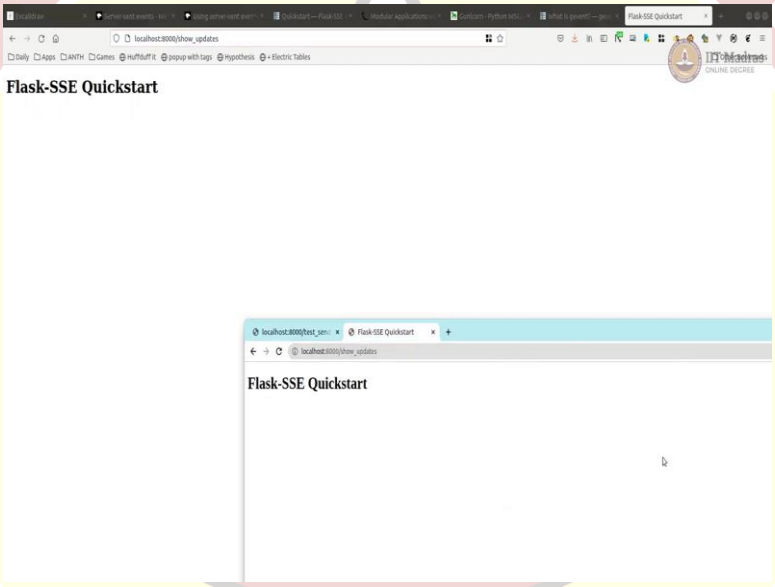


Instead of actually saying alert, you can actually add it to the div. But let us go make one more small change, instead of doing this.

(Refer Slide Time: 21:53)

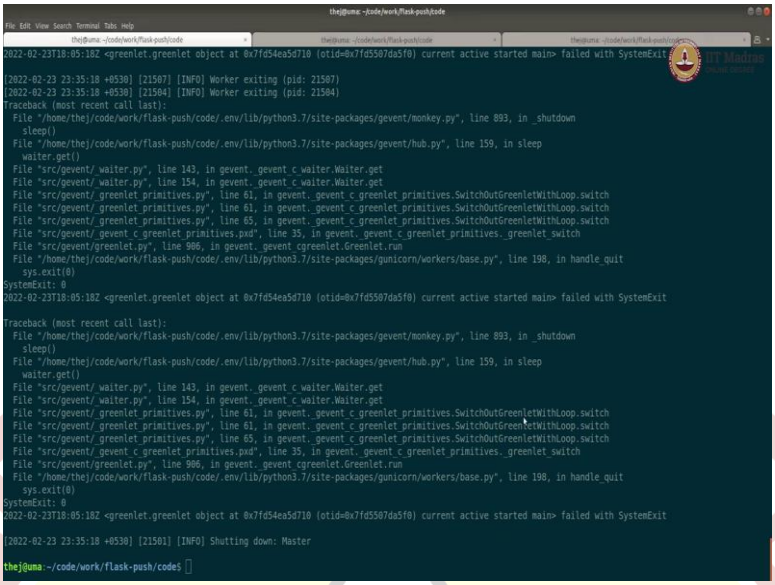






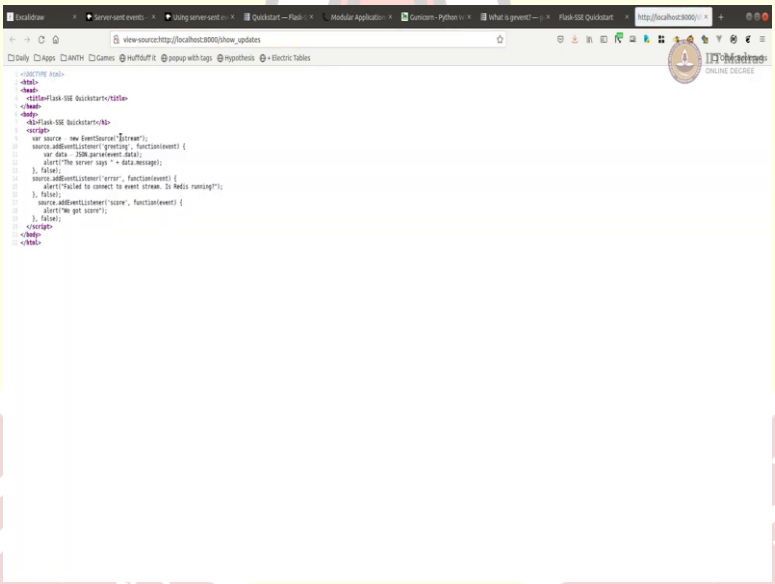
You could, I mean, just going to repeat once more, I am going to press second here, you can see that server says hello. And it can be multiple pages. One here, and one here, Now let us refresh this here, twice, and here, so you get everywhere, wherever the users are, if they were scoring, they would come here and here. You are going to just upgrade it little more, and make it to use with you. And we will update the div, so that you we also know that it can be used with view.

(Refer Slide Time: 22:50)



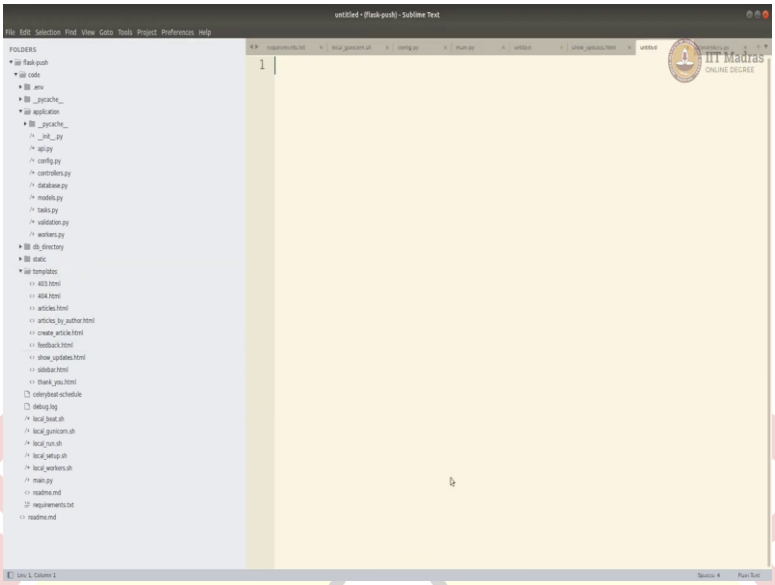
So, now I am going to create another template called show update veu dot html.

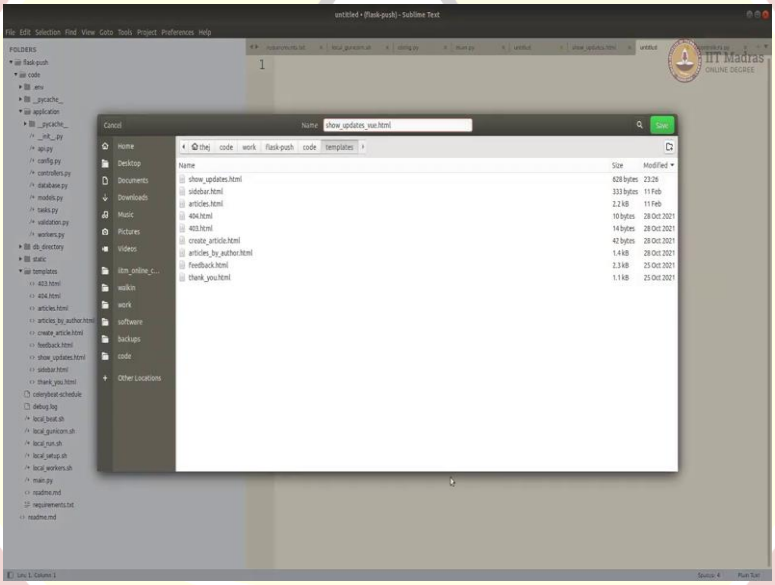
(Refer Slide Time: 23:01)

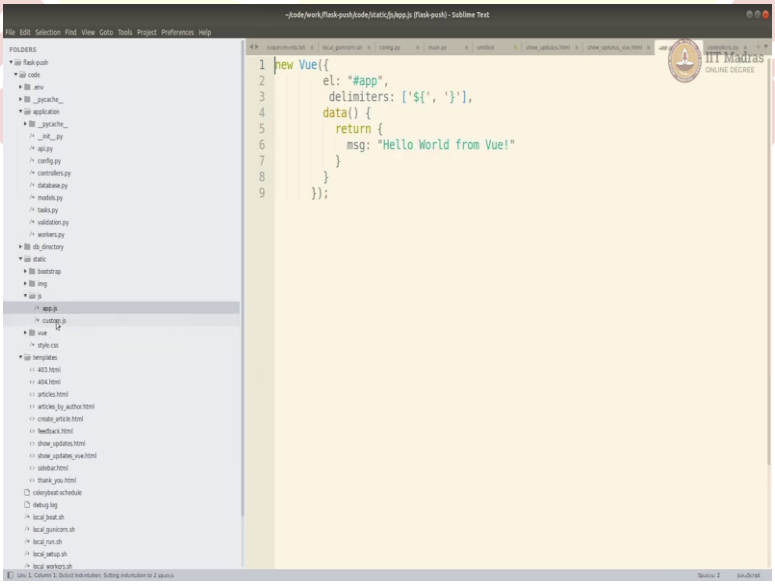


Now, one thing that I wanted to show was this, you can see that it is the slash string. It could have been localhost colon slash, 8000 slash stream, what it is, slash stream is appended to the domain. So, it works like that.

(Refer Slide Time: 23:20)

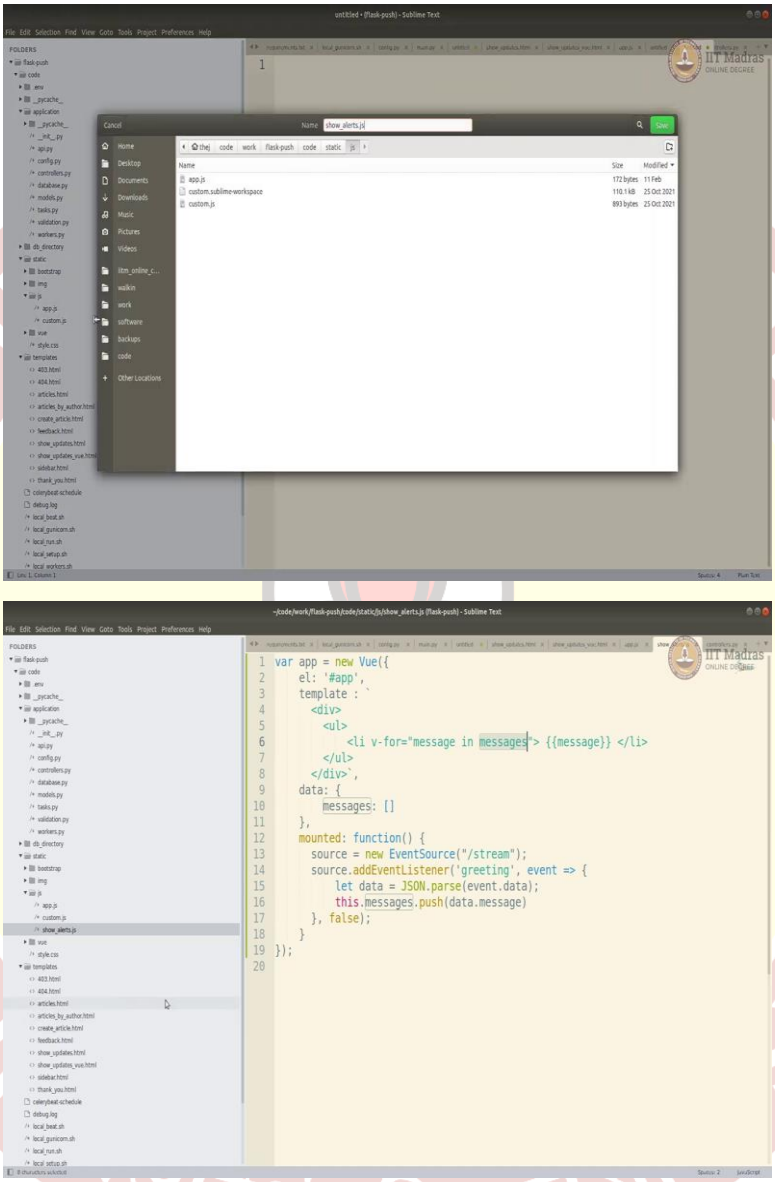






Now, let us create another template, new file. We are going to use veu to do it. We are going to create a new JavaScript, I mean we are not going to use app, we are going to use a separate app for this. So, let us show let us call it show alert or something.

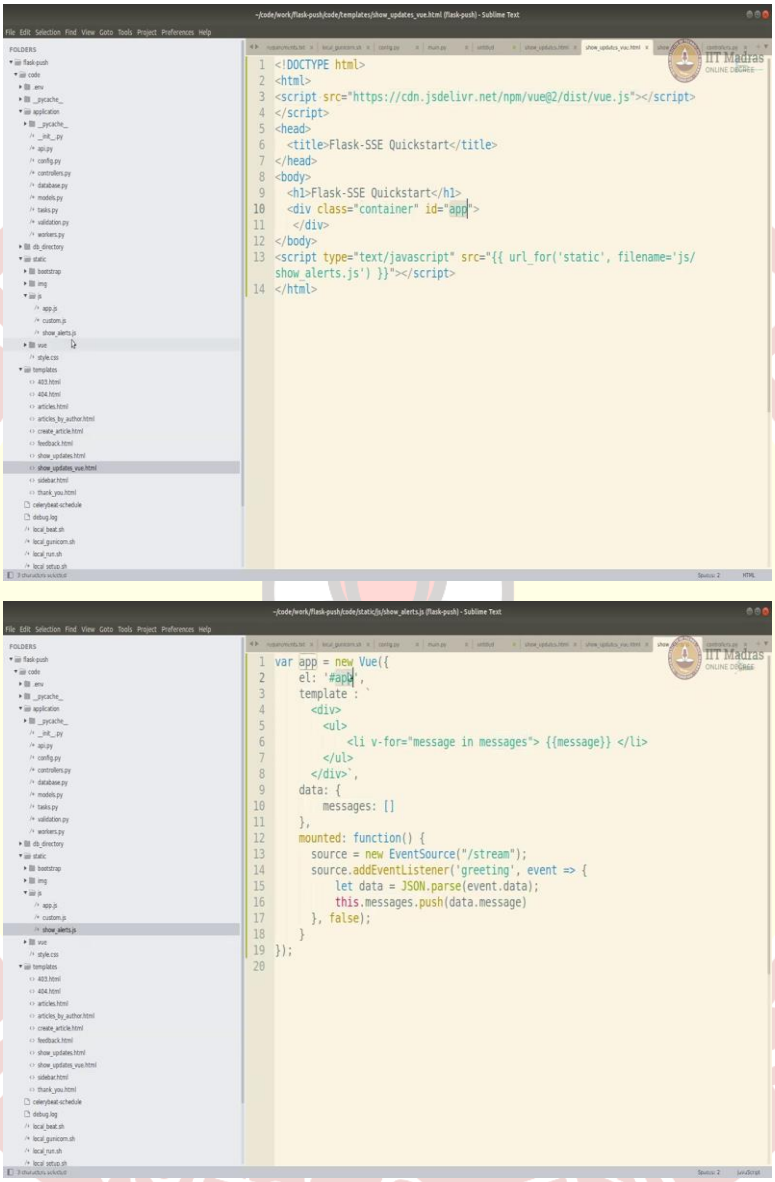
(Refer Slide Time: 23:46)

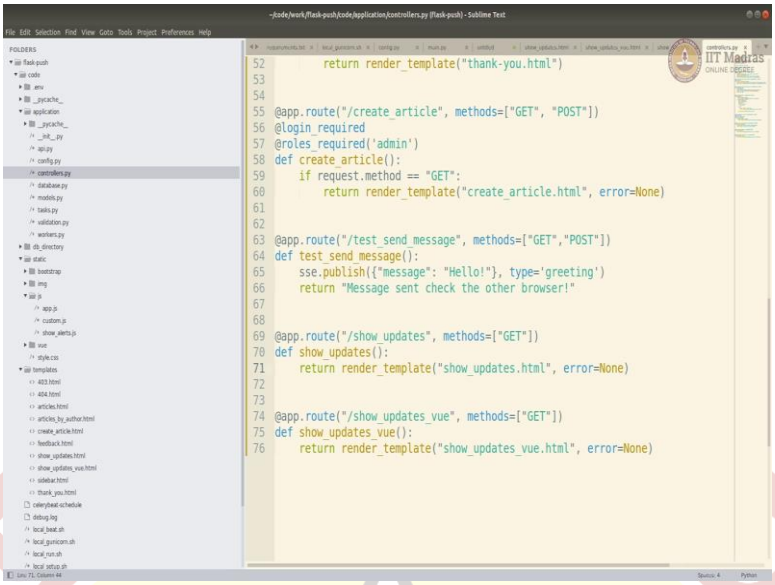


Dot js, in this case, I am just going to past some code and explain. It is pretty straightforward. You have an app, the div app, the template has a div, and it goes through a list pending all the messages in the list. Now, where does the list come from? List comes from data. Now where does the data come from? Data comes from our EventSource when I am going to define the EventSource in our mounted function.

And I am going to add listener to it. Whenever there is the greeting, I am going to pass the data, I am going to take message out of this data and put this message into this array. And hence this should get updated. So, let us go back to our template.

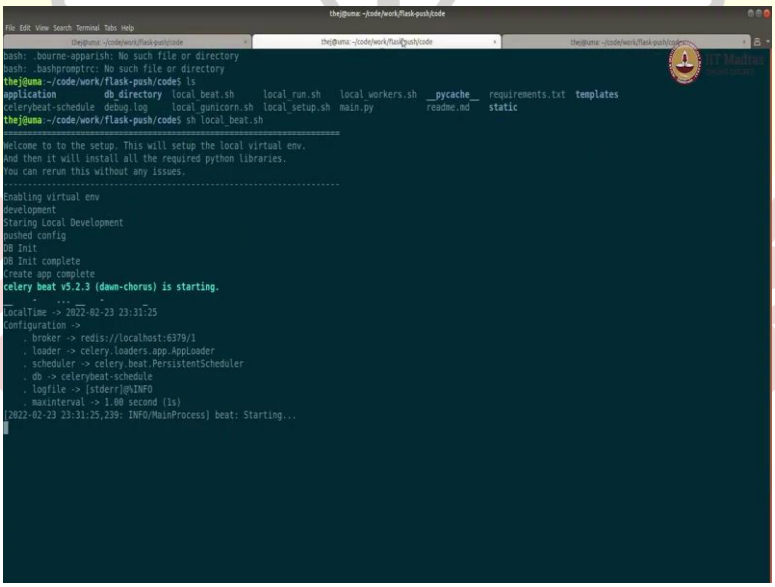
(Refer Slide Time: 24:48)

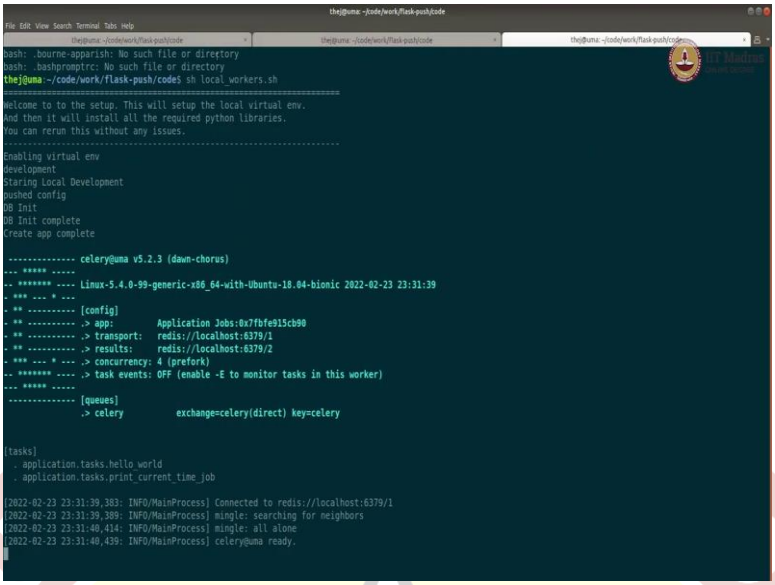


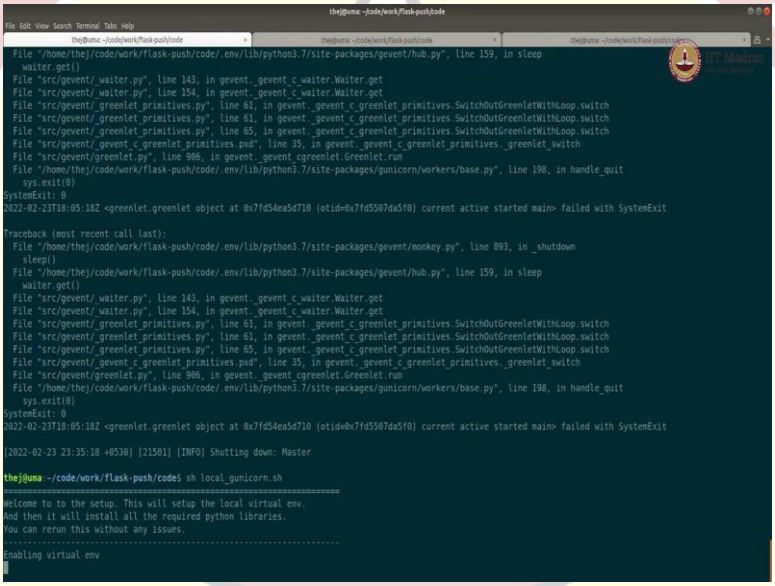


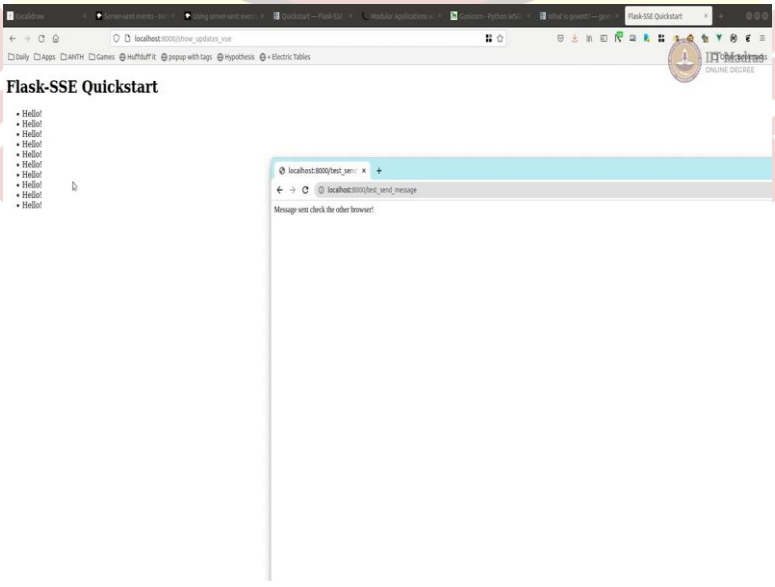
And add everything that is required. I am going to import, it is a simple one, so I am going to import veu directly. And then define a div with id app because that is what we are going to use here app, and then calling the show alerts dot js, that is it. So, now to open this, we have to add a controller, just like Show Update, I am going to write another one called Show update view, it is just going to do the same thing. And then nothing special. We are still going to use publish, and see how it publishes the hello to it.

