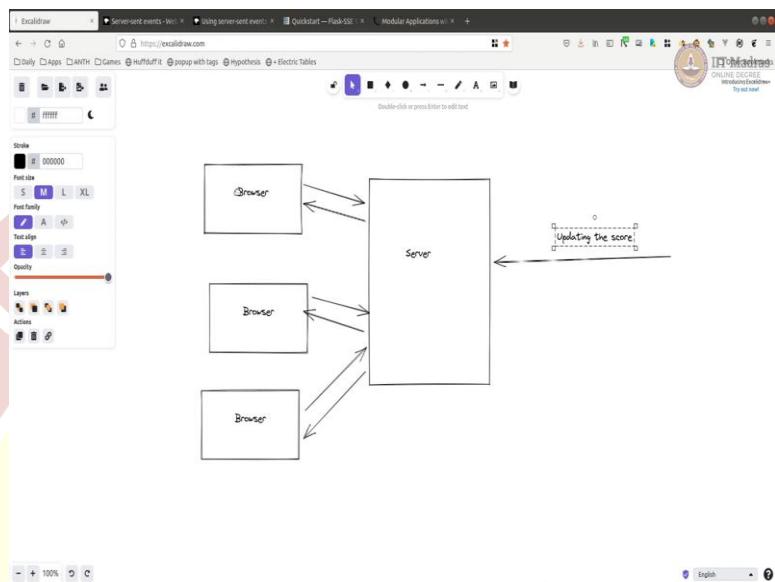


IIT Madras

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 score 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.

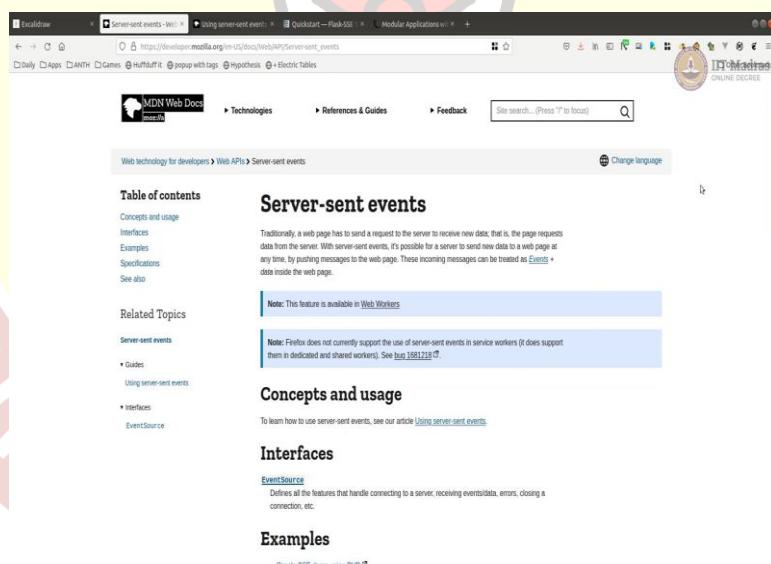
Now, what is the problem here? The match can go at its own speed. Sometimes the score does not change much. Sometimes the score changes, sometimes you add commentary,

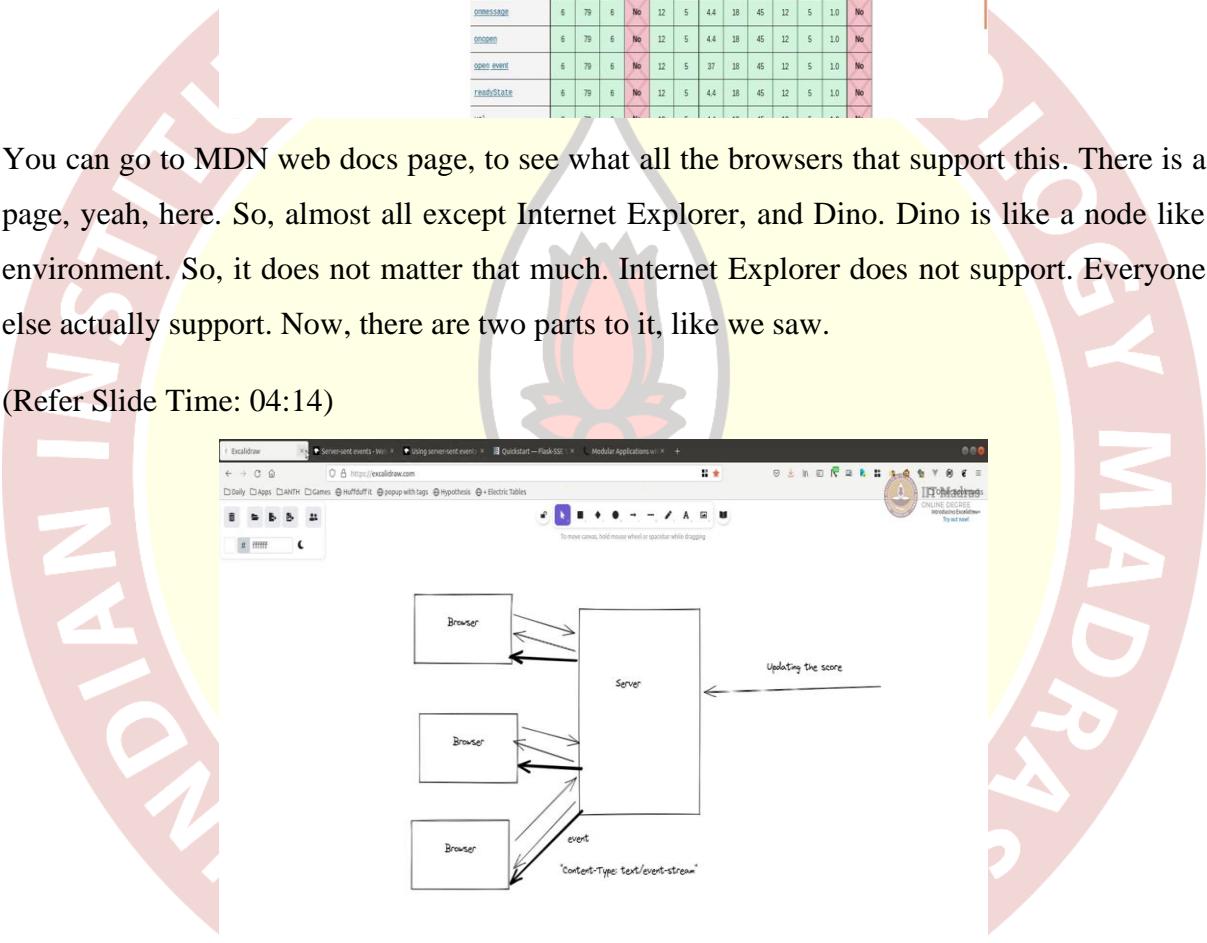
sometimes you do not add commentary. So, for the users to get the latest, he has to keep refreshing the page, Otherwise, you will not get the latest, everything is a request response, you will not get any score if it does not refresh.