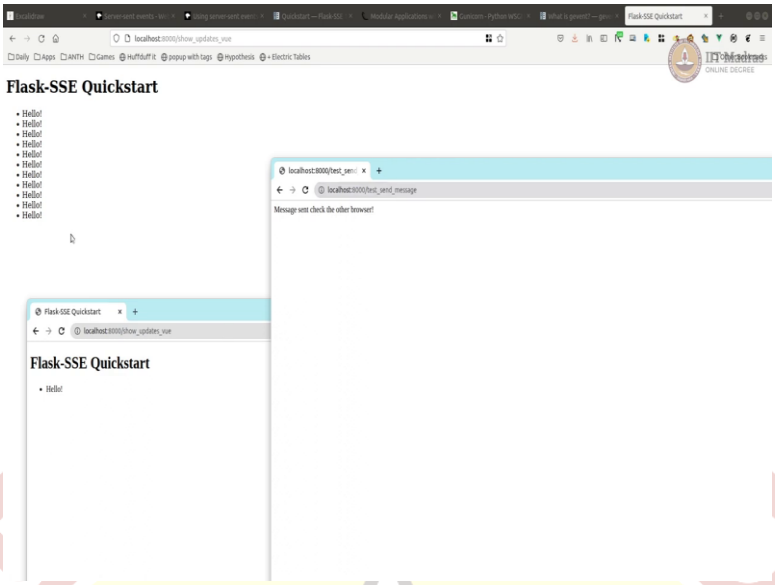
(Refer Slide Time: 25:42)




thej@uma: ~/code/work/flask-push/code
File Edit View Search Terminal Tabs Help
bash: .bourne-apparish: No such file or directory
bash: .bashpromptrc: No such file or directory
thej@uma:~/code/work/flask-push/code$ sh local_workers.sh
Welcome to to the setup. This will setup the local virtual env.
And then it will install all the required python libraries.
You can rerun this without any issues.
Enabling virtual env
development
Staring Local Development
pushed config
DB Init
DB Init complete
Create app complete
-------------- celery@uma v5.2.3 (dawn-chorus)
-- ******* ---- Linux-5.4.0-99-generic-x86_64-with-Ubuntu-18.04-bionic 2022-02-23 23:31:39
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app: Application Jobs:0x7fbfe915cb90
- ** ---------- .> transport: redis://localhost:6379/1
- ** ---------- .> results: redis://localhost:6379/2
- *** --- * --- .> concurrency: 4 (prefork)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
-------------- [queues]
.> celery exchange=celery(direct) key=celery
[tasks]
. application.tasks.hello_world
. application.tasks.print_current_time_job
[2022-02-23 23:31:39,383: INFO/MainProcess] Connected to redis://localhost:6379/1
[2022-02-23 23:31:39,389: INFO/MainProcess] mingle: searching for neighbors
[2022-02-23 23:31:40,414: INFO/MainProcess] mingle: all alone
[2022-02-23 23:31:40,439: INFO/MainProcess] celery@uma ready.


thej@uma: ~/code/work/flask-push/code
File Edit View Search Terminal Tabs Help
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gevent/hub.py", line 159, in sleep
waiter.get()
File "src/gevent/_waiter.py", line 143, in gevent._gevent_c_waiter.Waiter.get
File "src/gevent/_waiter.py", line 154, in gevent._gevent_c_waiter.Waiter.get
File "src/gevent/_greenlet_primitives.py", line 61, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_greenlet_primitives.py", line 61, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_greenlet_primitives.py", line 65, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_gevent_c_greenlet_primitives.pxd", line 35, in gevent._gevent_c_greenlet_primitives._greenlet_switch
File "src/gevent/greenlet.py", line 906, in gevent._gevent_cgreenlet.Greenlet.run
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gunicorn/workers/base.py", line 198, in handle_quit
sys.exit(0)
SystemExit: 0
2022-02-23T18:05:18Z <greenlet.greenlet object at 0x7fd54ea5d710 (otid=0x7fd5507da5f0) current active started main> failed with SystemExit
Traceback (most recent call last):
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gevent/monkey.py", line 893, in _shutdown
sleep()
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gevent/hub.py", line 159, in sleep
waiter.get()
File "src/gevent/_waiter.py", line 143, in gevent._gevent_c_waiter.Waiter.get
File "src/gevent/_waiter.py", line 154, in gevent._gevent_c_waiter.Waiter.get
File "src/gevent/_greenlet_primitives.py", line 61, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_greenlet_primitives.py", line 61, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_greenlet_primitives.py", line 65, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_gevent_c_greenlet_primitives.pxd", line 35, in gevent._gevent_c_greenlet_primitives._greenlet_switch
File "src/gevent/greenlet.py", line 906, in gevent._gevent_cgreenlet.Greenlet.run
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gunicorn/workers/base.py", line 198, in handle_quit
sys.exit(0)
SystemExit: 0
2022-02-23T18:05:18Z <greenlet.greenlet object at 0x7fd54ea5d710 (otid=0x7fd5507da5f0) current active started main> failed with SystemExit
[2022-02-23 23:35:18 +0530] [21501] [INFO] Shutting down: Master
thej@uma:~/code/work/flask-push/code$ sh local_gunicorn.sh
Welcome to to the setup. This will setup the local virtual env.
And then it will install all the required python libraries.
You can rerun this without any issues.
Enabling virtual env

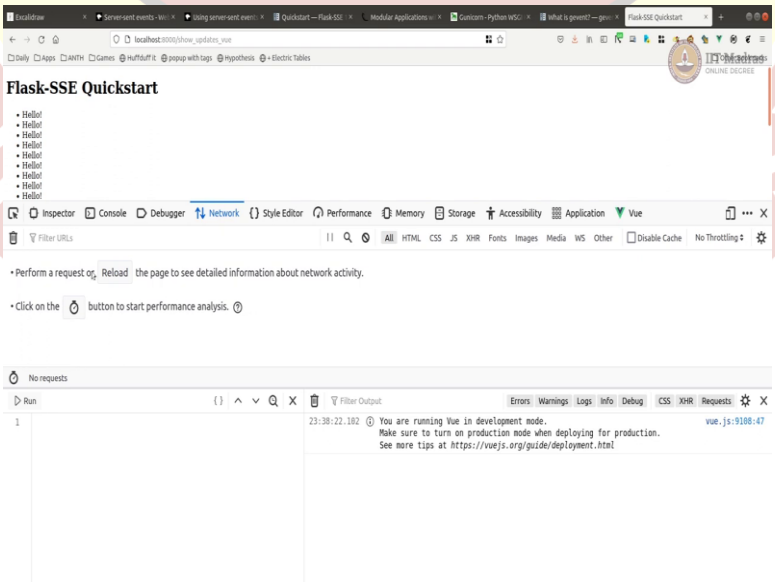

localhost:8000/show_updates_vue
Flask-SSE Quickstart
Flask-SSE Quickstart
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
localhost:8000/test_send_message
Message sent check the other browser!

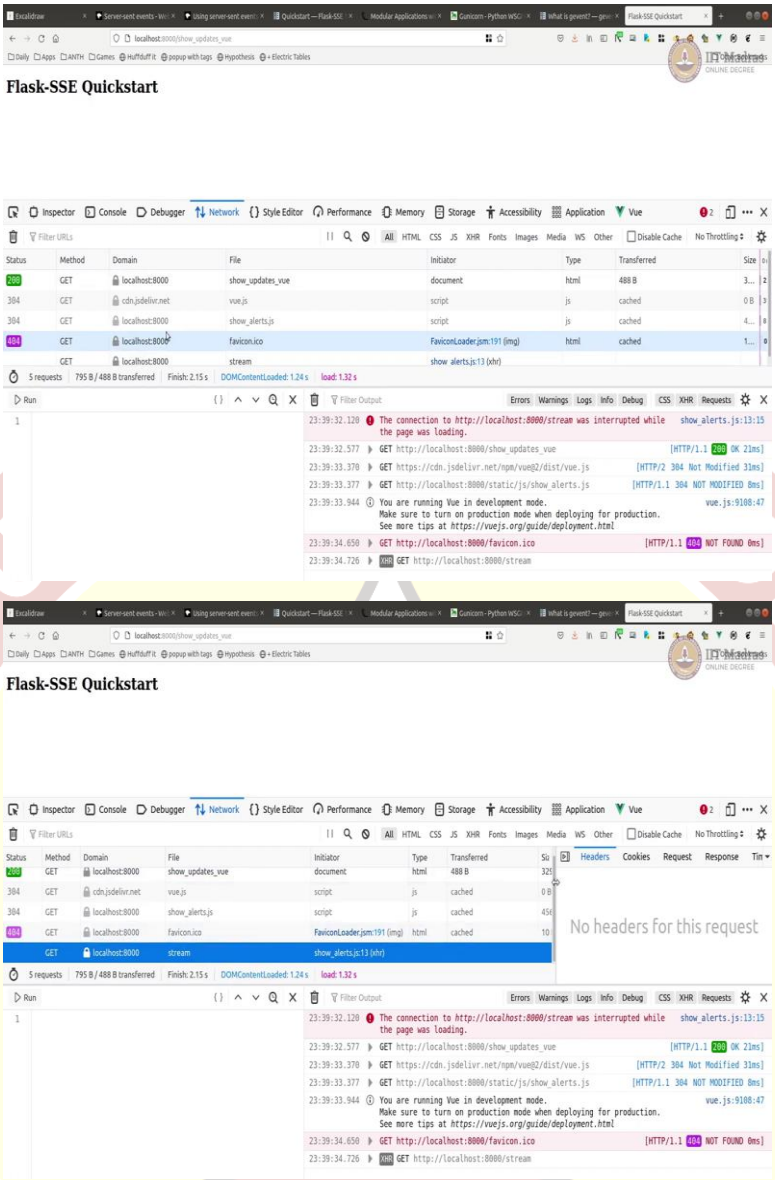


Let us start this again. This it running. This is running. This will start again. It will be Show update underscore vue. And now, if I have another browser, can you see? You can see how fast it gets. We will open one more. Open the same browser. Let us assume these are the two people who are watching the score.

Now, as you can see both of them. You see within this data thing. Here, you are not refreshing the page. You are not refreshing the page. You are not refreshing the page. As soon as the data is sent from the backend, it goes to all the browser (())(26:48). This is called server sent event.

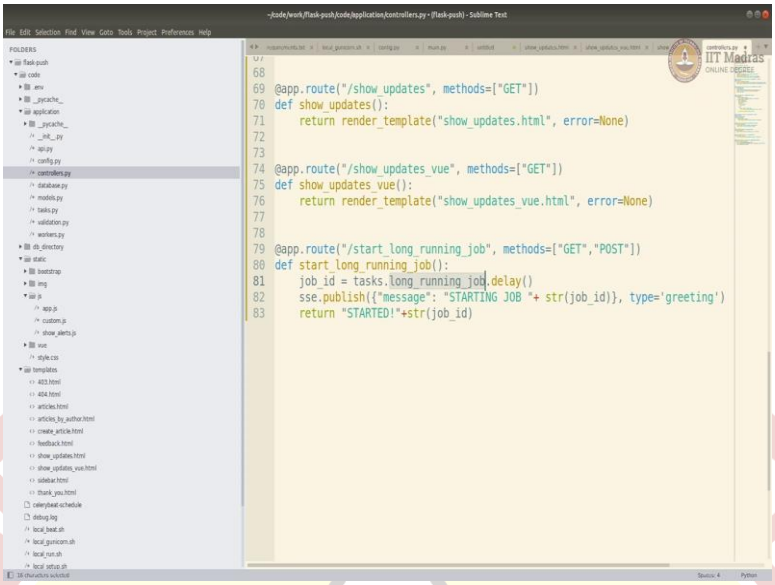
(Refer Slide Time: 26:51)





Now, if you are interested, you can click on network part of it and see the connection details. We can reload this. Here, is the stream. You can see that the file type is stream. And it is a get request. And you can see other details. Actually, there is not much details at this point because it may not made in, it has not made any call. But it is this is what is getting called every time when the server sends the push.

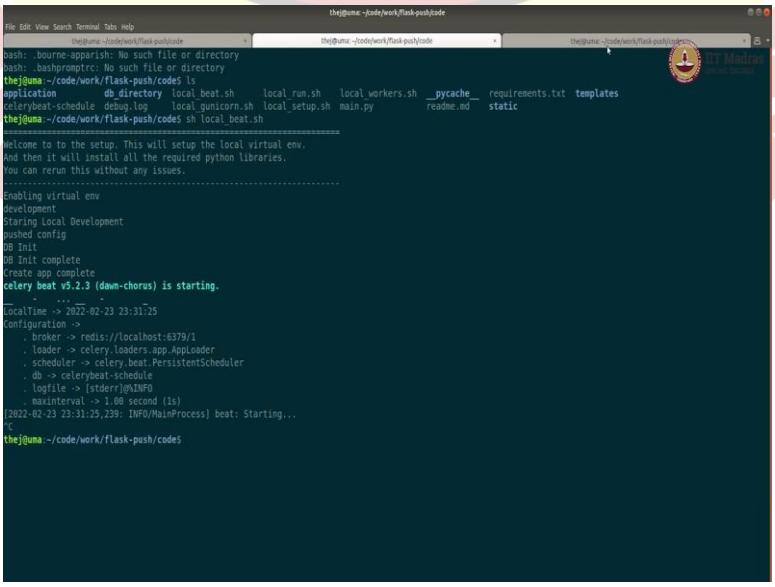
(Refer Slide Time: 27:41)

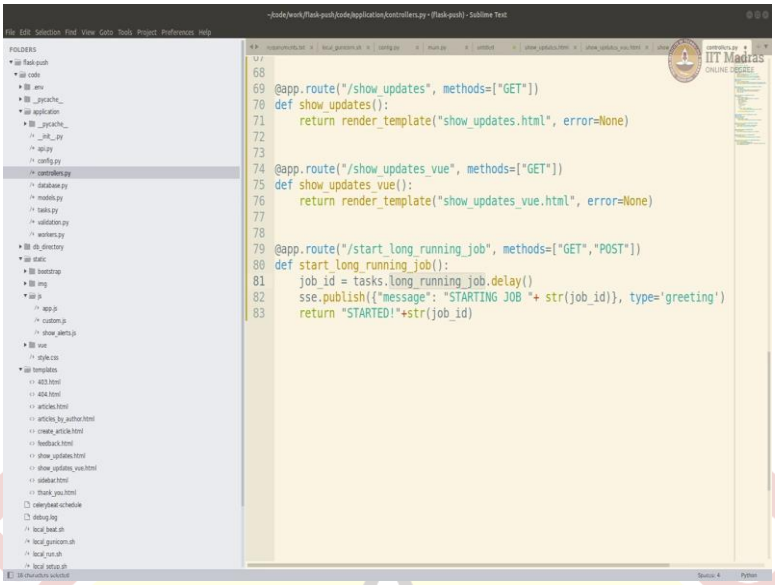


Now, we will go one more step in. I am just going to close these things. Pull this away. we are going to do one more step, instead of me calling test message, assume that I am going to run a long job. And the job wants to send an update. But that long job is triggered by the user. Let us say it is something like this. Starting a long running job.

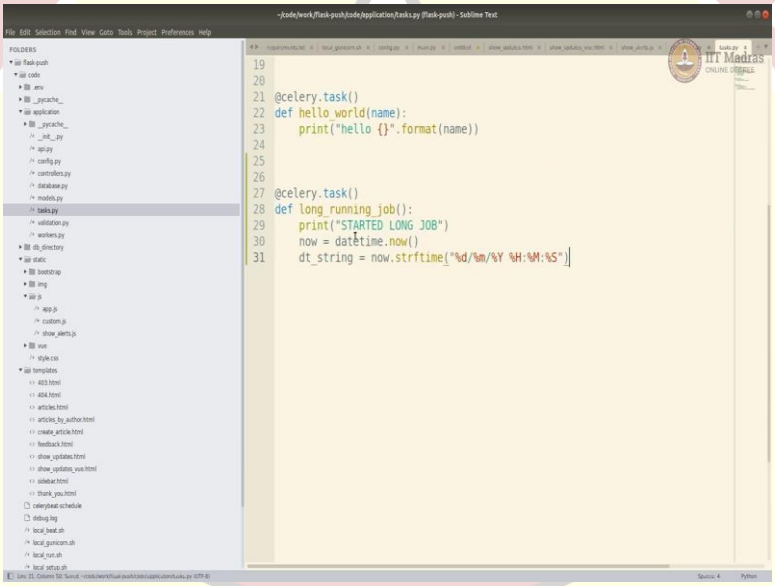
And the job is called long running job. And then I am just going to print, I am starting a job. And whenever that gets completed it gets completed. We are not waiting for the job to complete here. Then just returning job status. Now, in our long running job, you want to give updates to the end user saying what is the current status of the job, I am going to add a job.

(Refer Slide Time: 28:44)

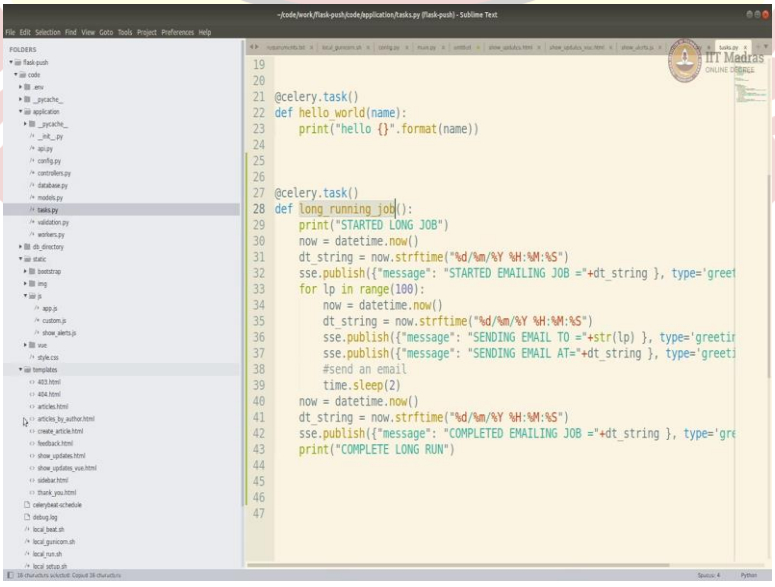




```
@app.route("/show_updates", methods=["GET"])
def show_updates():
    return render_template("show_updates.html", error=None)


@app.route("/show_updates_vue", methods=["GET"])
def show_updates_vue():
    return render_template("show_updates_vue.html", error=None)


@app.route("/start_long_running_job", methods=["GET","POST"])
def start_long_running_job():
    job_id = tasks.long_running_job.delay()
    sse.publish({"message": "STARTING JOB "+ str(job_id)}, type='greeting')
    return "STARTED!"+str(job_id)
```



```
@celery.task()
def hello_world(name):
    print("hello {}".format(name))



@celery.task()
def long_running_job():
    print("STARTED LONG JOB")
    now = datetime.now()
    dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
```



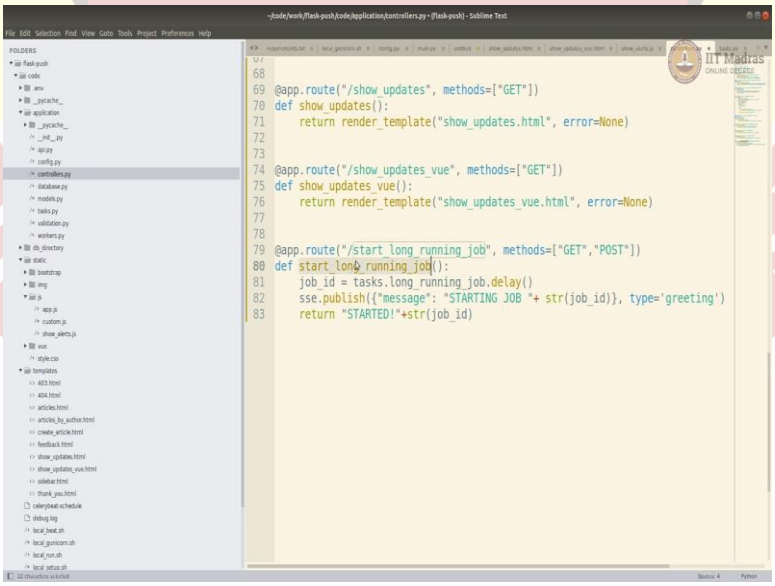
```
@celery.task()
def hello_world(name):
    print("hello {}".format(name))



@celery.task()
def long_running_job():
    print("STARTED LONG JOB")
    now = datetime.now()
    dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
    sse.publish({"message": "STARTED EMAILING JOB ="+dt_string }, type='greet
    for lp in range(100):
        now = datetime.now()
        dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
        sse.publish({"message": "SENDING EMAIL TO ="+str(lp) }, type='greetin
        sse.publish({"message": "SENDING EMAIL AT="+dt_string }, type='greeti
        #send an email
        time.sleep(2)
    now = datetime.now()
    dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
    sse.publish({"message": "COMPLETED EMAILING JOB ="+dt_string }, type='gre
    print("COMPLETE LONG RUN")
```

So, let us talk these things, we can start in a while long running job. So, let us add a task called long running job. In the tasks, I am going to add a long running job. And in the I am just, this is print, it will not be sent to the user, it has a string, then I am going to push this to the end user saying job has started. And let us say we will write whatever as part of the job, let us say file uploading, or, let us say your media conversion, you are converting mp3 into mp4 or whatever, that long running job.

Or you are sending mails to some hundreds of people. Let us think about that only. Let us say we are sending mails to 100s of people. So, I am just going to dummy it here saying, okay for people in 100, 100 people send an email. Now, sending an email will take some time. Let us assume it will make an external call, it probably takes a second. Because it makes an SMTP call, say sends it and then closes the connection and gets back. It might take way less than 1 second but for our clarity we will put 1 or 2 second just to see.

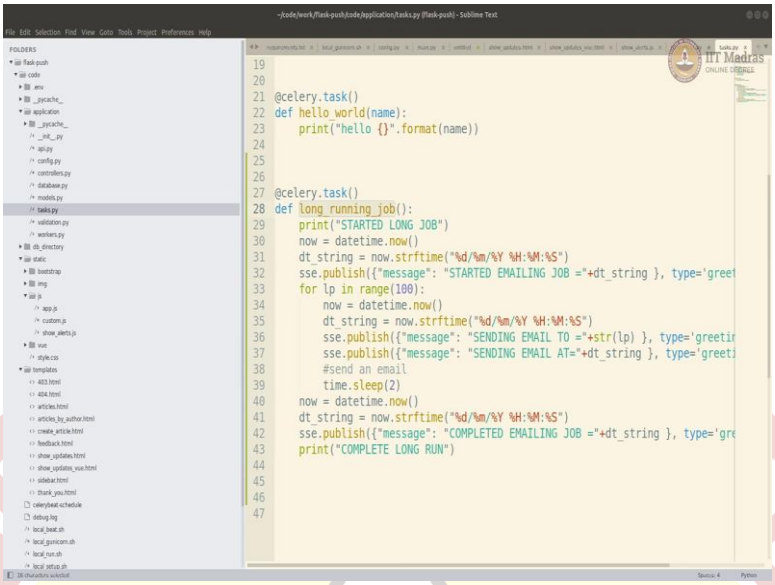
We will print something so that we know that the job is running and then we are going to get the update. Not going to print it here, not going to print it here. Running it will say, we will do it sending email dt string say we will print lp also, lp, AT. And then once everyone is sent, we will push job completion. We will send to the user, the job is complete. Now, so we printed, started emailing job, completed emailing job.

(Refer Slide Time: 32:04)

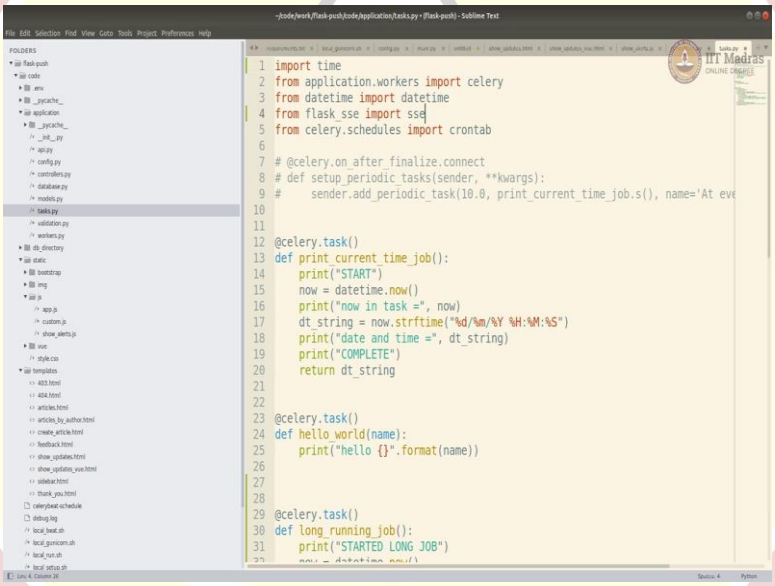


Now, this is getting called from our controller called start long running job, you could call it start emailing job, for example, but I leave that here.

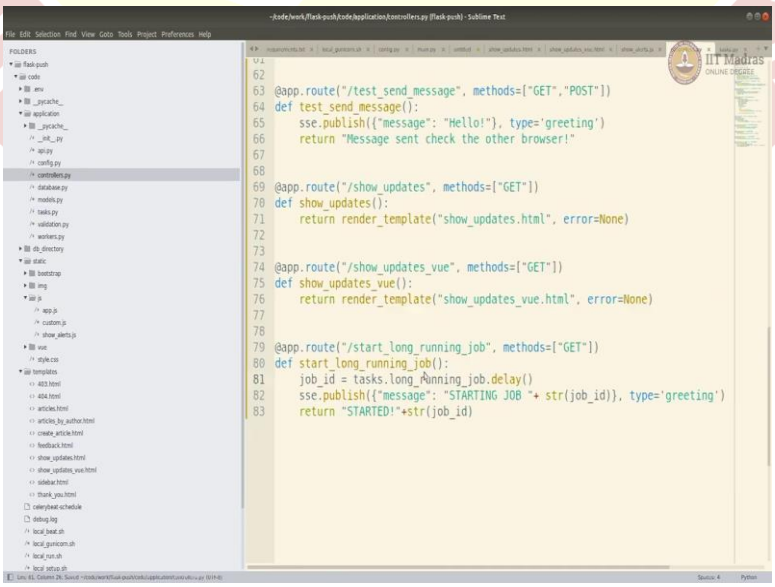
(Refer Slide Time: 32:14)



```
@celery.task()
def hello_world(name):
    print("hello {}".format(name))


@celery.task()
def long_running_job():
    print("STARTED LONG JOB")
    now = datetime.now()
    dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
    sse.publish({"message": "STARTED EMAILING JOB ="+dt_string }, type='greet
    for lp in range(100):
        now = datetime.now()
        dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
        sse.publish({"message": "SENDING EMAIL TO ="+str(lp) }, type='greetin
        sse.publish({"message": "SENDING EMAIL AT="+dt_string }, type='greeti
        #send an email
        time.sleep(2)
    now = datetime.now()
    dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
    sse.publish({"message": "COMPLETED EMAILING JOB ="+dt_string }, type='gre
    print("COMPLETE LONG RUN")
```



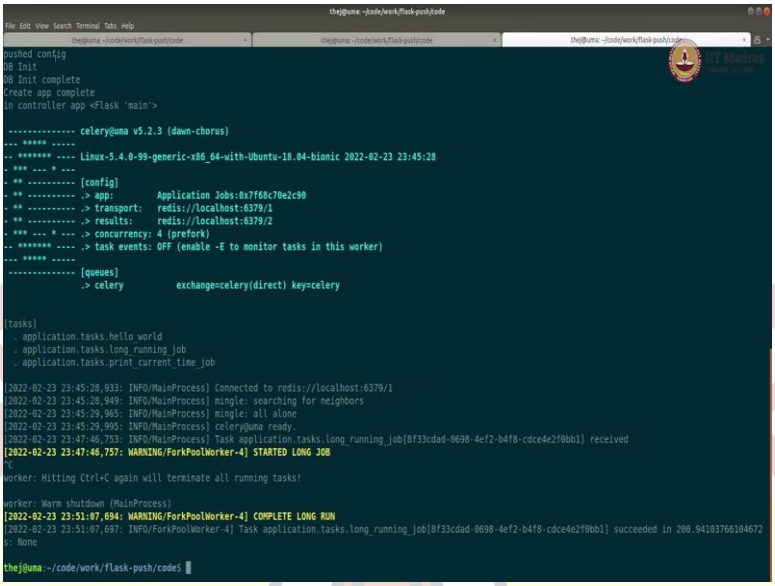
```
import time
from application.workers import celery
from datetime import datetime
from flask_sse import sse
from celery.schedules import crontab

# @celery.on_after_finalize.connect
# def setup_periodic_tasks(sender, **kwargs):
#     sender.add_periodic_task(10.0, print_current_time_job.s(), name='At eve


@celery.task()
def print_current_time_job():
    print("START")
    now = datetime.now()
    print("now in task =", now)
    dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
    print("date and time =", dt_string)
    print("COMPLETE")
    return dt_string


@celery.task()
def hello_world(name):
    print("hello {}".format(name))


@celery.task()
def long_running_job():
    print("STARTED LONG JOB")
```

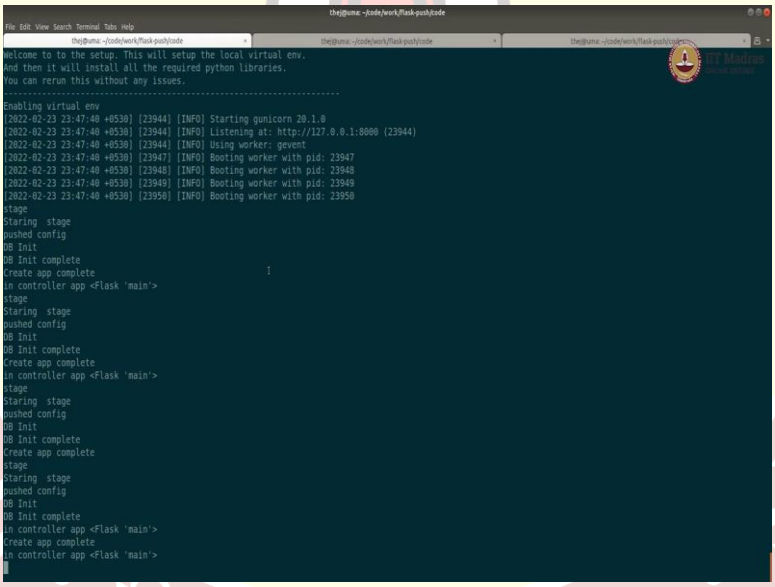


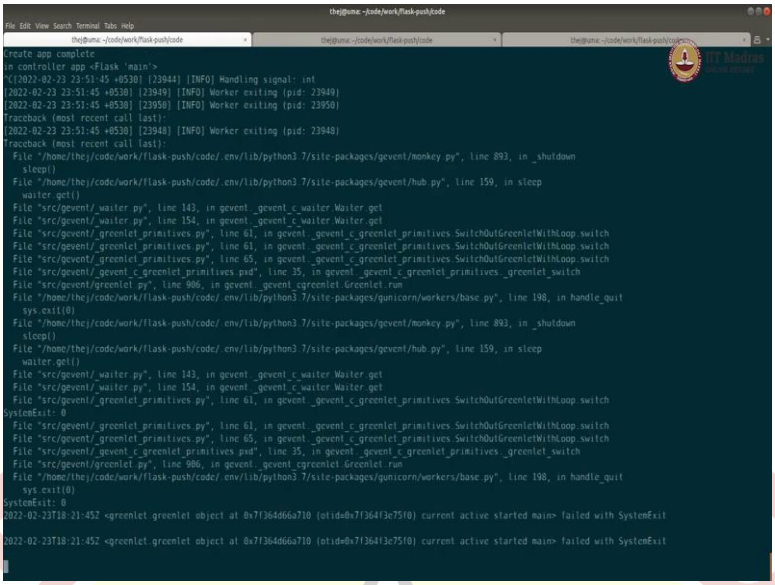
```
@app.route("/test_send_message", methods=["GET","POST"])
def test_send_message():
    sse.publish({"message": "Hello!"}, type='greeting')
    return "Message sent check the other browser!"


@app.route("/show_updates", methods=["GET"])
def show_updates():
    return render_template("show_updates.html", error=None)


@app.route("/show_updates_vue", methods=["GET"])
def show_updates_vue():
    return render_template("show_updates_vue.html", error=None)


@app.route("/start_long_running_job", methods=["GET"])
def start_long_running_job():
    job_id = tasks.long_running_job.delay()
    sse.publish({"message": "STARTING JOB "+ str(job_id)}, type='greeting')
    return "STARTED!"+str(job_id)
```

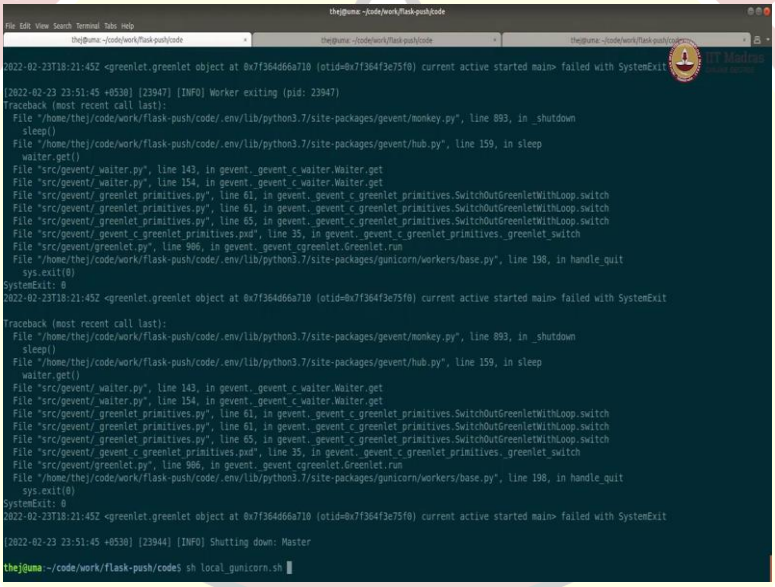
Now, we had some import here. So, I am going to do that import time, and also import sse, go down. It is seems ready.

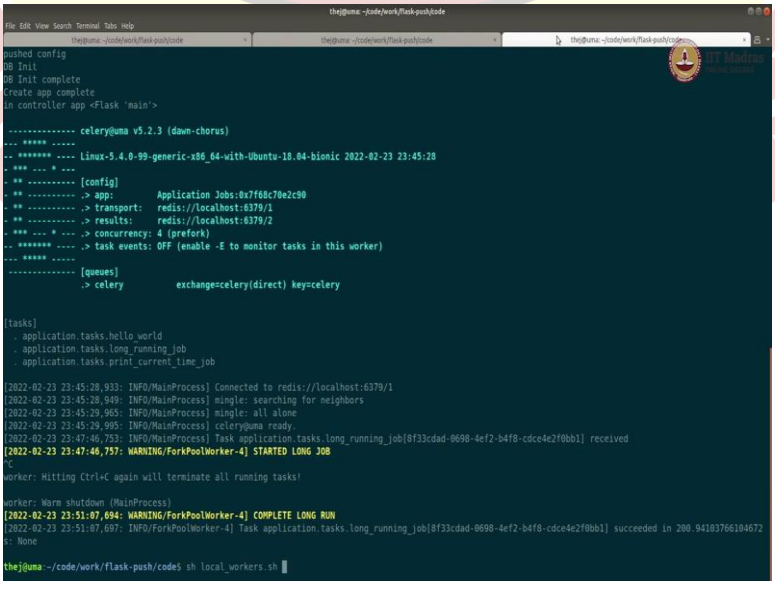
(Refer Slide Time: 32:43)

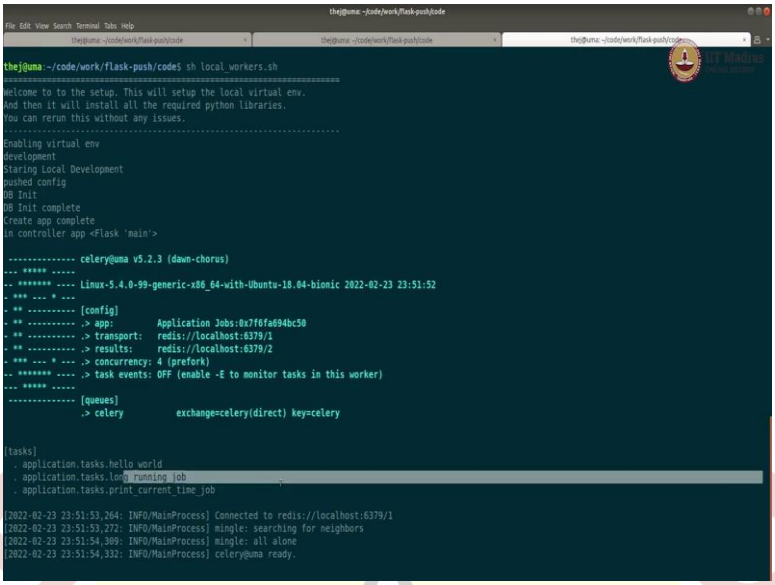





thej@uma: ~/code/work/flask-push/code
File Edit View Search Terminal Tabs Help
thej@uma: ~/code/work/flask-push/code
thej@uma: ~/code/work/flask-push/code
thej@uma: ~/code/work/flask-push/code
Create app complete
in controller app <Flask 'main'>
^C[2022-02-23 23:51:45 +0530] [23944] [INFO] Handling signal: int
[2022-02-23 23:51:45 +0530] [23949] [INFO] Worker exiting (pid: 23949)
[2022-02-23 23:51:45 +0530] [23950] [INFO] Worker exiting (pid: 23950)
Traceback (most recent call last):
[2022-02-23 23:51:45 +0530] [23948] [INFO] Worker exiting (pid: 23948)
Traceback (most recent call last):
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gevent/monkey.py", line 893, in _shutdown
sleep()
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gevent/hub.py", line 159, in sleep
waiter.get()
File "src/gevent/_waiter.py", line 143, in gevent._gevent_c_waiter.Waiter.get
File "src/gevent/_waiter.py", line 154, in gevent._gevent_c_waiter.Waiter.get
File "src/gevent/_greenlet_primitives.py", line 61, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_greenlet_primitives.py", line 61, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_greenlet_primitives.py", line 65, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_gevent_c_greenlet_primitives.pxd", line 35, in gevent._gevent_c_greenlet_primitives._greenlet_switch
File "src/gevent/greenlet.py", line 906, in gevent._gevent_cgreenlet.Greenlet.run
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gunicorn/workers/base.py", line 198, in handle_quit
sys.exit(0)
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gevent/monkey.py", line 893, in _shutdown
sleep()
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gevent/hub.py", line 159, in sleep
waiter.get()
File "src/gevent/_waiter.py", line 143, in gevent._gevent_c_waiter.Waiter.get
File "src/gevent/_waiter.py", line 154, in gevent._gevent_c_waiter.Waiter.get
File "src/gevent/_greenlet_primitives.py", line 61, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
SystemExit: 0
File "src/gevent/_greenlet_primitives.py", line 61, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_greenlet_primitives.py", line 65, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_gevent_c_greenlet_primitives.pxd", line 35, in gevent._gevent_c_greenlet_primitives._greenlet_switch
File "src/gevent/greenlet.py", line 906, in gevent._gevent_cgreenlet.Greenlet.run
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gunicorn/workers/base.py", line 198, in handle_quit
sys.exit(0)
SystemExit: 0
2022-02-23T18:21:45Z <greenlet.greenlet object at 0x7f364d66a710 (otid=0x7f364f3e75f0) current active started main> failed with SystemExit
2022-02-23T18:21:45Z <greenlet.greenlet object at 0x7f364d66a710 (otid=0x7f364f3e75f0) current active started main> failed with SystemExit

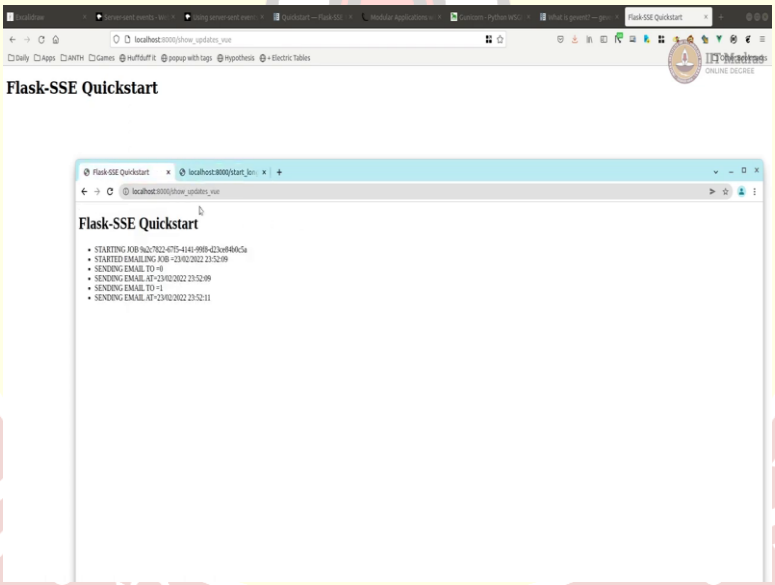

thej@uma: ~/code/work/flask-push/code
File Edit View Search Terminal Tabs Help
thej@uma: ~/code/work/flask-push/code
thej@uma: ~/code/work/flask-push/code
thej@uma: ~/code/work/flask-push/code
2022-02-23T18:21:45Z <greenlet.greenlet object at 0x7f364d66a710 (otid=0x7f364f3e75f0) current active started main> failed with SystemExit
[2022-02-23 23:51:45 +0530] [23947] [INFO] Worker exiting (pid: 23947)
Traceback (most recent call last):
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gevent/monkey.py", line 893, in _shutdown
sleep()
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gevent/hub.py", line 159, in sleep
waiter.get()
File "src/gevent/_waiter.py", line 143, in gevent._gevent_c_waiter.Waiter.get
File "src/gevent/_waiter.py", line 154, in gevent._gevent_c_waiter.Waiter.get
File "src/gevent/_greenlet_primitives.py", line 61, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_greenlet_primitives.py", line 61, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_greenlet_primitives.py", line 65, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_gevent_c_greenlet_primitives.pxd", line 35, in gevent._gevent_c_greenlet_primitives._greenlet_switch
File "src/gevent/greenlet.py", line 906, in gevent._gevent_cgreenlet.Greenlet.run
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gunicorn/workers/base.py", line 198, in handle_quit
sys.exit(0)
SystemExit: 0
2022-02-23T18:21:45Z <greenlet.greenlet object at 0x7f364d66a710 (otid=0x7f364f3e75f0) current active started main> failed with SystemExit
Traceback (most recent call last):
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gevent/monkey.py", line 893, in _shutdown
sleep()
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gevent/hub.py", line 159, in sleep
waiter.get()
File "src/gevent/_waiter.py", line 143, in gevent._gevent_c_waiter.Waiter.get
File "src/gevent/_waiter.py", line 154, in gevent._gevent_c_waiter.Waiter.get
File "src/gevent/_greenlet_primitives.py", line 61, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_greenlet_primitives.py", line 61, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_greenlet_primitives.py", line 65, in gevent._gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
File "src/gevent/_gevent_c_greenlet_primitives.pxd", line 35, in gevent._gevent_c_greenlet_primitives._greenlet_switch
File "src/gevent/greenlet.py", line 906, in gevent._gevent_cgreenlet.Greenlet.run
File "/home/thej/code/work/flask-push/code/.env/lib/python3.7/site-packages/gunicorn/workers/base.py", line 198, in handle_quit
sys.exit(0)
SystemExit: 0
2022-02-23T18:21:45Z <greenlet.greenlet object at 0x7f364d66a710 (otid=0x7f364f3e75f0) current active started main> failed with SystemExit
[2022-02-23 23:51:45 +0530] [23944] [INFO] Shutting down: Master
thej@uma:~/code/work/flask-push/code$ sh local_gunicorn.sh