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)



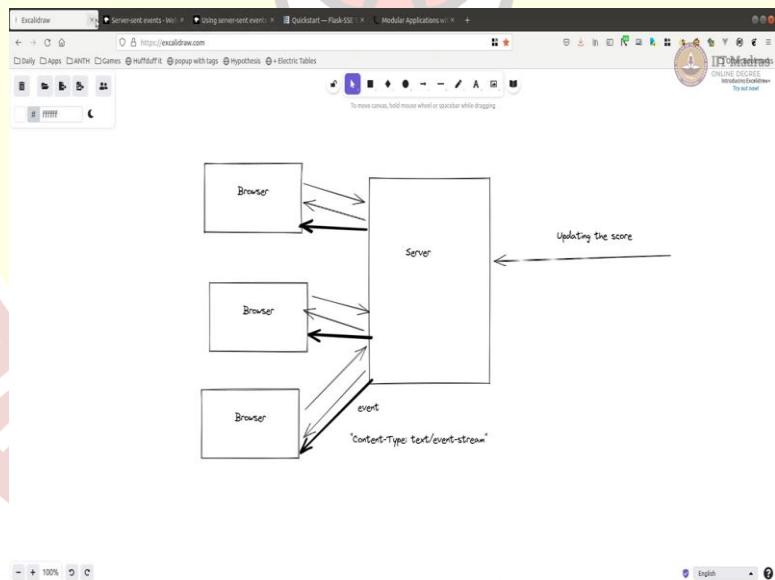


The table shows browser support for various EventSource methods across different browsers. The columns represent different browsers and devices, and the rows represent different methods.

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	Dino
<code>EventSource()</code>	6	79	6	No	11	5	44	18	45	11	5	10	No
<code>CORS support (withCredentials)</code>	26	79	11	No	12	7	37	26	45	12	7	20	No
<code>Close</code>	6	79	6	No	12	5	44	18	45	12	5	10	No
<code>error_event</code>	6	79	6	No	12	5	37	18	45	12	5	10	No
<code>message_event</code>	6	79	6	No	12	5	37	18	45	12	5	10	No
<code>onerror</code>	6	79	6	No	12	5	44	18	45	12	5	10	No
<code>onmessage</code>	6	79	6	No	12	5	44	18	45	12	5	10	No
<code>open</code>	6	79	6	No	12	5	44	18	45	12	5	10	No
<code>open_event</code>	6	79	6	No	12	5	37	18	45	12	5	10	No
<code>readyState</code>	6	79	6	No	12	5	44	18	45	12	5	10	No

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.