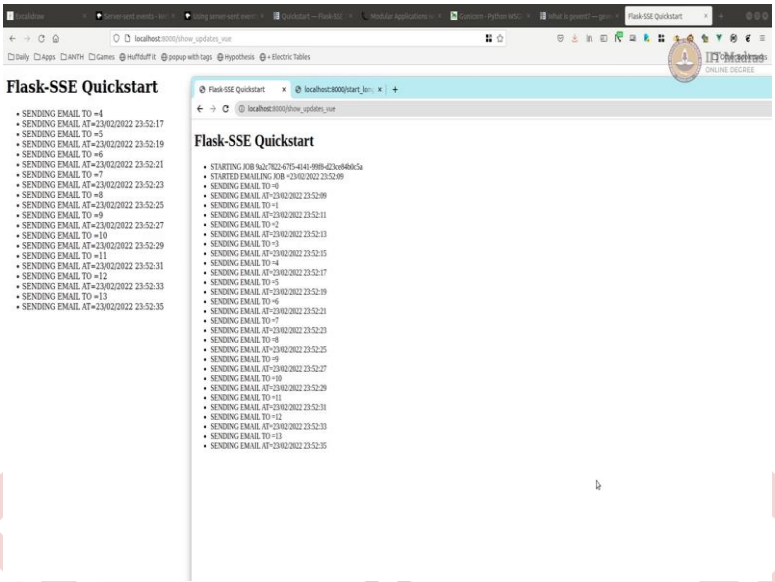

thej@uma: ~/code/work/flask-push/code
File Edit View Search Terminal Tabs Help
thej@uma: ~/code/work/flask-push/code
thej@uma: ~/code/work/flask-push/code
thej@uma: ~/code/work/flask-push/code
pushed config
DB Init
DB Init complete
Create app complete
in controller app <Flask 'main'>
-------------- celery@uma v5.2.3 (dawn-chorus)
--- ***** -----
-- ******* ---- Linux-5.4.0-99-generic-x86_64-with-Ubuntu-18.04-bionic 2022-02-23 23:45:28
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app: Application Jobs:0x7f68c70e2c90
- ** ---------- .> transport: redis://localhost:6379/1
- ** ---------- .> results: redis://localhost:6379/2
- *** --- * --- .> concurrency: 4 (prefork)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
-------------- [queues]
.> celery exchange=celery(direct) key=celery
[tasks]
. application.tasks.hello_world
. application.tasks.long_running_job
. application.tasks.print_current_time_job
[2022-02-23 23:45:28,933: INFO/MainProcess] Connected to redis://localhost:6379/1
[2022-02-23 23:45:28,949: INFO/MainProcess] mingle: searching for neighbors
[2022-02-23 23:45:29,965: INFO/MainProcess] mingle: all alone
[2022-02-23 23:45:29,995: INFO/MainProcess] celery@uma ready.
[2022-02-23 23:47:46,753: INFO/MainProcess] Task application.tasks.long_running_job[8f33cdad-0698-4ef2-b4f8-cdce4e2f0bb1] received
[2022-02-23 23:47:46,757: WARNING/ForkPoolWorker-4] STARTED LONG JOB
^C
worker: Hitting Ctrl+C again will terminate all running tasks!
worker: Warm shutdown (MainProcess)
[2022-02-23 23:51:07,694: WARNING/ForkPoolWorker-4] COMPLETE LONG RUN
[2022-02-23 23:51:07,697: INFO/ForkPoolWorker-4] Task application.tasks.long_running_job[8f33cdad-0698-4ef2-b4f8-cdce4e2f0bb1] succeeded in 200.94103766104672
s: None
thej@uma:~/code/work/flask-push/code$ sh local_workers.sh

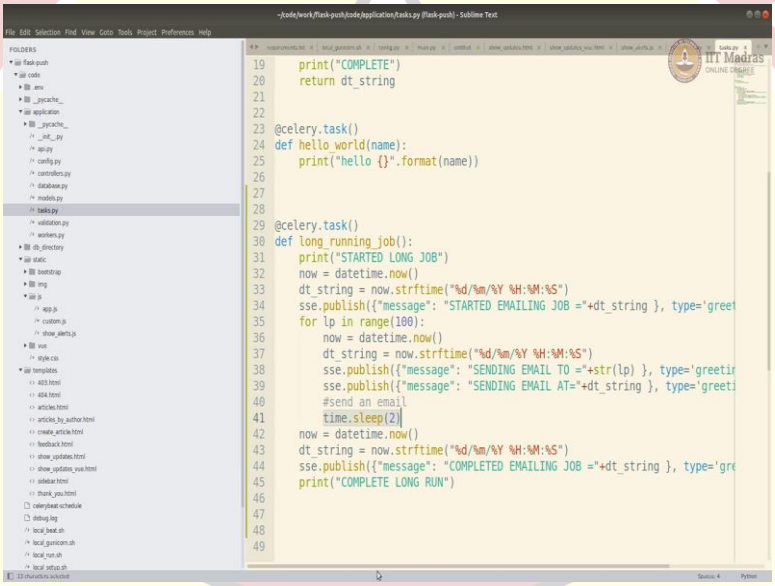


Going to start the controller main app again, I am going to start the jobs again, until long running job is there.

(Refer Slide Time: 32:59)

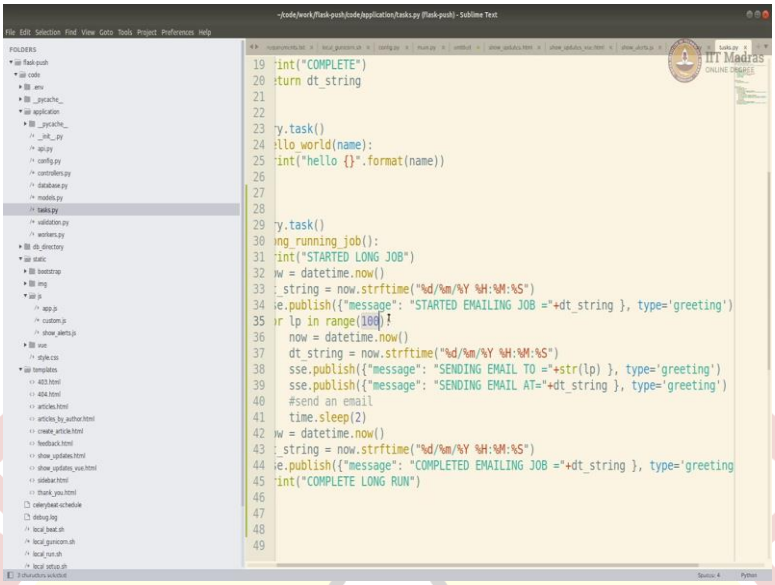


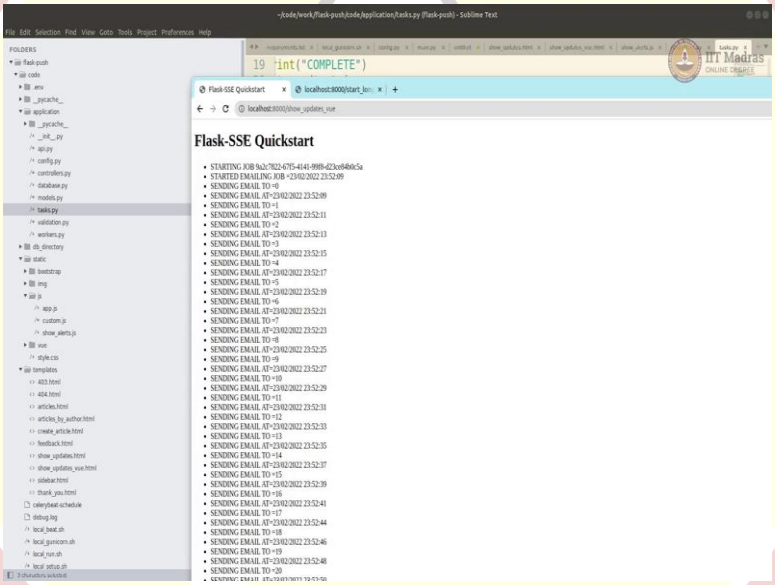


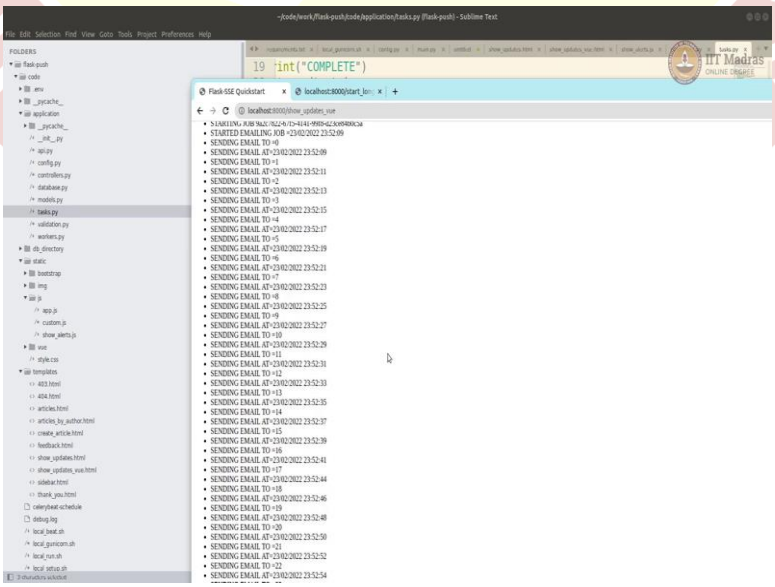


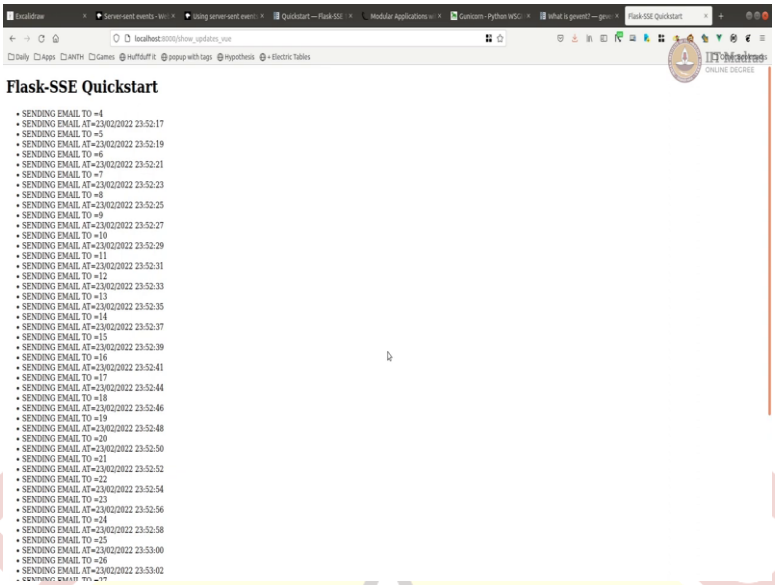
I am going to go, I am already on that page, going to have another page to sync to the same page so that we see update at both. Now I am going to start the long running job. The job has started here there is an update. Here I will just make a connection here so, this is actually getting pushed from the backend, as the job goes, you can see that the call is every 2 seconds 17, 19, 21, 23, 25, 27. So, if you see our code in the jobs you can see that we are sending it every 2 seconds because we are sleeping for 2 seconds.

(Refer Slide Time: 33:50)



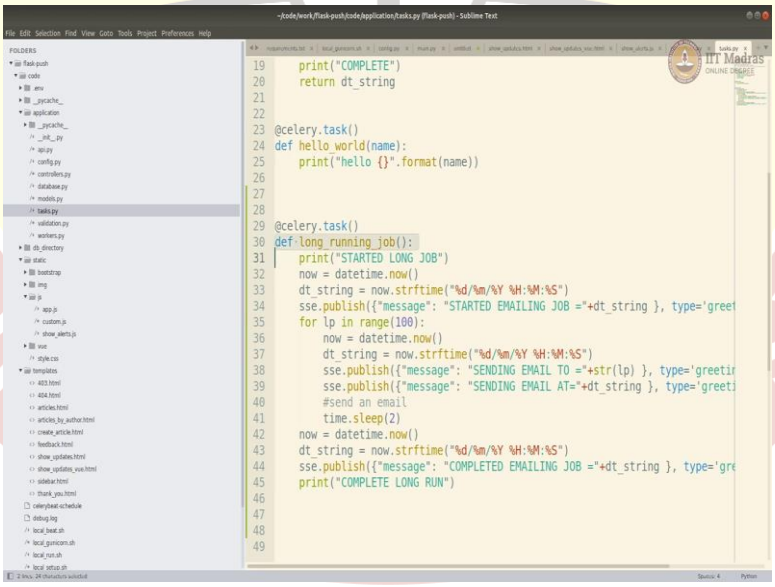


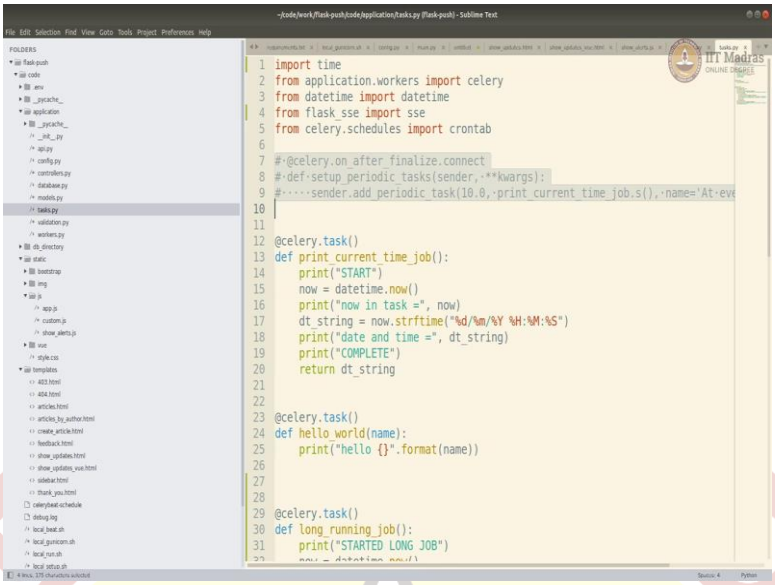


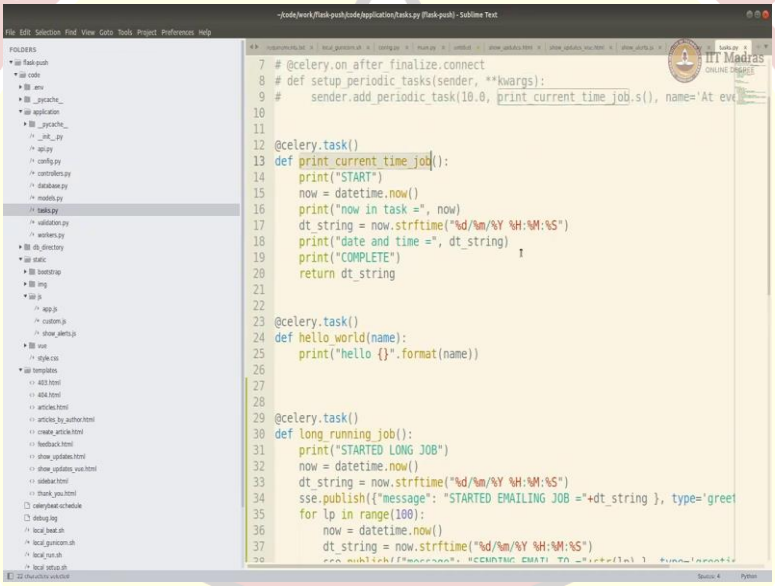


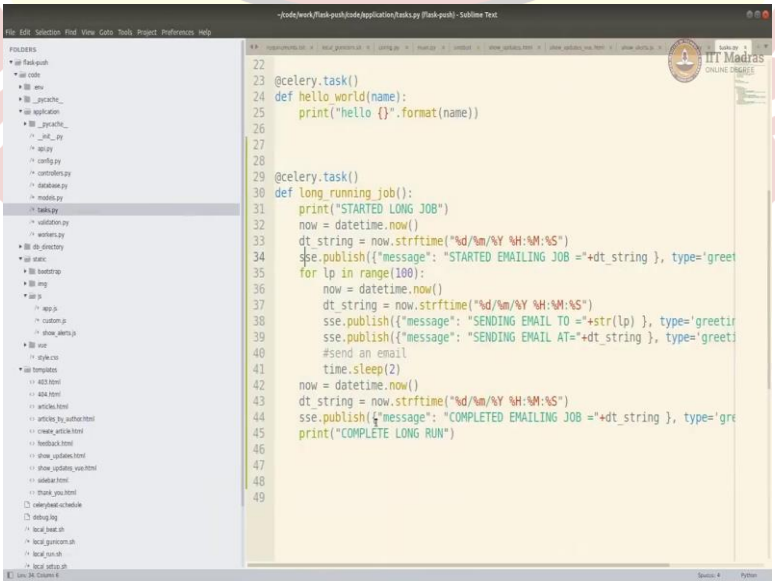
And then we are printing this and we are going to go up to 100 and send. Now like usual, without pulling it again and again is getting updates to instead of showing this you could even actually show your progress bar saying it to 100 percent or make a percentage of the total into that and show a progress bar that how many emails are being processed or async? So, this is easy to implement and do as well.

(Refer Slide Time: 34:29)



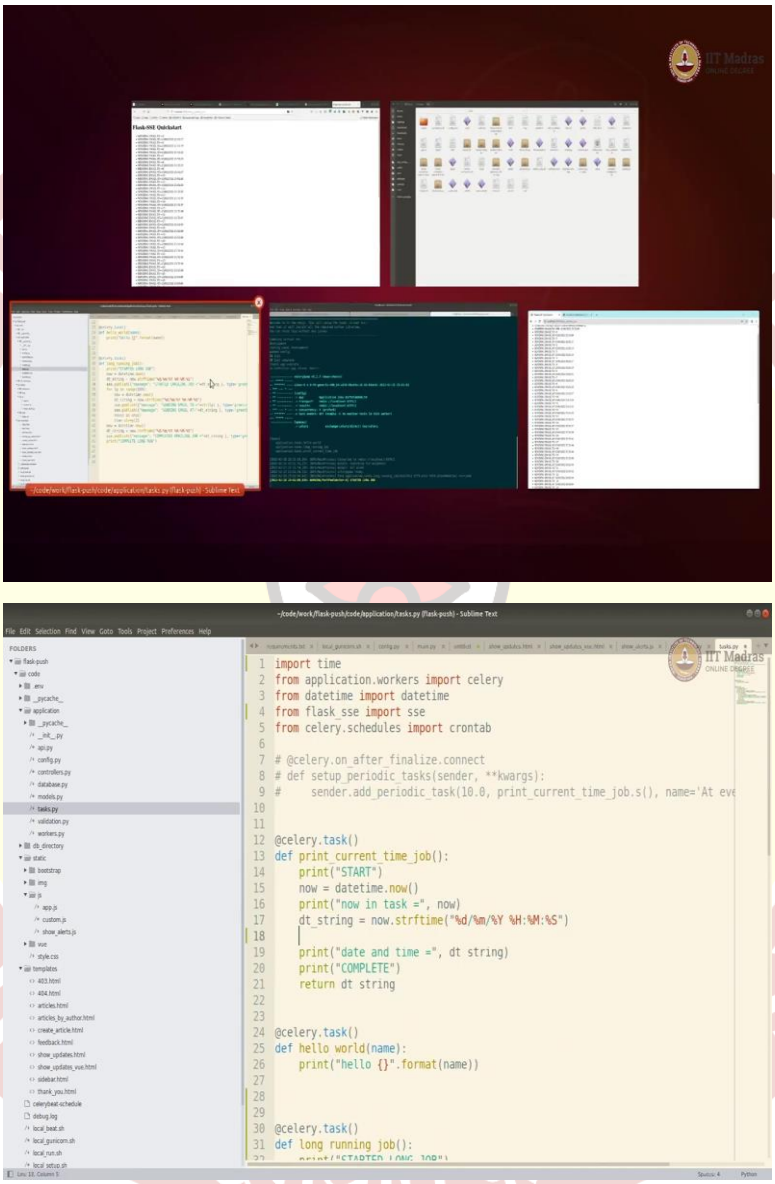

~/code/work/flask-push/code/application/tasks.py (flask-push) - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
flask-push
code
application
tasks.py
import time
from application.workers import celery
from datetime import datetime
from flask_sse import sse
from celery.schedules import crontab
#@celery.on_after_finalize.connect
#def setup_periodic_tasks(sender, **kwargs):
sender.add_periodic_task(10.0, print_current_time_job.s(), name='At eve
@celery.task()
def print_current_time_job():
print("START")
now = datetime.now()
print("now in task =", now)
dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
print("date and time =", dt_string)
print("COMPLETE")
return dt_string
@celery.task()
def hello_world(name):
print("hello {}".format(name))
@celery.task()
def long_running_job():
print("STARTED LONG JOB")
IIT Madras
ONLINE DEGREE

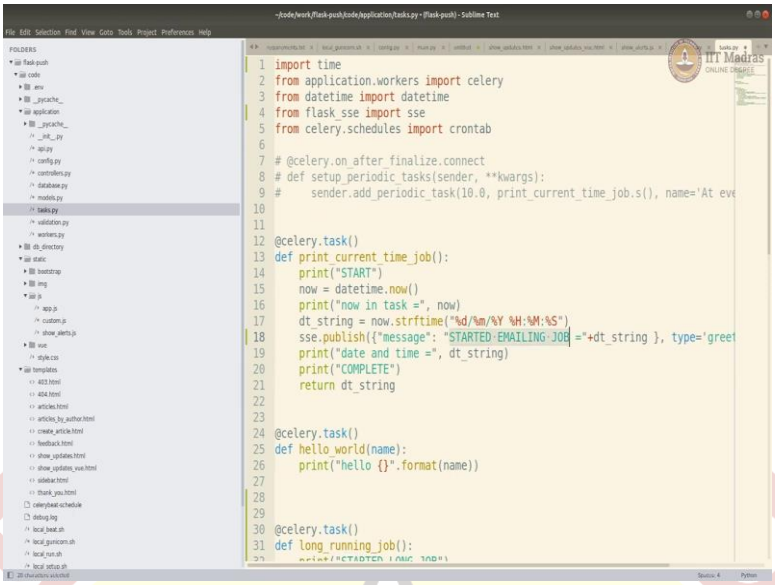

~/code/work/flask-push/code/application/tasks.py (flask-push) - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
flask-push
code
application
tasks.py
@celery.on_after_finalize.connect
def setup_periodic_tasks(sender, **kwargs):
sender.add_periodic_task(10.0, print_current_time_job.s(), name='At eve
@celery.task()
def print_current_time_job():
print("START")
now = datetime.now()
print("now in task =", now)
dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
print("date and time =", dt_string)
print("COMPLETE")
return dt_string
@celery.task()
def hello_world(name):
print("hello {}".format(name))
@celery.task()
def long_running_job():
print("STARTED LONG JOB")
now = datetime.now()
dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
sse.publish({"message": "STARTED EMAILING JOB ="+dt_string }, type='greet
for lp in range(100):
now = datetime.now()
dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
IIT Madras
ONLINE DEGREE

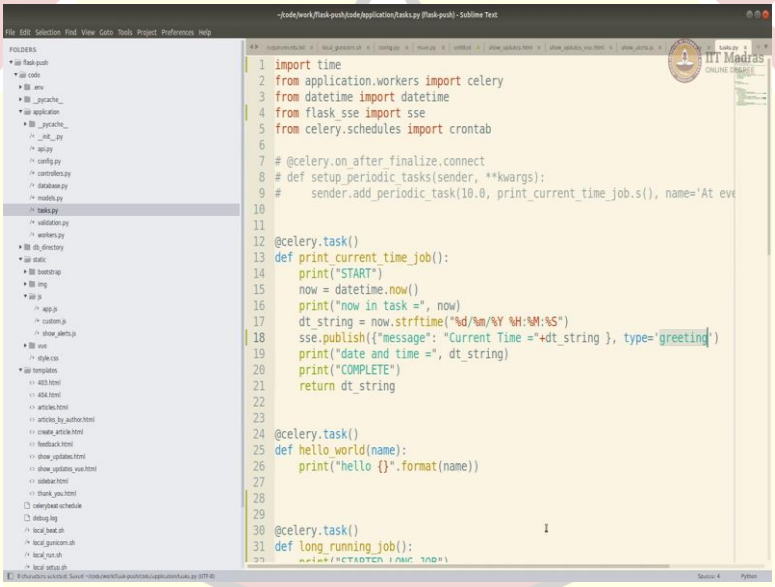

~/code/work/flask-push/code/application/tasks.py (flask-push) - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
flask-push
code
application
tasks.py
@celery.task()
def hello_world(name):
print("hello {}".format(name))
@celery.task()
def long_running_job():
print("STARTED LONG JOB")
now = datetime.now()
dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
sse.publish({"message": "STARTED EMAILING JOB ="+dt_string }, type='greet
for lp in range(100):
now = datetime.now()
dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
sse.publish({"message": "SENDING EMAIL TO ="+str(lp) }, type='greetin
sse.publish({"message": "SENDING EMAIL AT="+dt_string }, type='greeti
#send an email
time.sleep(2)
now = datetime.now()
dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
sse.publish({"message": "COMPLETED EMAILING JOB ="+dt_string }, type='gre
print("COMPLETE LONG RUN")
IIT Madras
ONLINE DEGREE

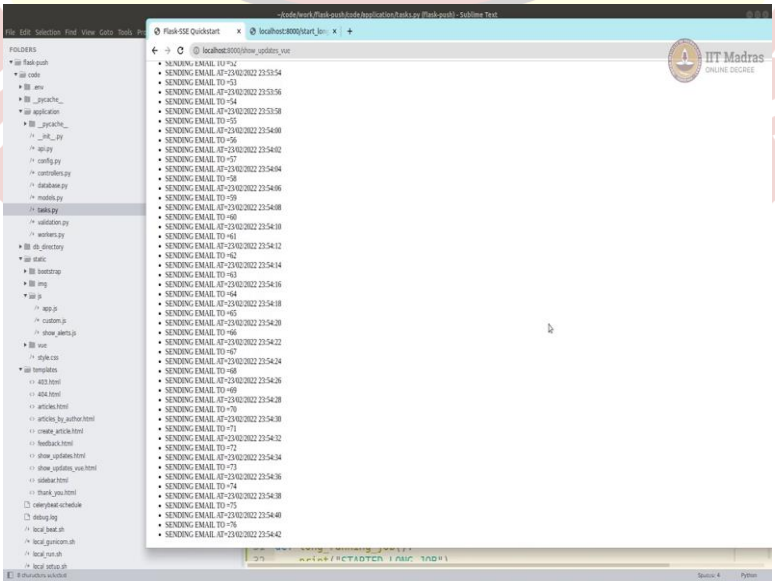
Now just like how we triggered this bio by a use a trigger we could also trigger this using a timer like we saw last time you could trigger I will add a simple thing there is a print current time job along with printing we could also send it to the front end. Instead of greeting we will make it as some other type.

(Refer Slide Time: 34:57)



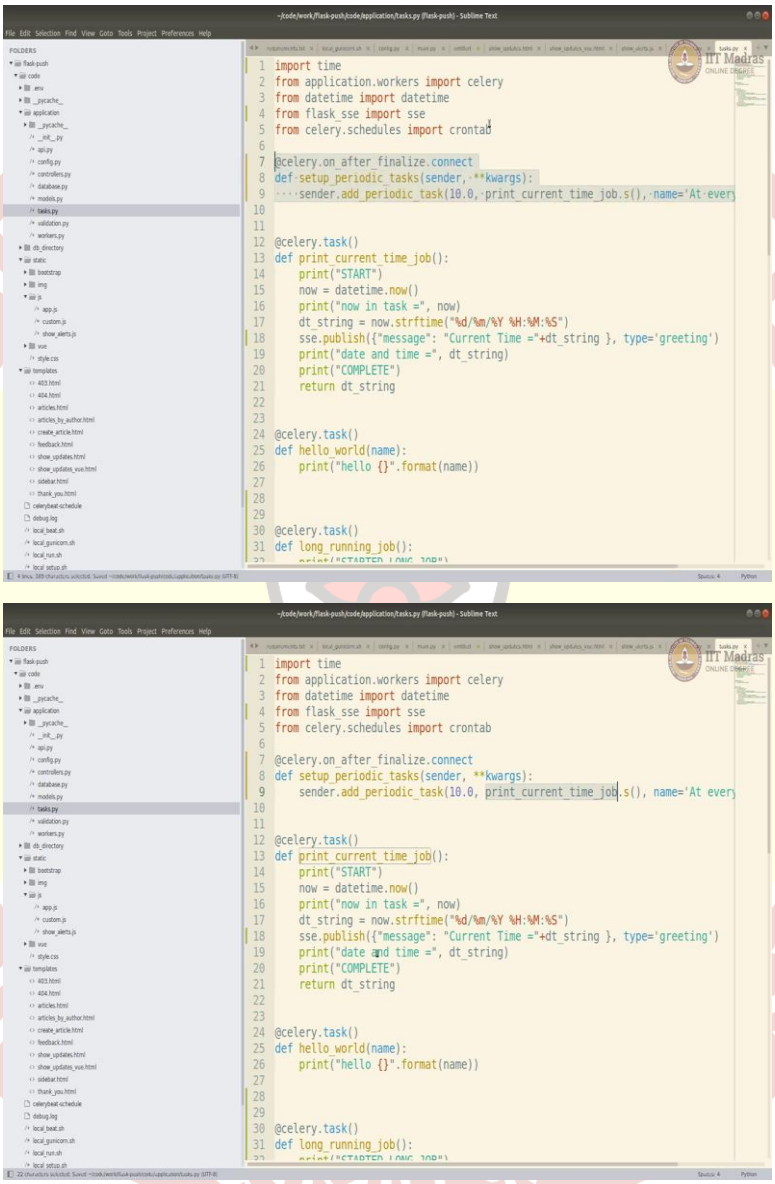

~/code/work/flask-push/code/application/tasks.py (flask-push) - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
flask-push
code
application
tasks.py
import time
from application.workers import celery
from datetime import datetime
from flask_sse import sse
from celery.schedules import crontab
# @celery.on_after_finalize.connect
# def setup_periodic_tasks(sender, **kwargs):
#   sender.add_periodic_task(10.0, print_current_time_job.s(), name='At eve
@celery.task()
def print_current_time_job():
print("START")
now = datetime.now()
print("now in task =", now)
dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
sse.publish({"message": "STARTED EMAILING JOB ="+dt_string }, type='greet
print("date and time =", dt_string)
print("COMPLETE")
return dt_string
@celery.task()
def hello_world(name):
print("hello {}".format(name))
@celery.task()
def long_running_job():
IIT Madras
ONLINE DEGREE

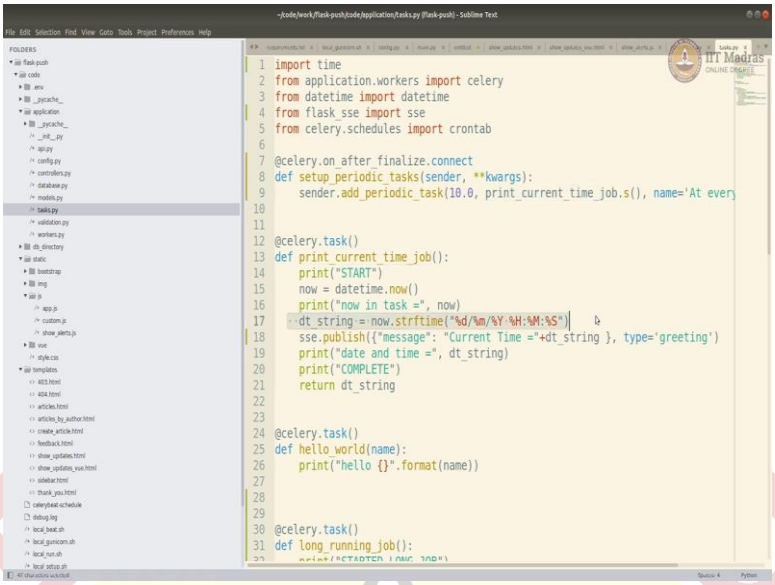

~/code/work/flask-push/code/application/tasks.py (flask-push) - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
flask-push
code
application
tasks.py
import time
from application.workers import celery
from datetime import datetime
from flask_sse import sse
from celery.schedules import crontab
# @celery.on_after_finalize.connect
# def setup_periodic_tasks(sender, **kwargs):
#   sender.add_periodic_task(10.0, print_current_time_job.s(), name='At eve
@celery.task()
def print_current_time_job():
print("START")
now = datetime.now()
print("now in task =", now)
dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
sse.publish({"message": "Current Time ="+dt_string }, type='greeting')
print("date and time =", dt_string)
print("COMPLETE")
return dt_string
@celery.task()
def hello_world(name):
print("hello {}".format(name))
@celery.task()
def long_running_job():
IIT Madras
ONLINE DEGREE

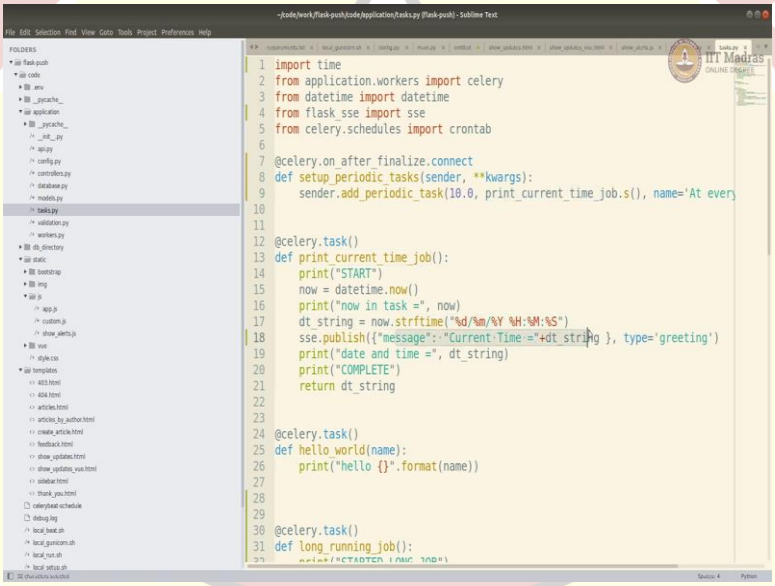

~/code/work/flask-push/code/application/tasks.py (flask-push) - Sublime Text
Flask-SSE Quickstart
localhost:8000/start_lon
localhost:8000/show_updates_vue
SENDING EMAIL TO =52
SENDING EMAIL AT=23/02/2022 23:53:54
SENDING EMAIL TO =53
SENDING EMAIL AT=23/02/2022 23:53:56
SENDING EMAIL TO =54
SENDING EMAIL AT=23/02/2022 23:53:58
SENDING EMAIL TO =55
SENDING EMAIL AT=23/02/2022 23:54:00
SENDING EMAIL TO =56
SENDING EMAIL AT=23/02/2022 23:54:02
SENDING EMAIL TO =57
SENDING EMAIL AT=23/02/2022 23:54:04
SENDING EMAIL TO =58
SENDING EMAIL AT=23/02/2022 23:54:06
SENDING EMAIL TO =59
SENDING EMAIL AT=23/02/2022 23:54:08
SENDING EMAIL TO =60
SENDING EMAIL AT=23/02/2022 23:54:10
SENDING EMAIL TO =61
SENDING EMAIL AT=23/02/2022 23:54:12
SENDING EMAIL TO =62
SENDING EMAIL AT=23/02/2022 23:54:14
SENDING EMAIL TO =63
SENDING EMAIL AT=23/02/2022 23:54:16
SENDING EMAIL TO =64
SENDING EMAIL AT=23/02/2022 23:54:18
SENDING EMAIL TO =65
SENDING EMAIL AT=23/02/2022 23:54:20
SENDING EMAIL TO =66
SENDING EMAIL AT=23/02/2022 23:54:22
SENDING EMAIL TO =67
SENDING EMAIL AT=23/02/2022 23:54:24
SENDING EMAIL TO =68
SENDING EMAIL AT=23/02/2022 23:54:26
SENDING EMAIL TO =69
SENDING EMAIL AT=23/02/2022 23:54:28
SENDING EMAIL TO =70
SENDING EMAIL AT=23/02/2022 23:54:30
SENDING EMAIL TO =71
SENDING EMAIL AT=23/02/2022 23:54:32
SENDING EMAIL TO =72
SENDING EMAIL AT=23/02/2022 23:54:34
SENDING EMAIL TO =73
SENDING EMAIL AT=23/02/2022 23:54:36
SENDING EMAIL TO =74
SENDING EMAIL AT=23/02/2022 23:54:38
SENDING EMAIL TO =75
SENDING EMAIL AT=23/02/2022 23:54:40
SENDING EMAIL TO =76
SENDING EMAIL AT=23/02/2022 23:54:42
IIT Madras
ONLINE DEGREE

This one be greeting and let it be greeting I think we will just be war there is this over is upload over. If I was a user, let us what I would have been checking 72 percent over over. 100 out of 100 users, 35:45 70 to 75. Let us wait for another 30 seconds for it to get over here.

(Refer Slide Time: 35:47)

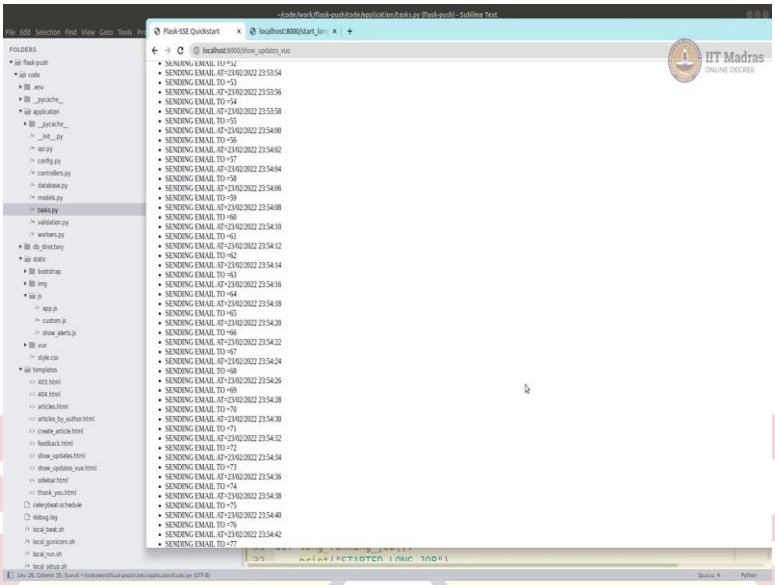


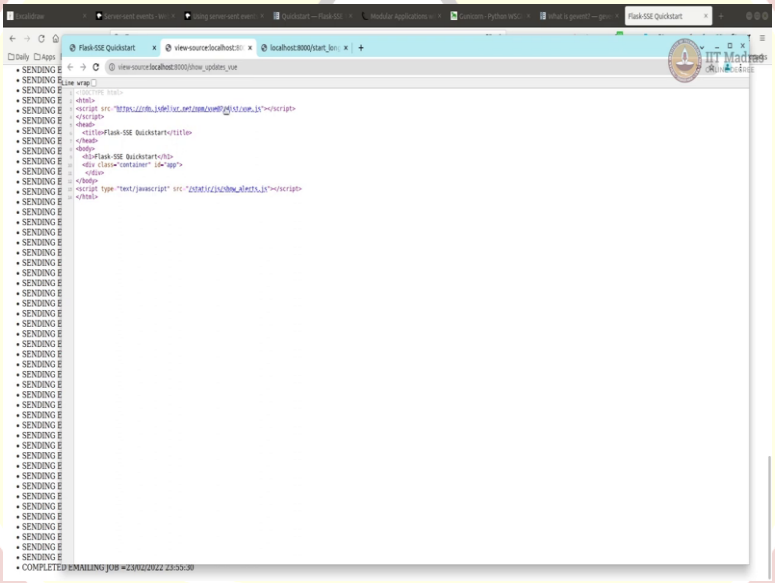


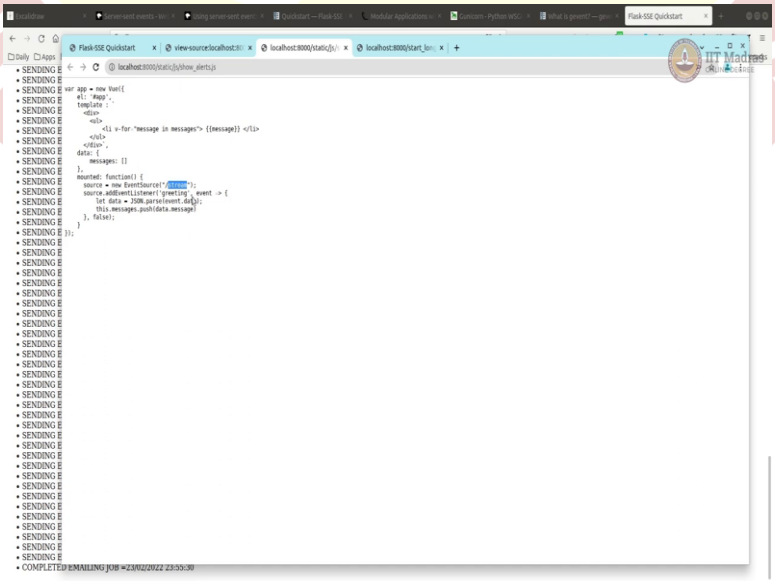


Here we are going to call setup a (())(35:53) job that calls this print timer. It also print and along with it sends to the whole user whoever is online and trying to watch this. So, we will do that once this gets over.