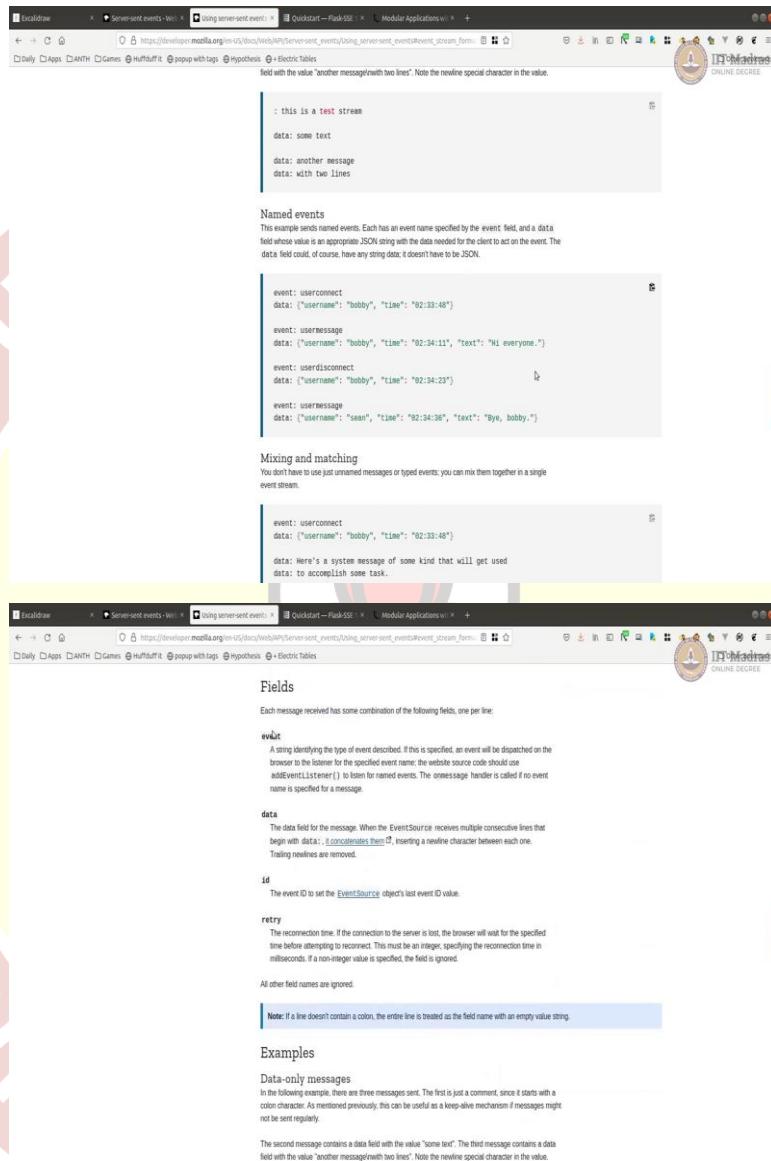
(Refer Slide Time: 04:14)



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.

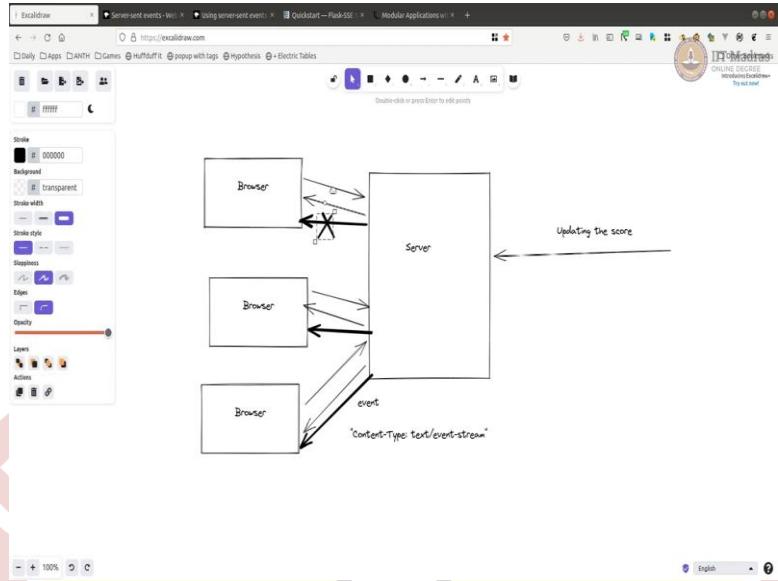
So, this is how content type text slash event stream. And that is how the server sends it, just like in this format, but it is a text that it sent. And then the browser knows how to understand this event stream and calls respect to JavaScript methods, so that you can handle them.

(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)

Screenshot of the Mozilla developer documentation for Server-Sent Events (https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events). The page is titled 'Fields' and provides details about the fields used in event messages. It includes sections for 'event', 'data', 'id', 'retry', and examples of valid message formats. A note at the bottom states: 'Note: If a line doesn't contain a colon, the entire line is treated as the field name with an empty value string.'

Fields

Each message received has some combination of the following fields, one per line:

event
A string identifying the type of event described. If this is specified, an event will be dispatched on the browser to the listener for the specified event name; the website source code should use `addEventListerner()` to listen for named events. The onmessage handler is called if no event name is specified for a message.

data
The data field for the message. When the EventSource receives multiple consecutive lines that begin with data: : `reconnection_time`, inserting a newline character between each one. Trailing newlines are removed.

id
`EventID` to set the `EventSource` object's id event ID value.

retry
The reconnection time. If the connection to the server is lost, the browser will wait for the specified time before attempting to reconnect. This must be an integer, specifying the reconnection time in milliseconds. If a non-integer value is specified, the field is ignored.

All other field names are ignored.

Note: If a line doesn't contain a colon, the entire line is treated as the field name with an empty value string.

Examples

Data-only messages
In the following example, there are three messages sent. The first is just a comment, since it starts with a colon character. As mentioned previously, this can be useful as a keep-alive mechanism if messages might not be sent regularly.

The second message contains a data field with the value "some text". The third message contains a data field with the value "another message\nwith two lines". Note the newline special character in the value.

Named events

```
event: userconnect
data: {"username": "bobby", "time": "02:33:48"}  
event: 1  
data: {"username": "bobby", "time": "02:34:11", "text": "Hi everyone."}  
event: userdisconnect
data: {"username": "bobby", "time": "02:34:23"}  
event: usermessage
data: {"username": "sean", "time": "02:34:36", "text": "Bye, bobby."}
```

Mixing and matching

```
event: userconnect
data: {"username": "bobby", "time": "02:33:48"}  
data: Here's a system message of some kind that will get used  
data: to accomplish some task.
```

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)

Table of contents

- Receiving events from the server
- Sending events from the server
- Error handling
- Closing event streams
- Event stream format
- Browser compatibility

Related Topics

- Guides
 - Using server-sent events
- Interfaces
 - EventSource

Using server-sent events

Developing a web application that uses `server-sent events` is straightforward. You need a bit of code on the server to stream events to the front-end, but the client side code works almost identically to `WebSockets` in part of handling incoming events. This is a one-way connection, so you can't send events from a client to a server.

Receiving events from the server

The server-sent event API is contained in the `EventSource` interface. To open a connection to the server to begin receiving events from it, create a new `EventSource` object with the URL or a script that generates the events. For example:

```
const evtSource = new EventSource("ssedemo.php");
```

If the event generator script is hosted on a different origin, a new `EventSource` object should be created with both the URL and an options dictionary. For example, assuming the client script is on `example.com`:

```
const evtSource = new EventSource("//api.example.com/ssedemo.php", { withCredentials: true } );
```

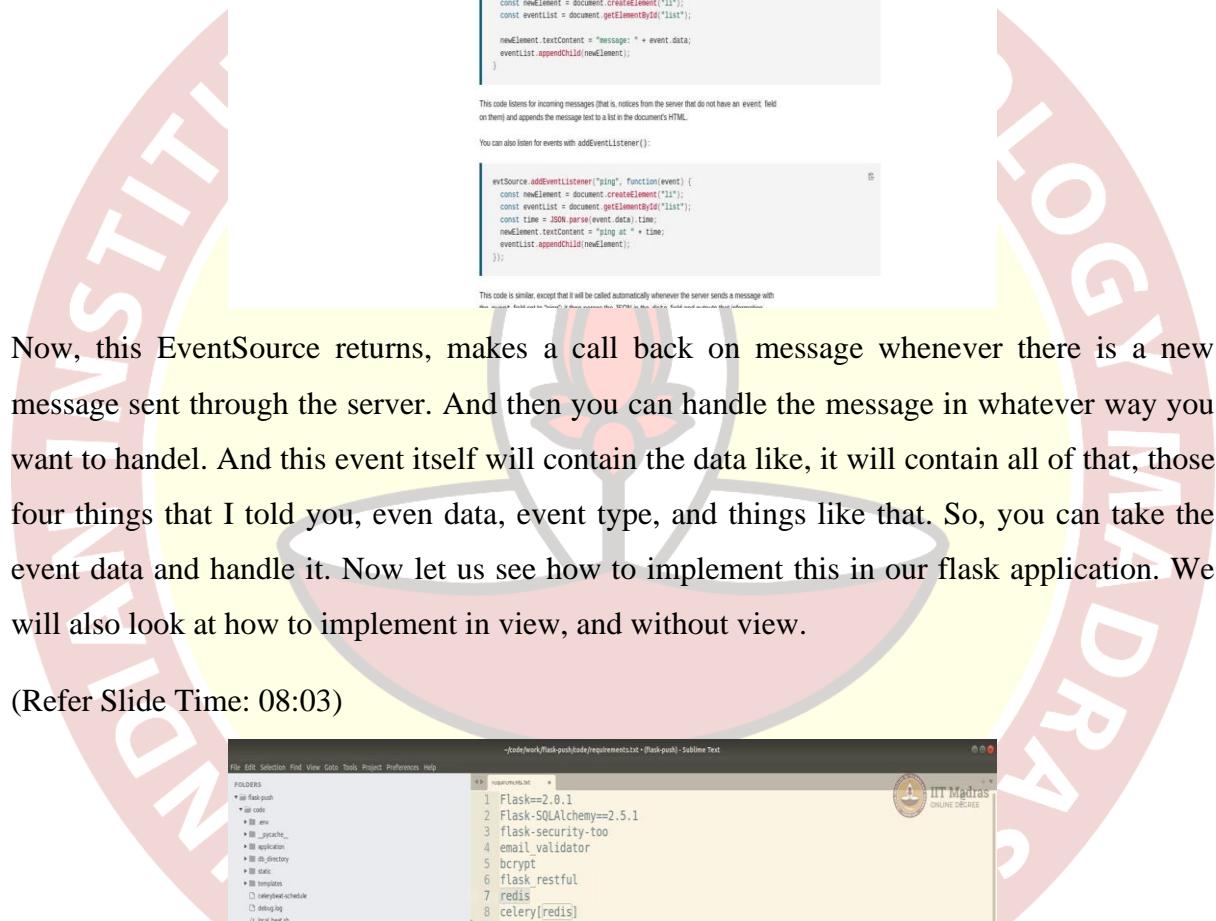
Once you've instantiated your event source, you can begin listening for messages from the server by attaching a handler for the `message` event:

```
evtSource.onmessage = function(event) {
```

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.