(Refer Slide Time: 36:09)

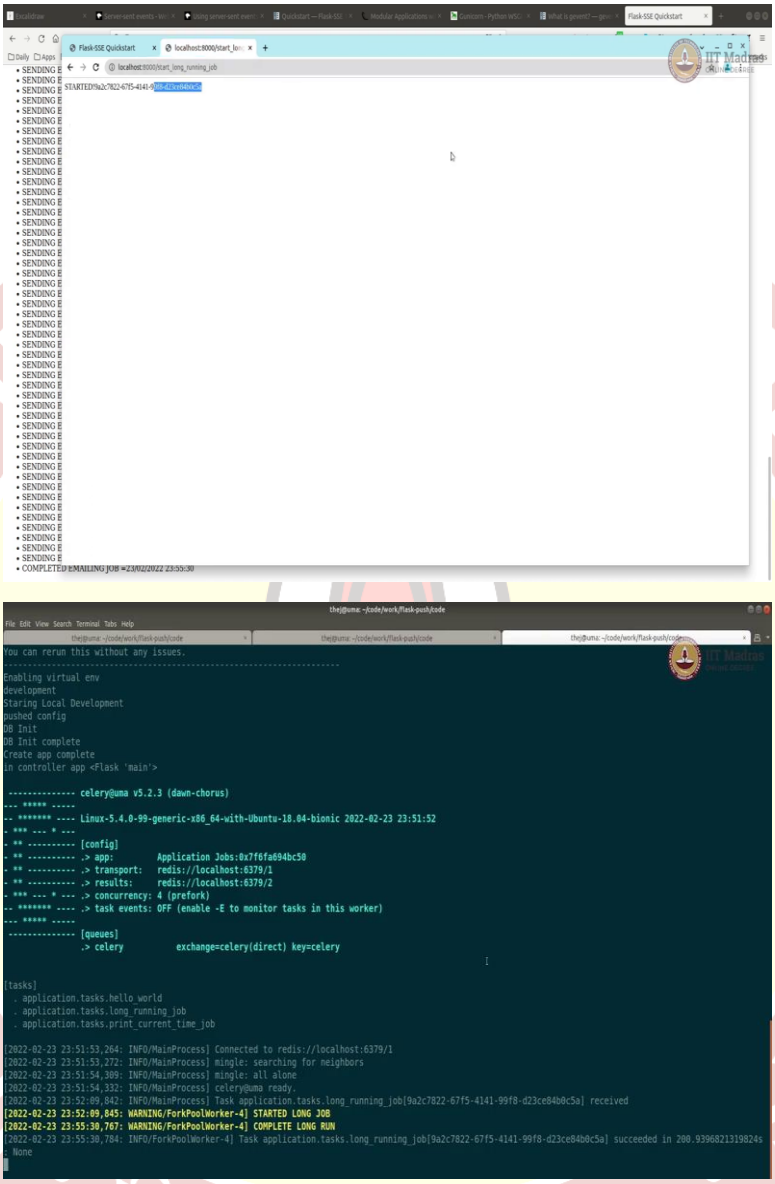


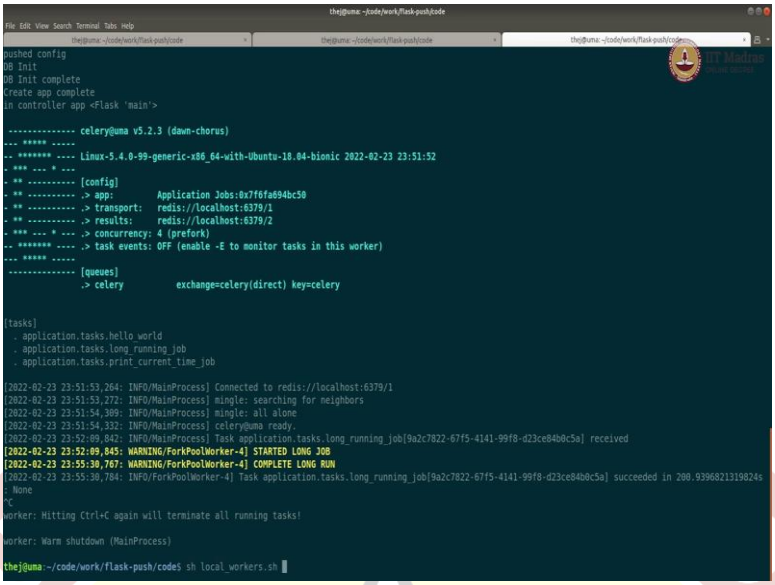


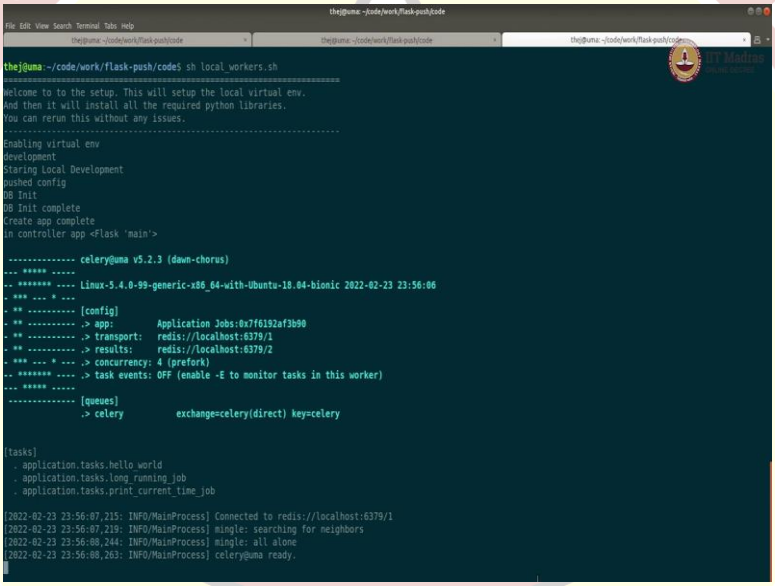


The job is completed. It is done whoever is subscripted. This stream has got it in the sense viewer ha subscripted this stream and this event has been getting the update.

(Refer Slide Time: 36:32)

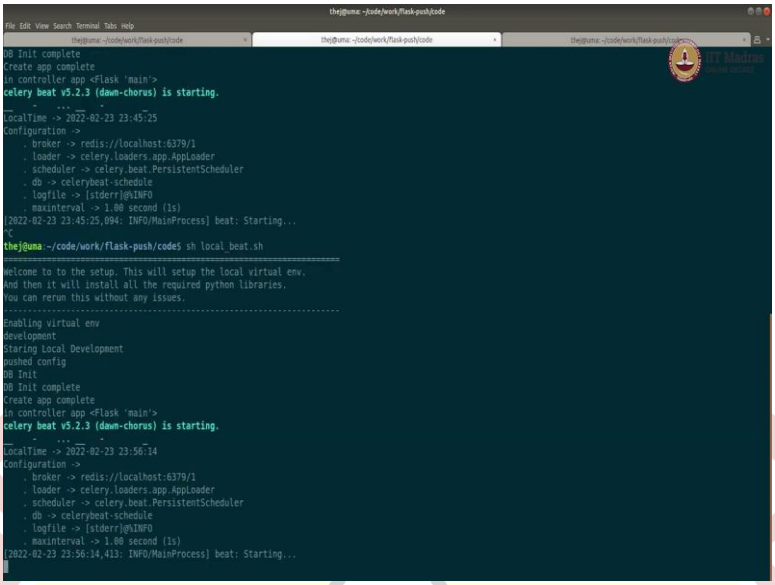


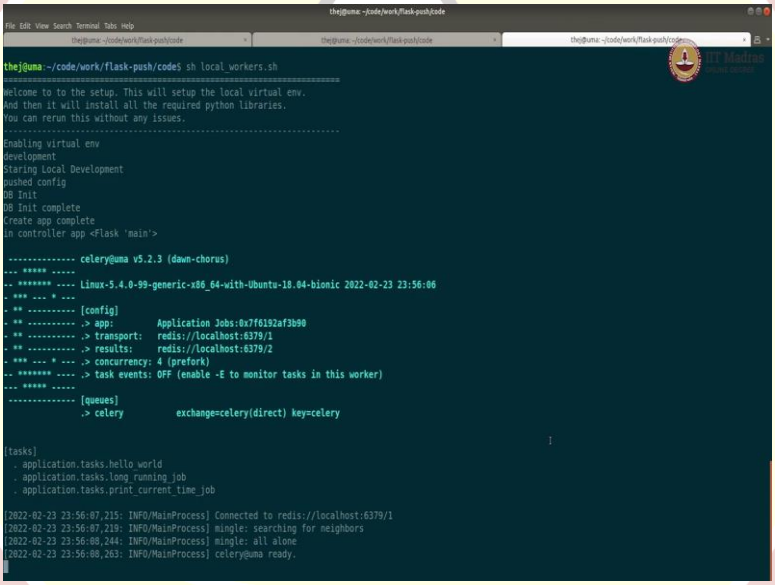


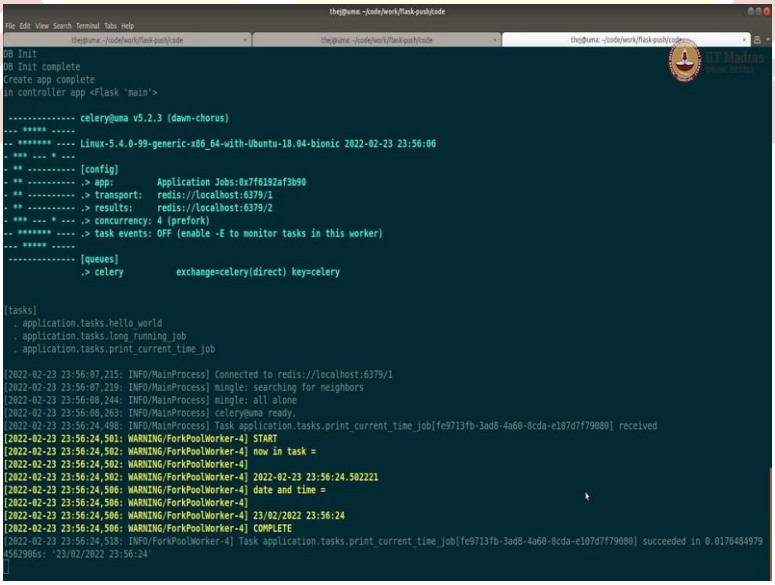


Now let us do one more. It is send just a time one.

(Refer Slide Time: 36:48)

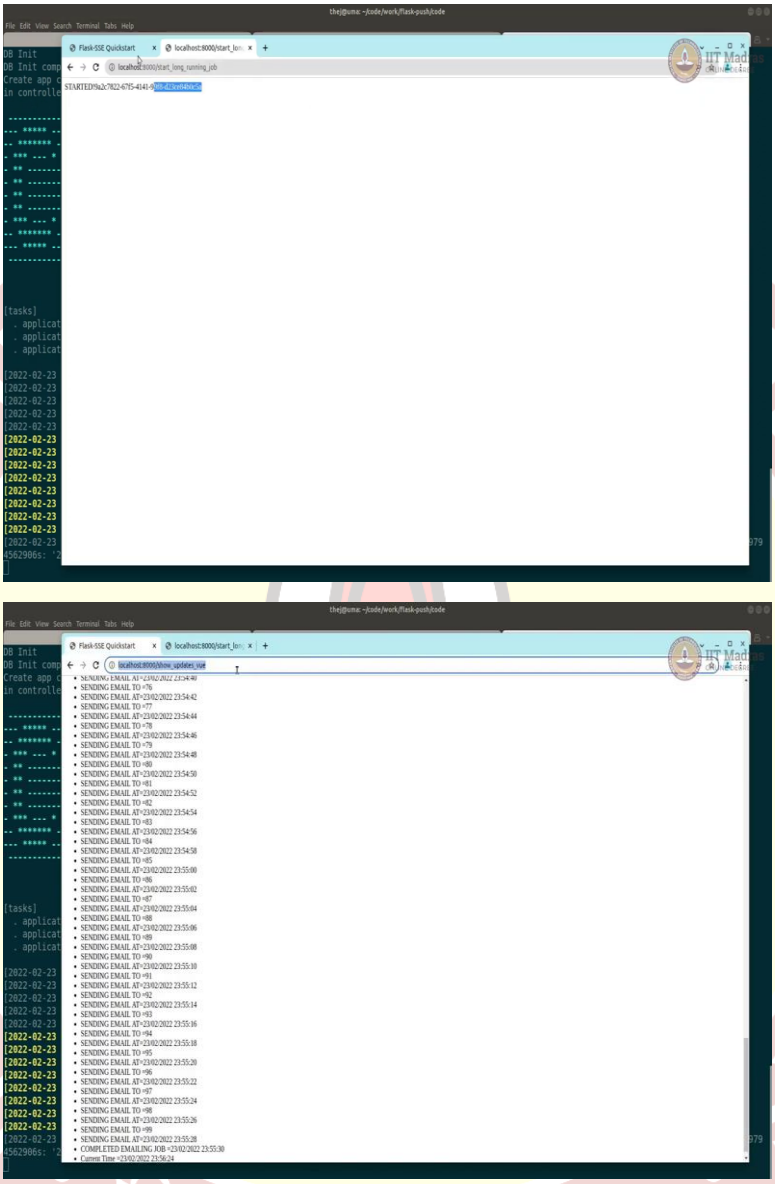


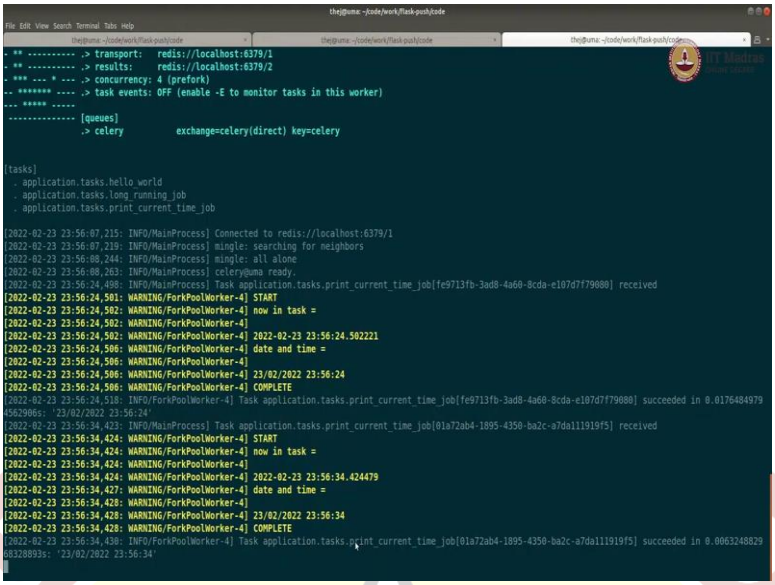


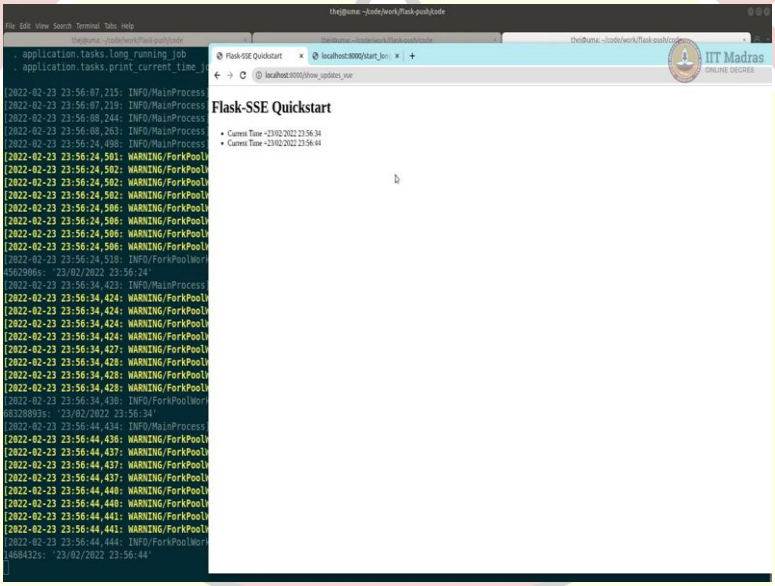


Let us start because there is a time when it needs to be it needs a timer. Long printing current it is getting sent.

(Refer Slide Time: 37:01)

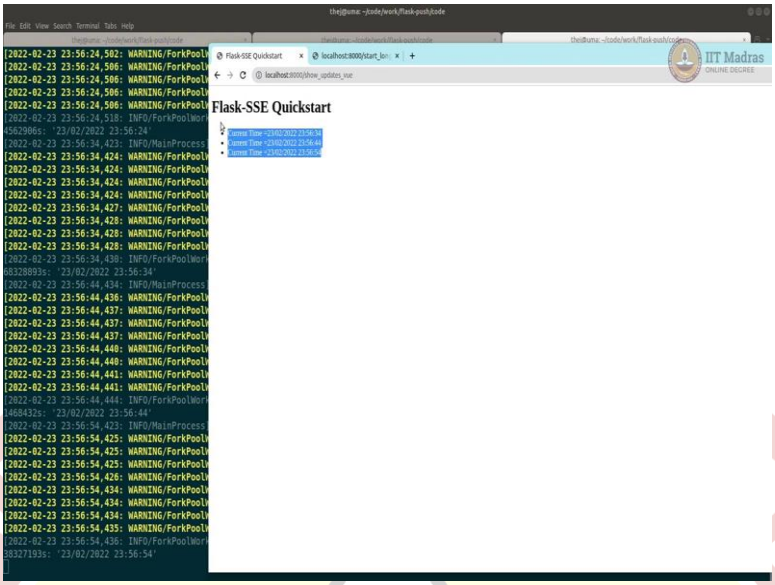


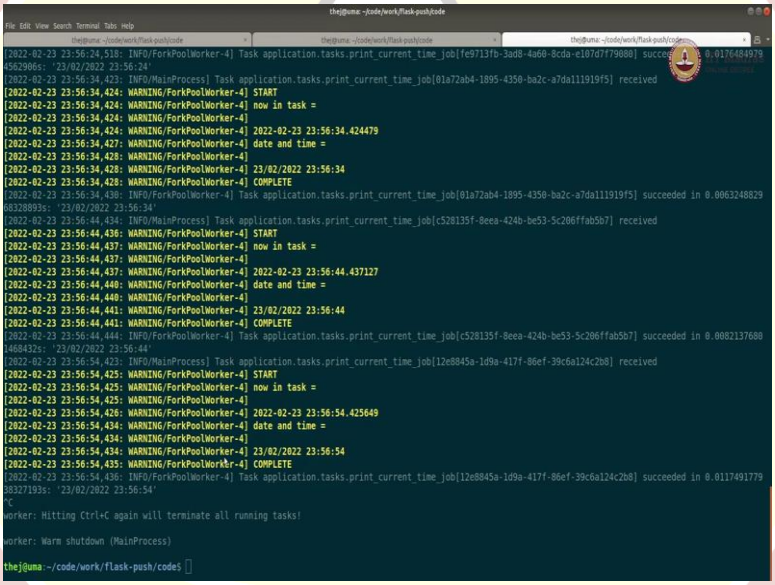


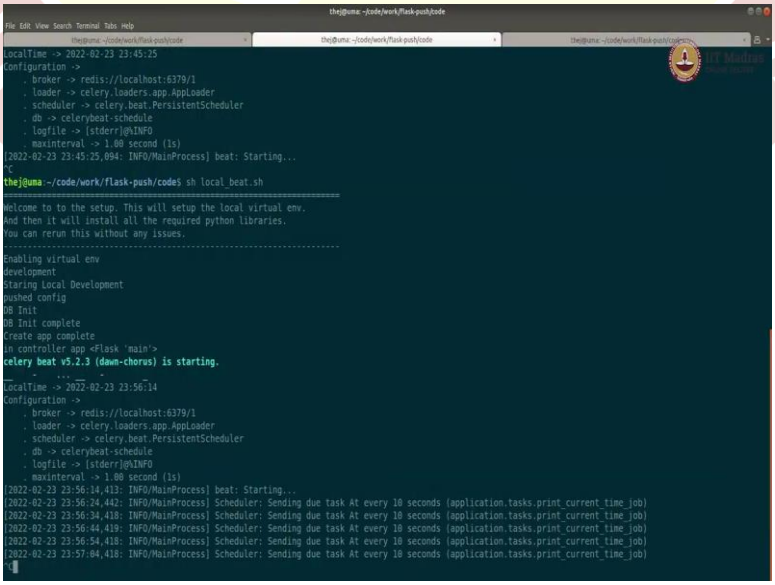


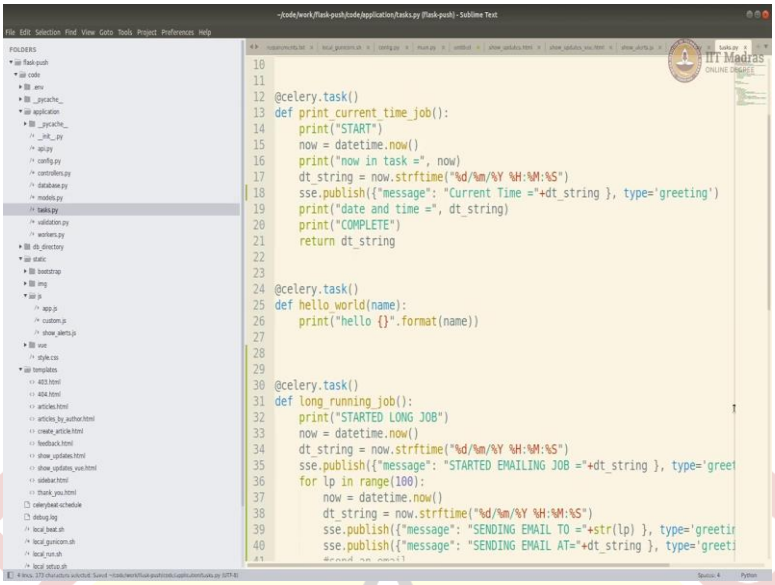
But I think let us see there is continue getting thin, there we go, rate for, there you go. This one is every 10 seconds to wait.

(Refer Slide Time: 37:29)







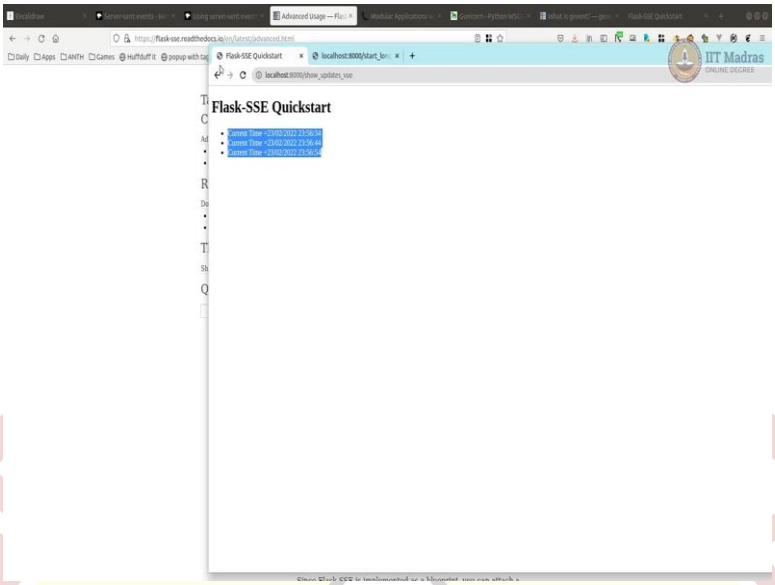


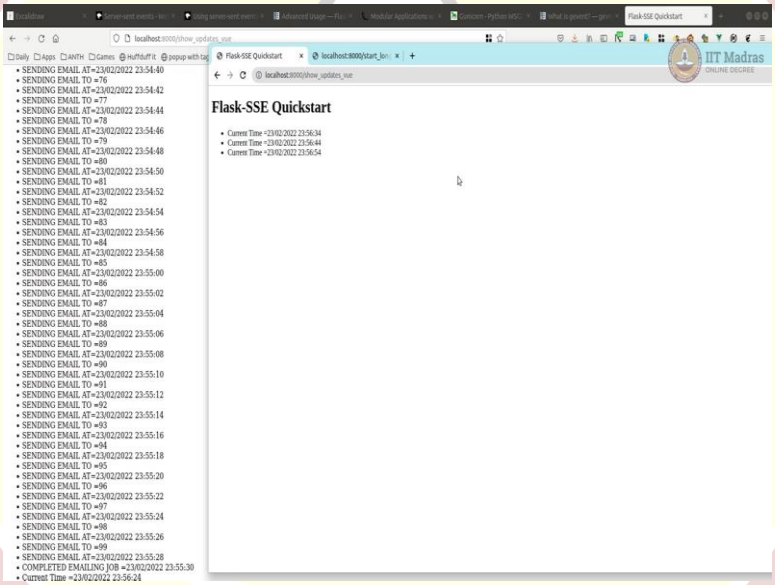
One more we will watch, so, now you get the idea. So it can be pushed, even without us being triggered. This is the time trigger. Just going to stop this time trigger. So, now until now what we learned, we learned that we can push the events by simple setup, but it needs (())(37:53) kind of setup, chase it to do that.