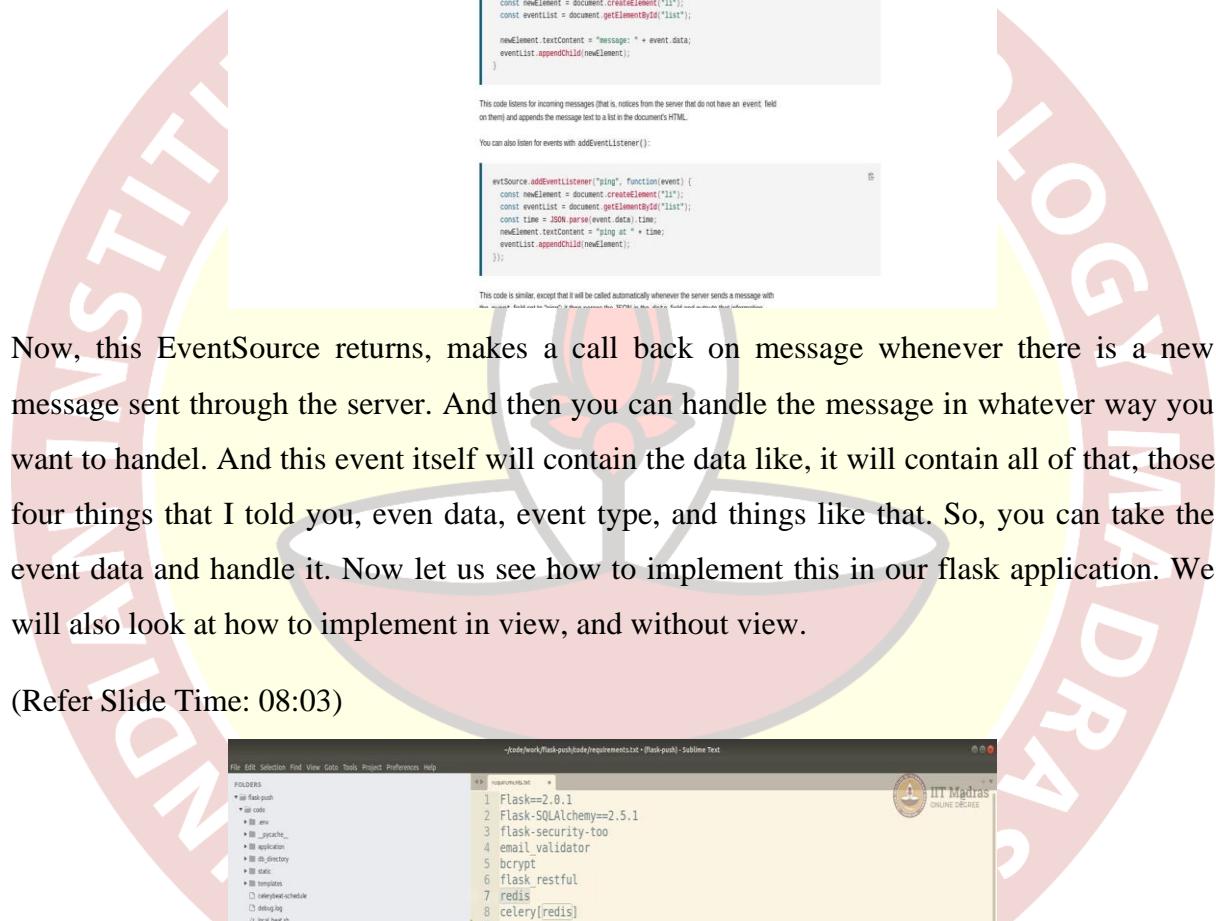
(Refer Slide Time: 07:25)



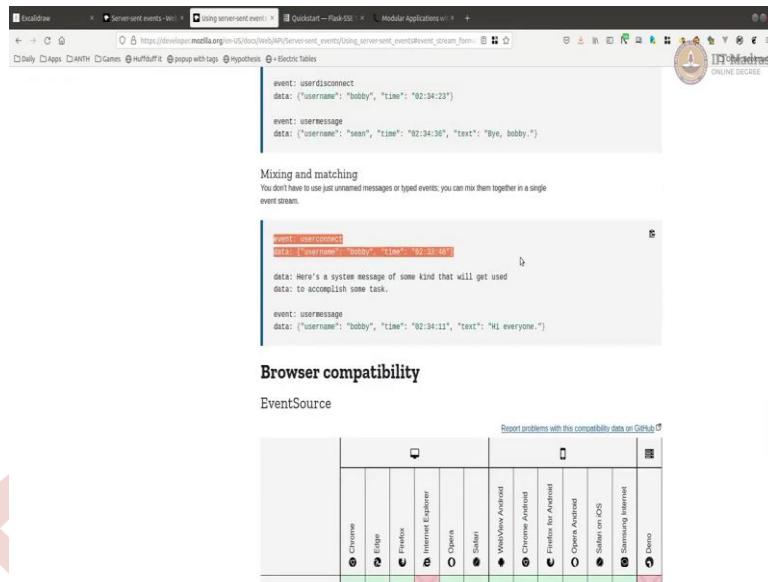
```
const evtSource = new EventSource("ssedemo.php");  
  
If the event generator script is hosted on a different origin, a new EventSource object should be created  
with both the URL and an options dictionary. For example, assuming the client script is on example.com:  
  
const evtSource = new EventSource("//api.example.com/ssedemo.php", { withCredentials: true } );  
  
Once you've instantiated your event source, you can begin listening for messages from the server by  
attaching a handler for the message event:  
  
evtSource.addEventListener("message", function(event) {  
  const newElement = document.createElement("li");  
  const eventList = document.getElementById("list");  
  
  newElement.textContent = "message: " + event.data;  
  eventList.appendChild(newElement);  
});  
  
This code listens for incoming messages (that is, notices from the server that do not have an event field  
on them) and appends the message text to a list in the document's HTML.  
  
You can also listen for events with addEventListener():  
  
evtSource.addEventListener("ping", function(event) {  
  const newElement = document.createElement("li");  
  const eventList = document.getElementById("list");  
  const time = JSON.parse(event.data).time;  
  newElement.textContent = "ping at " + time;  
  eventList.appendChild(newElement);  
});  
  
This code is similar, except that it will be called automatically whenever the server sends a message with  
the event field set to "ping". It then parses the JSON in the data field and inserts the information
```

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)



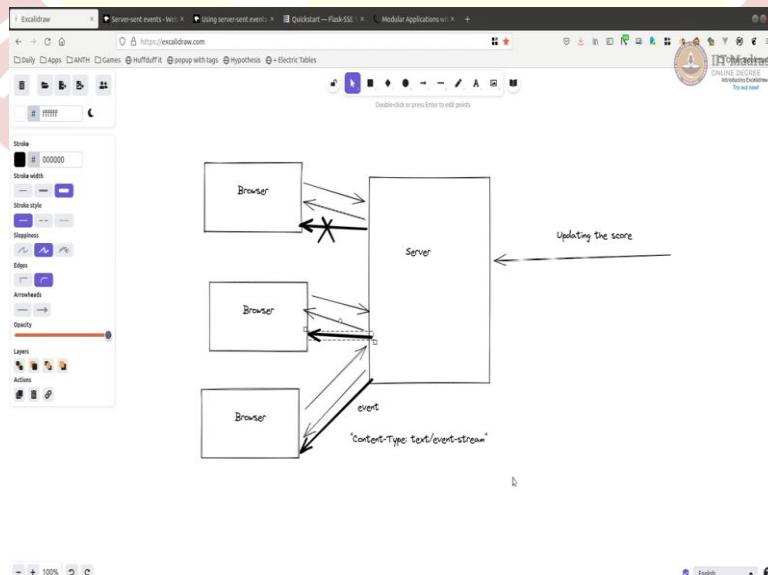
```
File Edit Selection Find View Goto Tools Project Preferences Help  
FOLDERS  
• app flask-push  
• .git  
• bin  
• lib  
• lib_venv  
• lib_python  
• lib_django  
• lib_static  
• lib_templates  
• lib_js_frontend  
• lib_css_frontend  
• local_beat.sh  
• local_rsys.sh  
• local_setup.sh  
• local_workers.sh  
• main.py  
• manage.md  
• requirements.txt  
• README.md  
• requirements.txt  
requirements.txt  
1 Flask==2.0.1  
2 Flask-SQLAlchemy==2.5.1  
3 flask-security-too  
4 email_validator  
5 bcrypt  
6 flask_restful  
7 redis  
8 celery[redis]  
9  
10 flask-sse  
11  
12 gunicorn  
13 gevent
```



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.