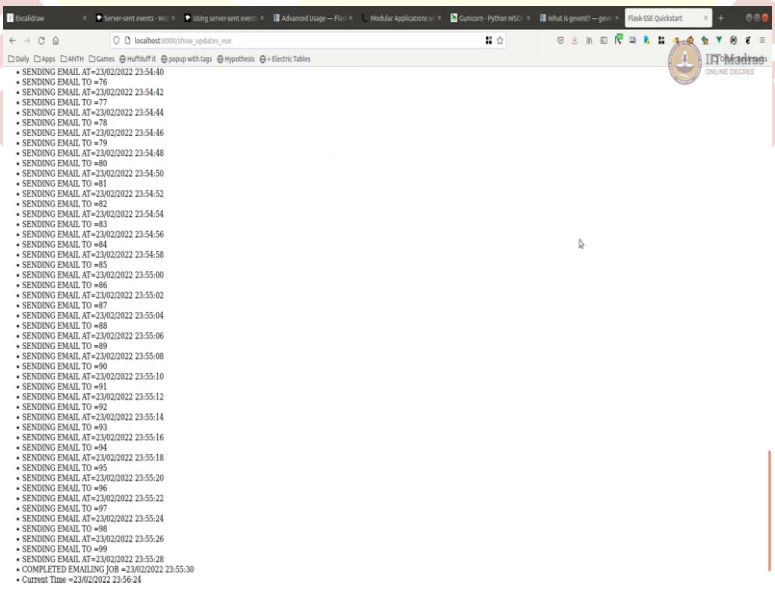
And we know that we can use in regular JavaScript or with view, what we need to know is there is a class type called event source that we are going to use. And we can call add listener on top of that for a specific type of event.

And then we can publish it, we are not doing any retry. Since last ID, that kind of thing, you can go actually look into that yourself. If you want to do retries. For example, once it gets disconnected, I want to go to the advanced part of it, there is one simple thing that I wanted to show in the advanced part, which is a need of not every user wants to see the same message.

(Refer Slide Time: 38:45)

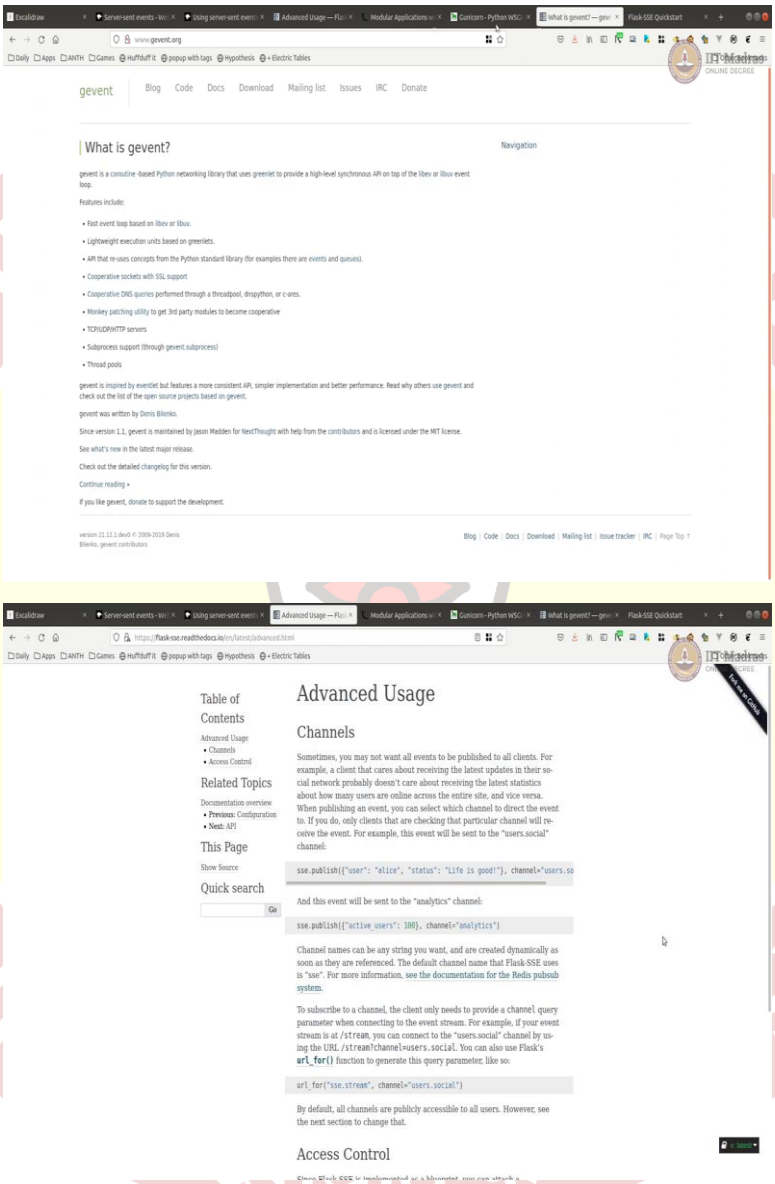


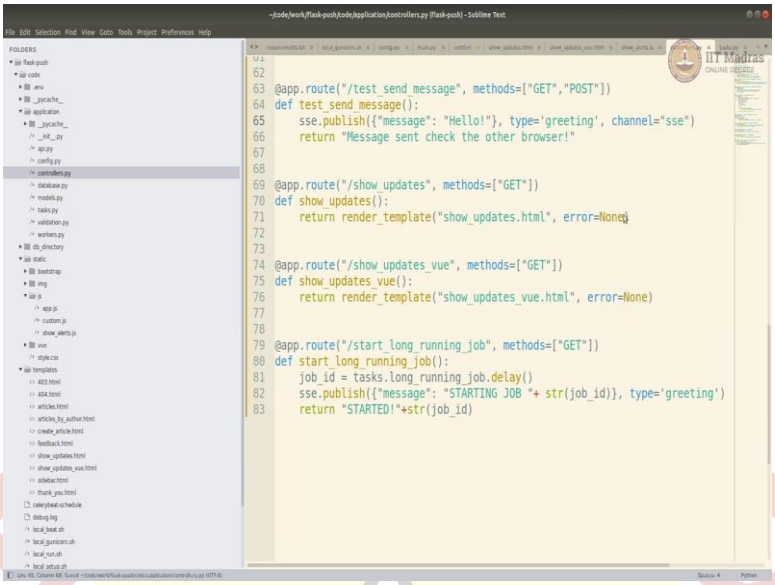




For example, here, this user and this user, were watching the same channel, and same user without any access control, or anything like that, for example, if you wanted to see his update, and this person wanted to see their update, they should have been looking into different channels.

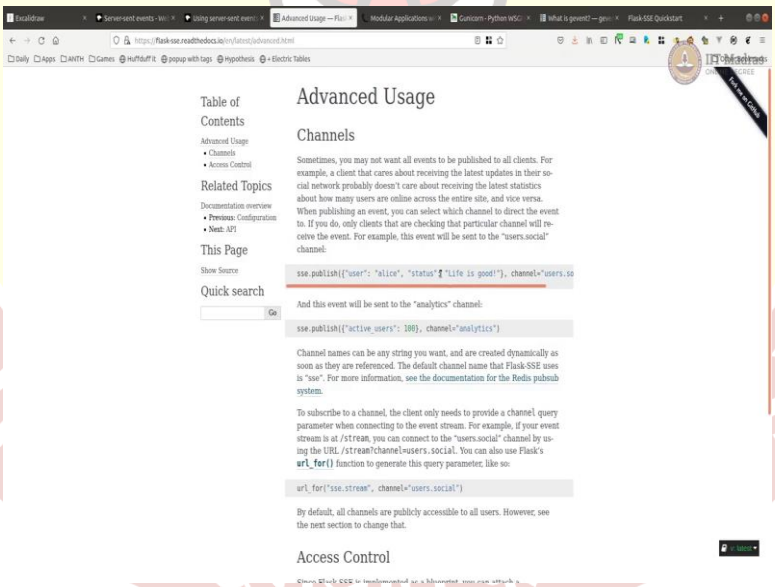
(Refer Slide Time: 39:05)

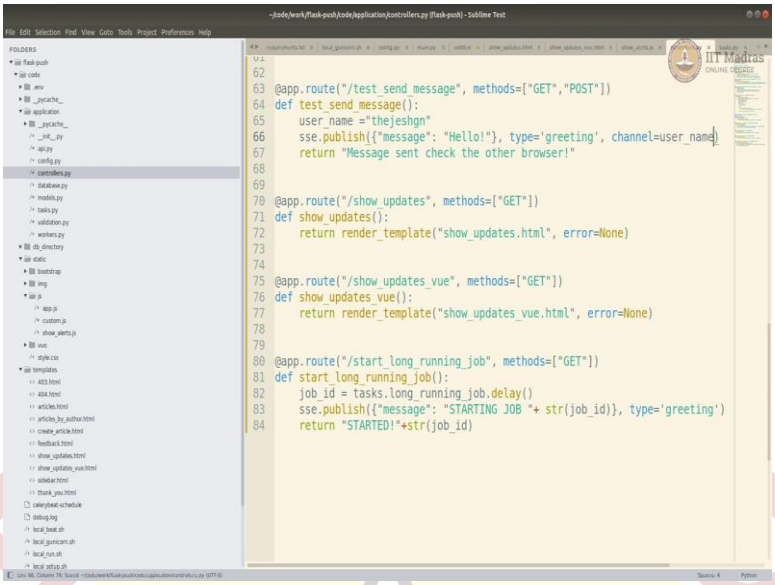




Now flask-sse gives you concept of channels, channels could be like very specific to user, and it is just a string. So, it could be like we could be passing it a specific channel, let us say. Here, if I go to tasks, let us go to controller. And here we are pushing it to this is even type, but channel is default, default analysts sse.

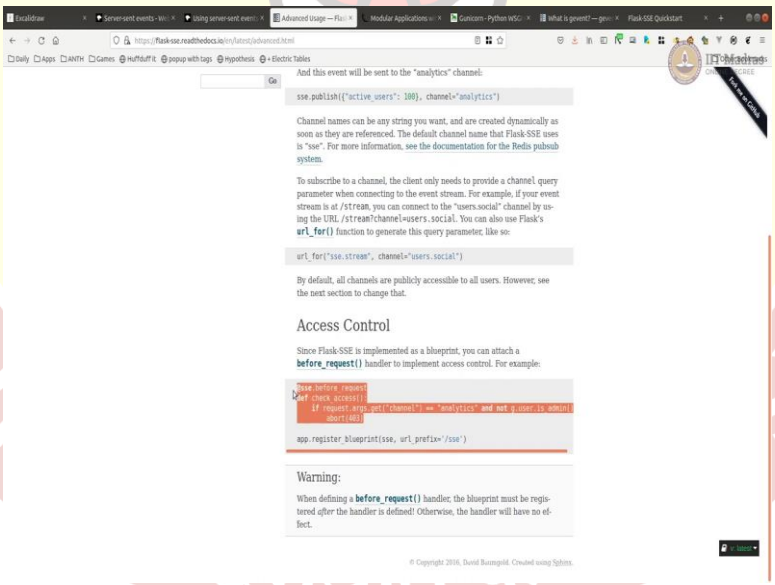
(Refer Slide Time: 39:44)

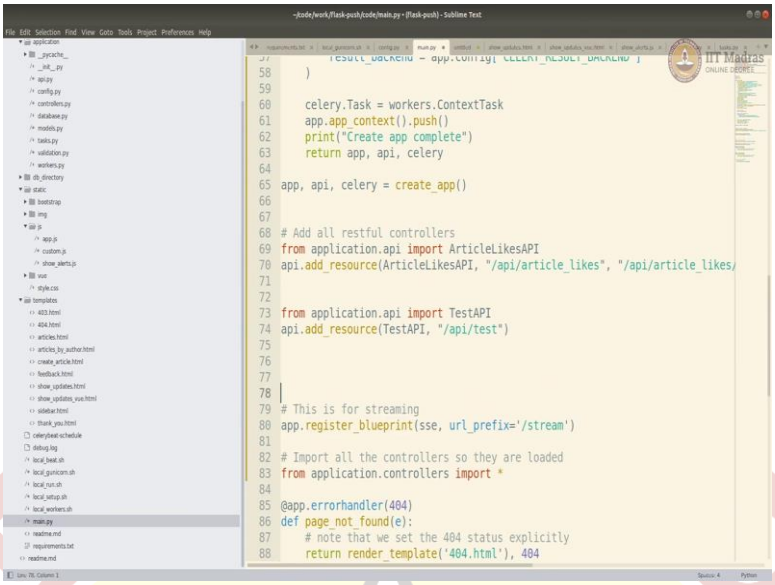


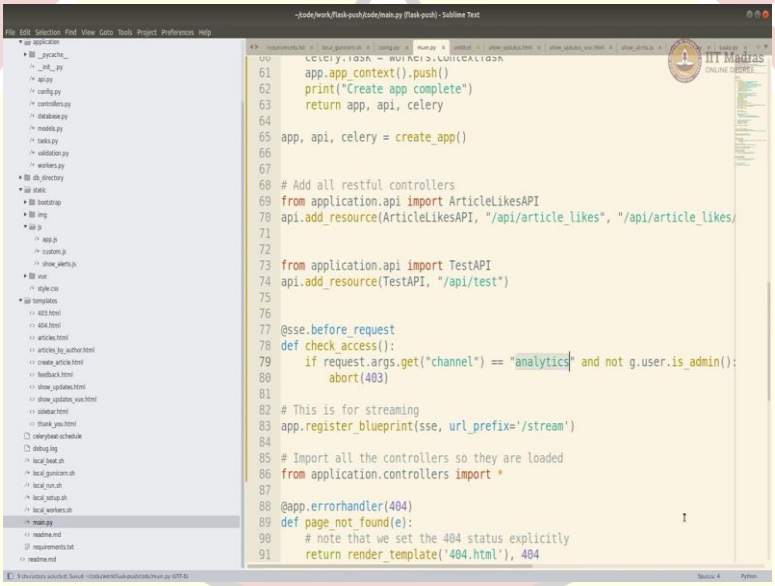


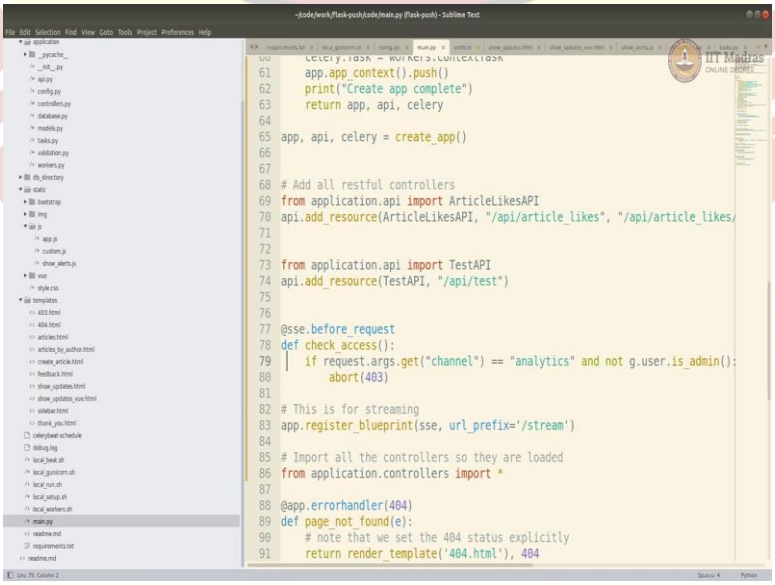
But let us say wanted to push into some other channel like user stock social or you wanted to point it to this specific loser, let us say if the user name is equal to a Thejesh. Ideally, I would be taking from art but I am just hard coding here. You could pass us the art of the channel limit he can subscribe to that channel.

(Refer Slide Time: 40:18)



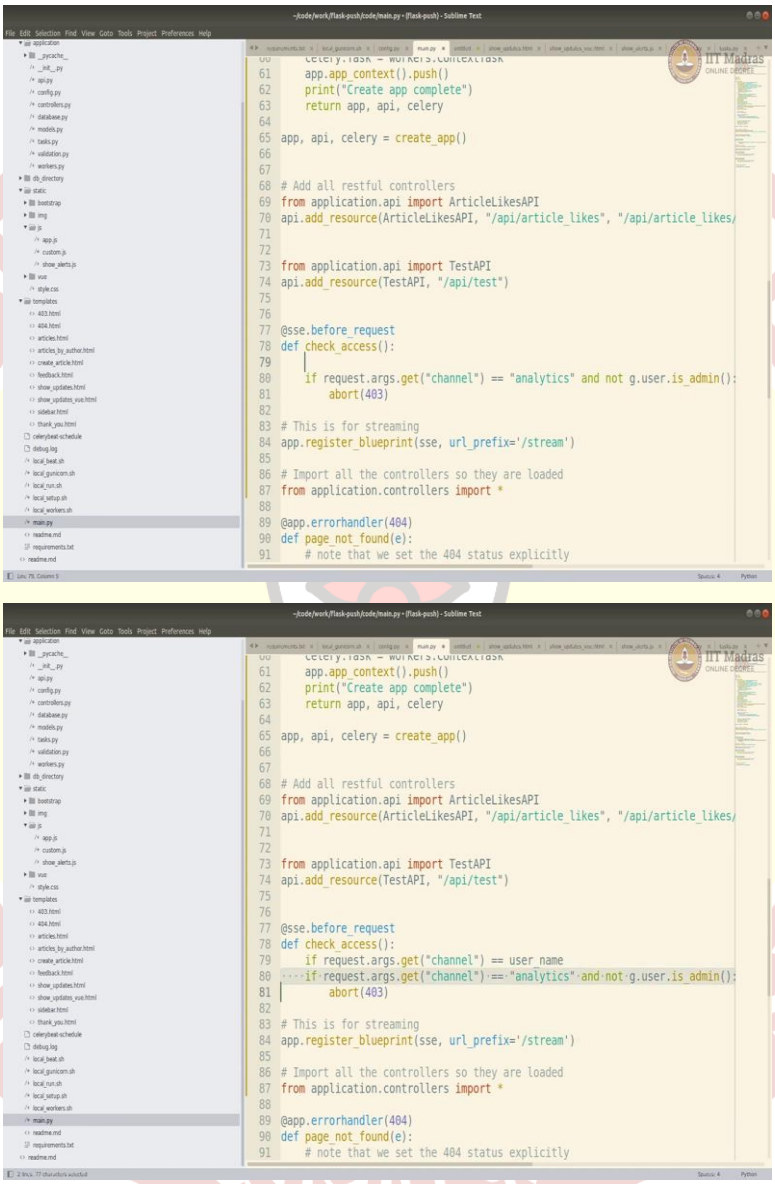

~/code/work/flask-push/code/main.py - (flask-push) - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
main.py
58)
59
60 celery.Task = workers.ContextTask
61 app.app_context().push()
62 print("Create app complete")
63 return app, api, celery
64
65 app, api, celery = create_app()
66
67
68 # Add all restful controllers
69 from application.api import ArticleLikesAPI
70 api.add_resource(ArticleLikesAPI, "/api/article_likes", "/api/article_likes/
71
72
73 from application.api import TestAPI
74 api.add_resource(TestAPI, "/api/test")
75
76
77
78
79 # This is for streaming
80 app.register_blueprint(sse, url_prefix='/stream')
81
82 # Import all the controllers so they are loaded
83 from application.controllers import *
84
85 @app.errorhandler(404)
86 def page_not_found(e):
87 # note that we set the 404 status explicitly
88 return render_template('404.html'), 404
Line 78, Column 1
Spaces: 4
Python

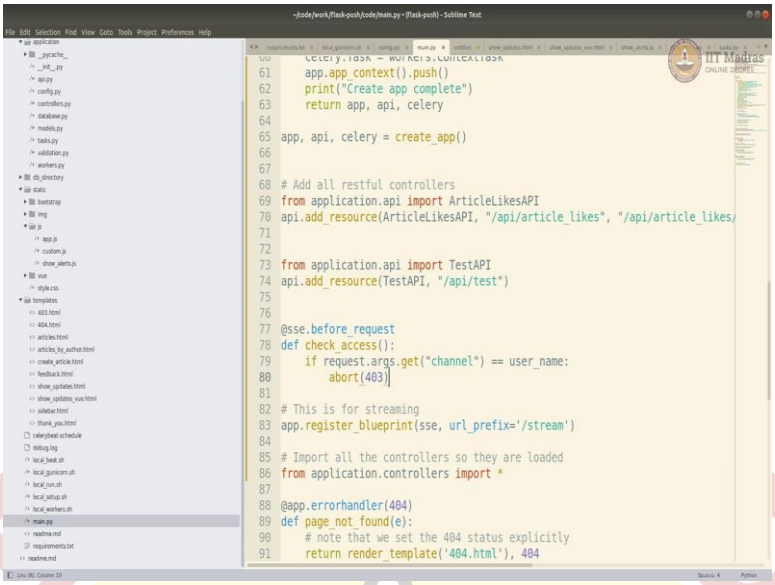

~/code/work/flask-push/code/main.py (flask-push) - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
main.py
61 app.app_context().push()
62 print("Create app complete")
63 return app, api, celery
64
65 app, api, celery = create_app()
66
67
68 # Add all restful controllers
69 from application.api import ArticleLikesAPI
70 api.add_resource(ArticleLikesAPI, "/api/article_likes", "/api/article_likes/
71
72
73 from application.api import TestAPI
74 api.add_resource(TestAPI, "/api/test")
75
76
77 @sse.before_request
78 def check_access():
79 if request.args.get("channel") == "analytics" and not g.user.is_admin():
80 abort(403)
81
82 # This is for streaming
83 app.register_blueprint(sse, url_prefix='/stream')
84
85 # Import all the controllers so they are loaded
86 from application.controllers import *
87
88 @app.errorhandler(404)
89 def page_not_found(e):
90 # note that we set the 404 status explicitly
91 return render_template('404.html'), 404
Spaces: 4
Python


~/code/work/flask-push/code/main.py (flask-push) - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
main.py
61 app.app_context().push()
62 print("Create app complete")
63 return app, api, celery
64
65 app, api, celery = create_app()
66
67
68 # Add all restful controllers
69 from application.api import ArticleLikesAPI
70 api.add_resource(ArticleLikesAPI, "/api/article_likes", "/api/article_likes/
71
72
73 from application.api import TestAPI
74 api.add_resource(TestAPI, "/api/test")
75
76
77 @sse.before_request
78 def check_access():
79 if request.args.get("channel") == "analytics" and not g.user.is_admin():
80 abort(403)
81
82 # This is for streaming
83 app.register_blueprint(sse, url_prefix='/stream')
84
85 # Import all the controllers so they are loaded
86 from application.controllers import *
87
88 @app.errorhandler(404)
89 def page_not_found(e):
90 # note that we set the 404 status explicitly
91 return render_template('404.html'), 404
Line 79, Column 2
Spaces: 4
Python

And the event could be anything and then he could go ahead and add an extract says control to check the axes in the default main let where we are including, where is it, here before we include we could do some access check before the request if channel is equal to analytics, then g is g an admin then only then.