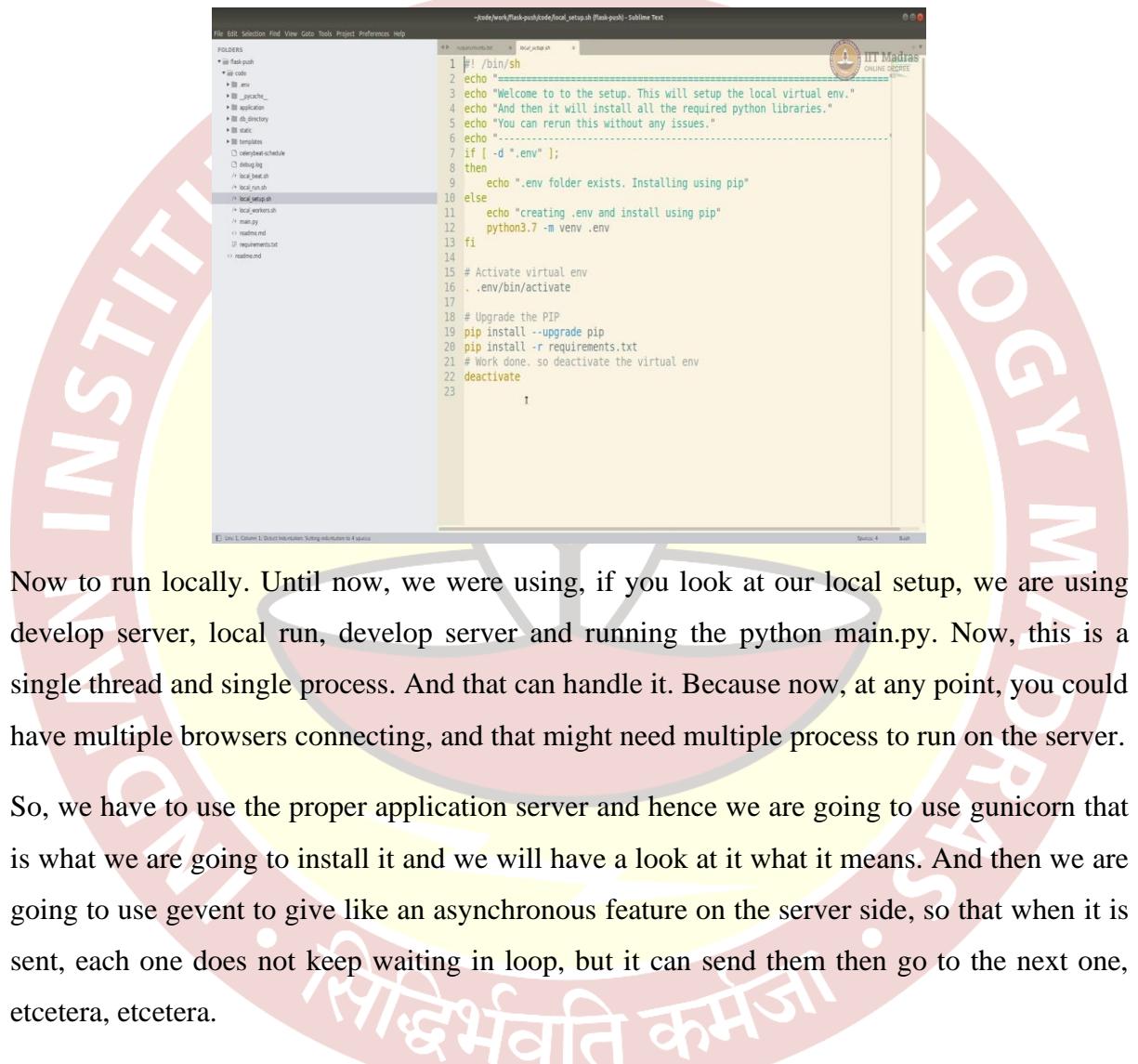
But to make it easy, flask-sse wraps around it. And it is easy to use. And it also uses Redis internally. So, you should have Redis installed already. So, I think since we are already installed Celery Redis is already installed. So, we are going to use the same package on same server as well. Why does 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)



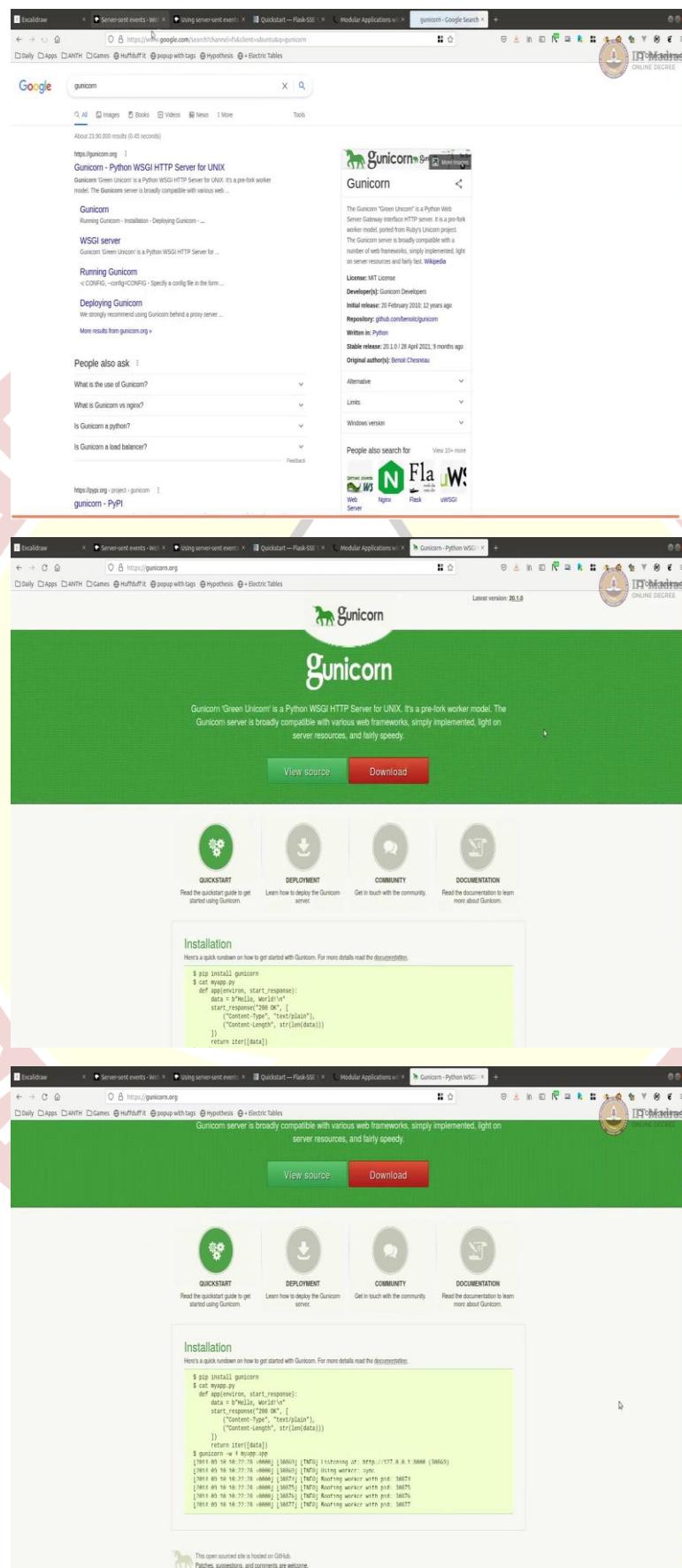
A screenshot of a Sublime Text editor window showing a file named `flask_push/code/flask_setup.sh`. The code in the file is a shell script for setting up a local virtual environment:

```
#!/bin/sh
echo =====
echo "Welcome 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 ".env folder exists. Installing using pip"
else
    echo "creating .env and install using pip"
    python3.7 -m venv .env
fi
#
# Activate virtual env
. .env/bin/activate
#
# Upgrade the PIP
pip install --upgrade pip
pip install -r requirements.txt
# Work done. so deactivate the virtual env
deactivate
```