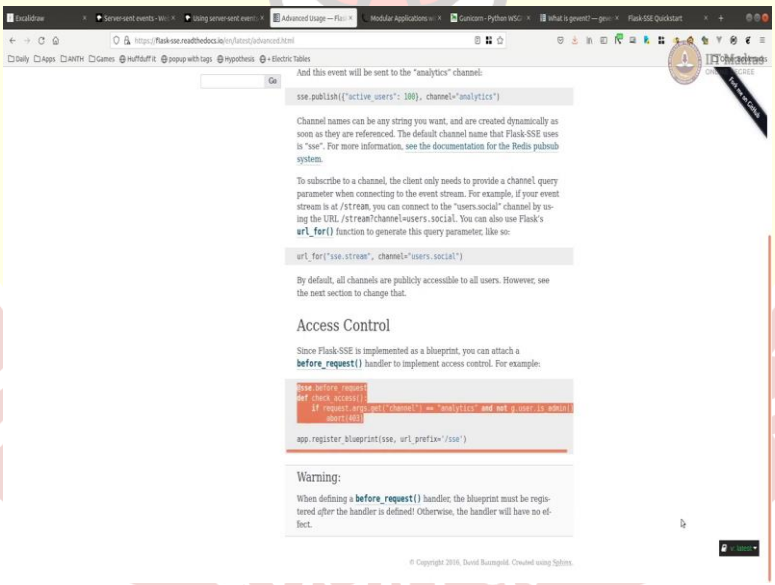
(Refer Slide Time: 40:54)

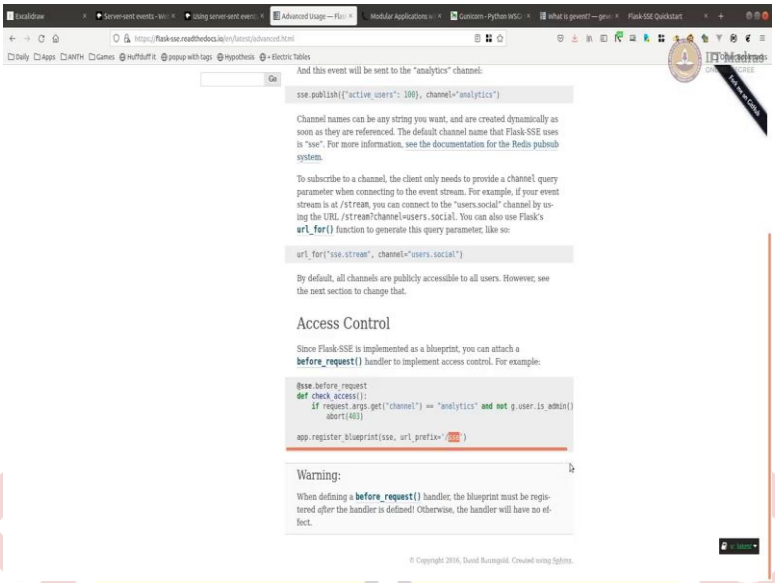


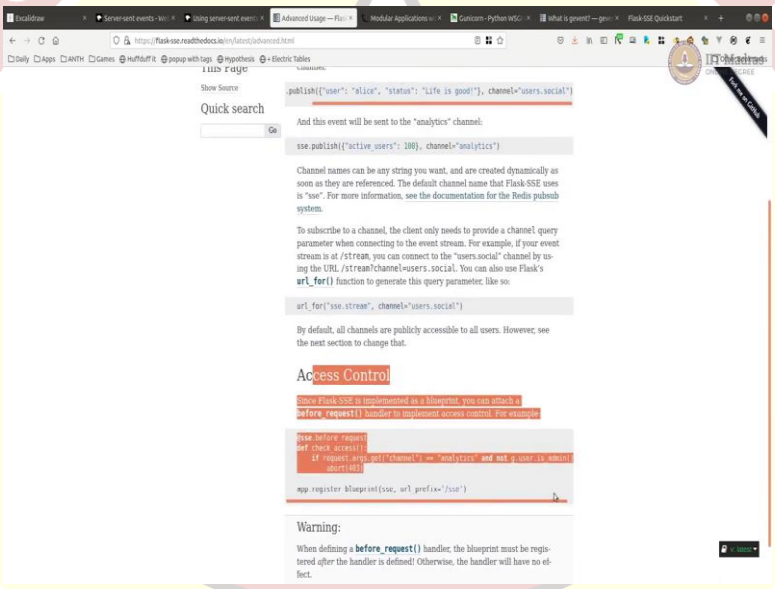


Or it could go with like if channel security is equal to user name. Only then allow hilt, do 403 you could do this as well. Then what it does it? It checks whether he is trying to subscribe to specific channel if the channel name is user name, if not, we will run 403. So, you can do this kind of access control.

(Refer Slide Time: 41:29)

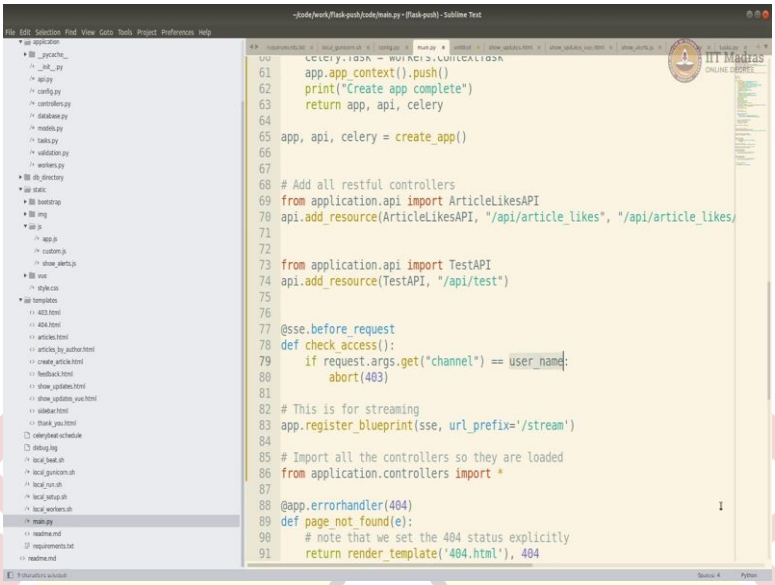






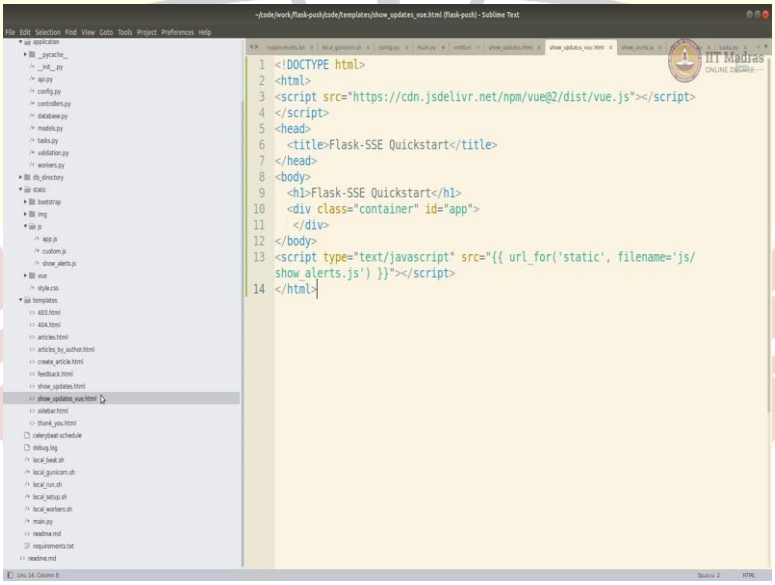
I am here, here of your prefix sse, but it could both or works. So, this is how you do channels and access control on the channels make specific channels for reach of the user or reach action activity or something.

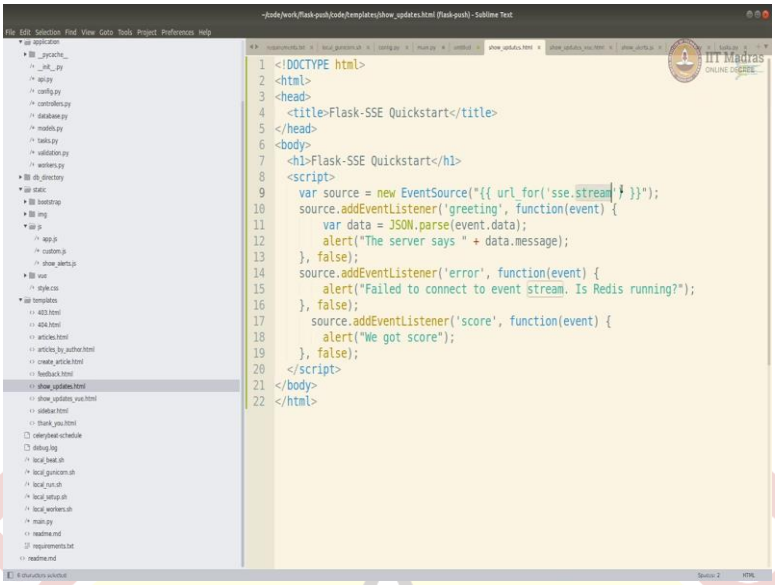
(Refer Slide Time: 41:52)



And validate based on the user or the home that activity belongs to. I usually generally, how like channels per user, if you want to send an update to that specific user, and then actually use that to get the access control, check his username is equal to same as channel, if not through an error.

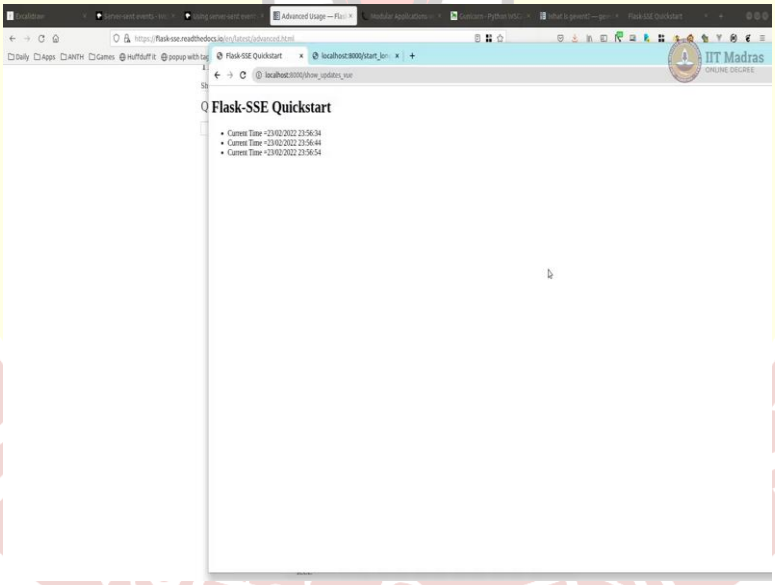
(Refer Slide Time: 42:14)

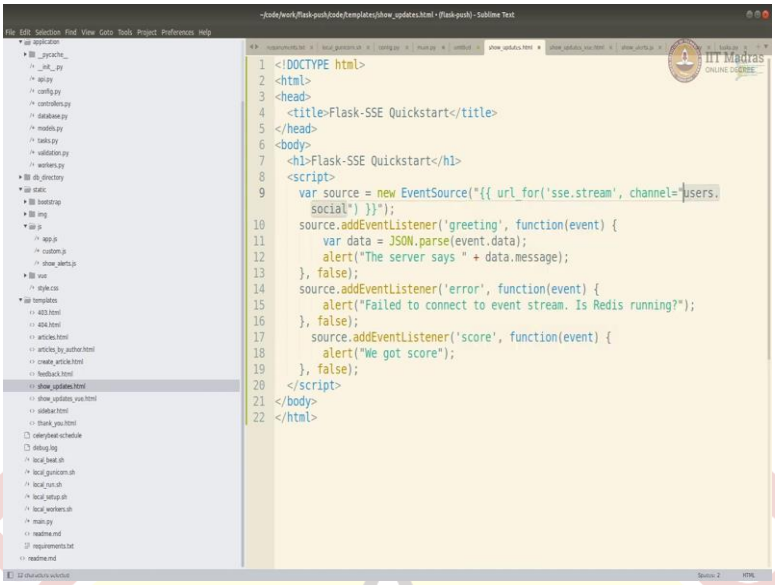


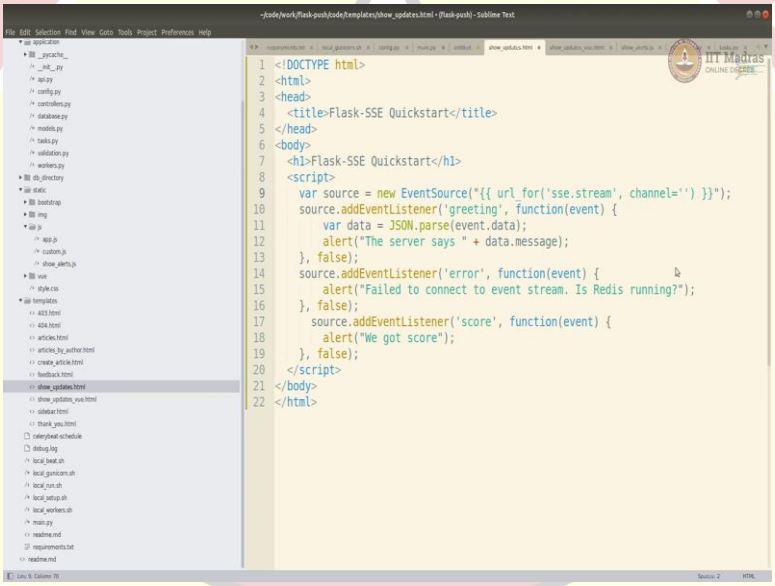


Then you will have to make one small change here too. Because you will not be for example, here, you will not be directly mapping it to things you have to will be mapping to specific channel, not just UN type. So, you could use channel like that.

(Refer Slide Time: 42:32)

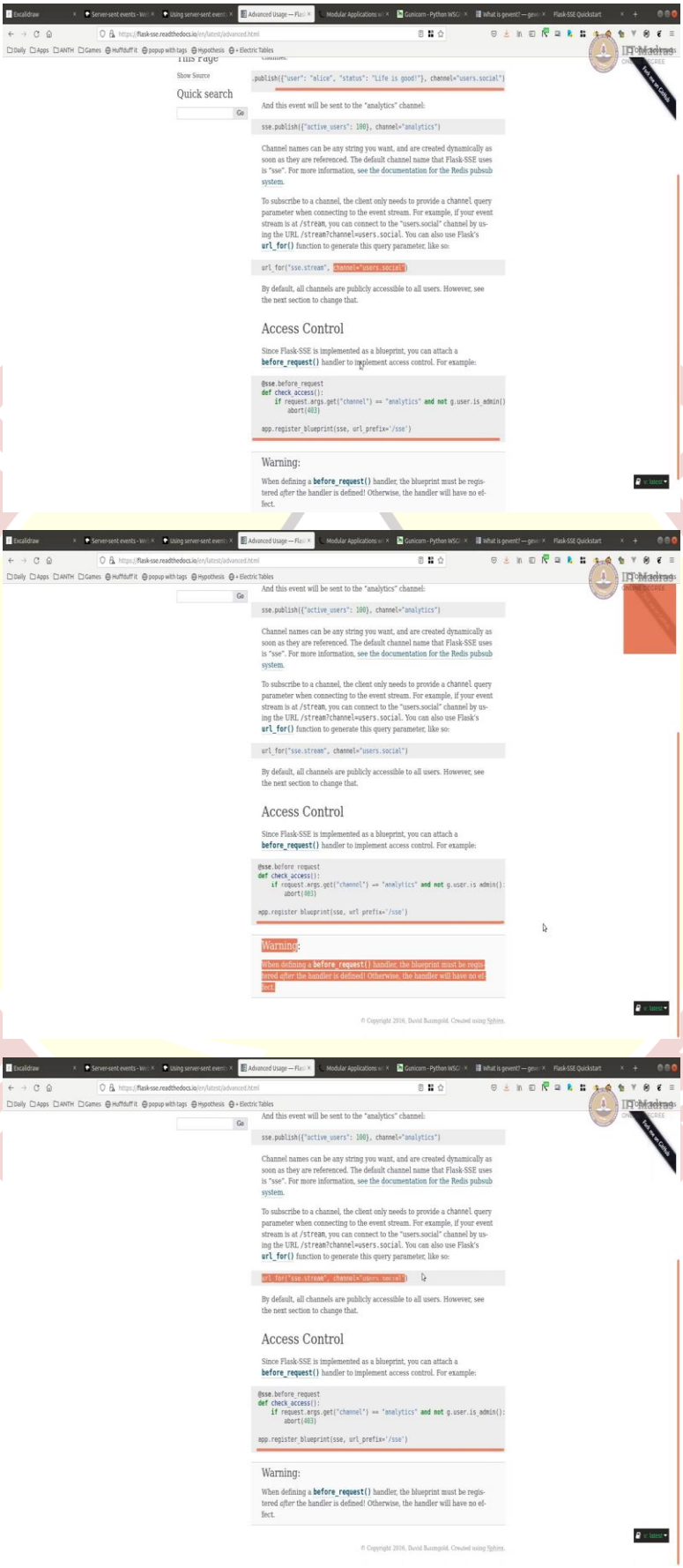






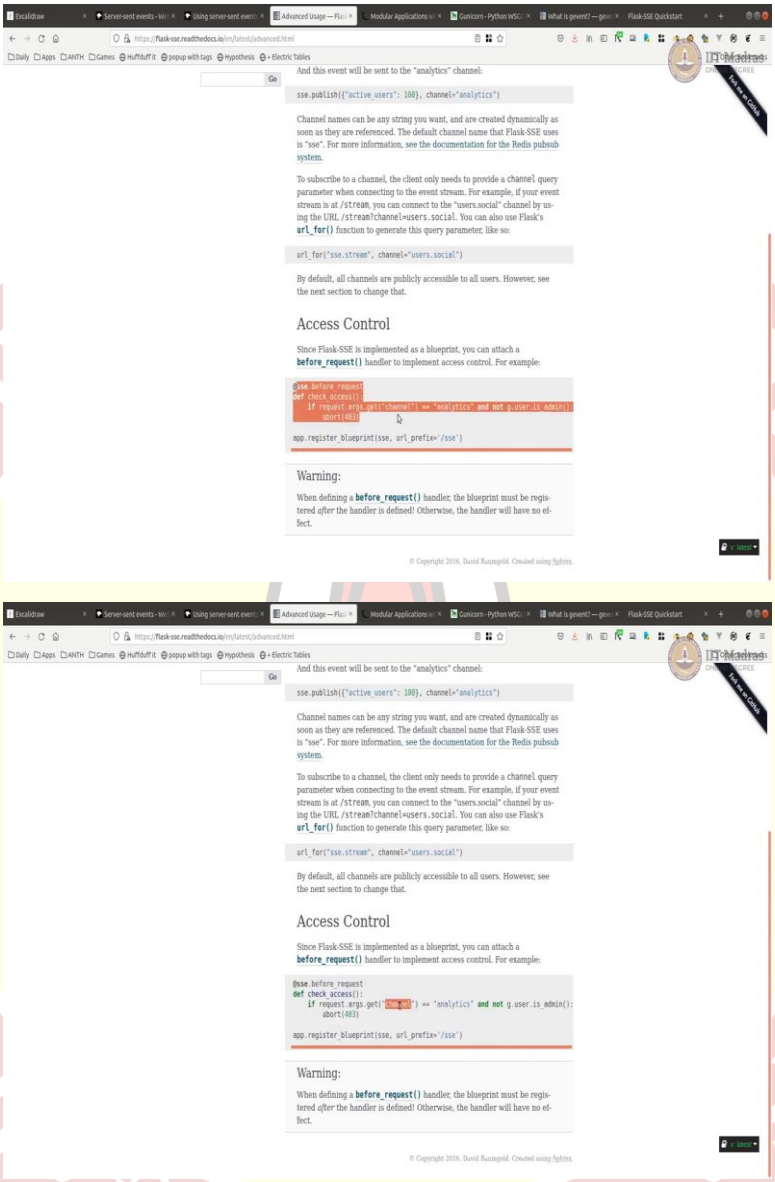
For example, this will remain the same, then let me use a name that will be Thejesh G. N. and then it will form the URL for you. And then you are subscripted that specific topic, with double quote, single quote, so that how you get the type of subscribe to specific user for a specific channel.

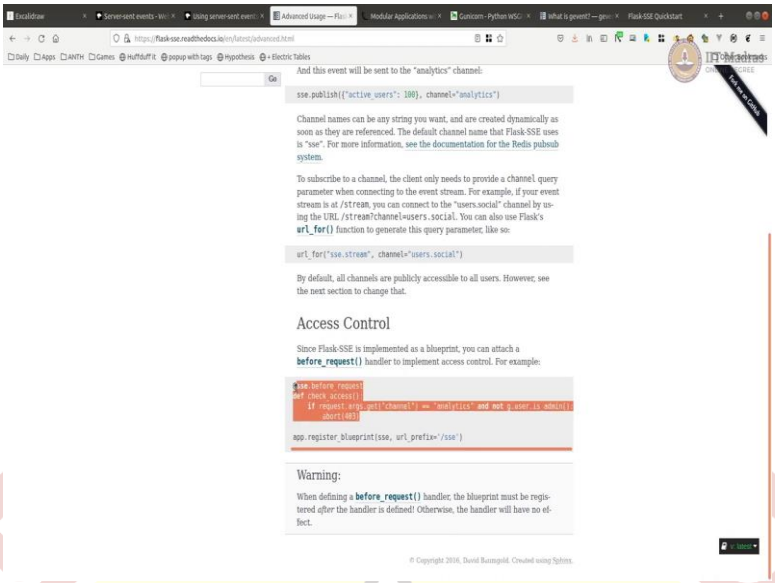
(Refer Slide Time: 43:05)



I not going to show you the example with that, I would actually want you to try it and see how it goes. Add a user specific channel based on this user name.

(Refer Slide Time: 43:17)





And make sure who is trying to use the channel, his user name and the channel names are same (())(43:23) through an error. Make that change. Try it out locally. Thank you so much for watching.