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.

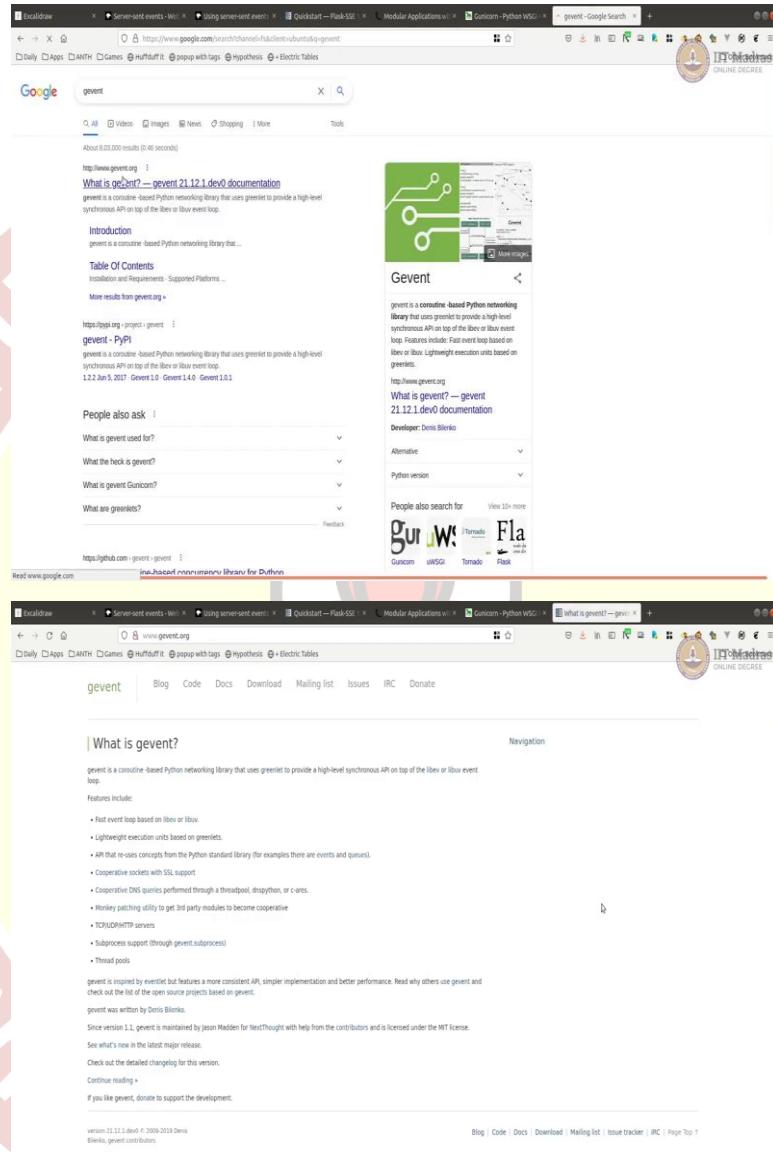
So, we have to use the proper application server and hence we are going to use gunicorn that is what we are going to install it and we will have a look at it what it means. And then we are going to use gevent to give like an asynchronous feature on the server side, so that when it is sent, each one does 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)

The image shows two screenshots of a terminal window and a code editor. The terminal window at the bottom displays a command-line interface with several files listed, including 'application', 'celerybeat-schedule', 'db_directory', 'debug.log', 'dot.env', 'main.py', 'pycache', 'requirements.txt', 'static', 'templates', 'local.beat.sh', 'local.run.sh', 'local.setup.sh', 'readme.md', and 'local.workers.sh'. The code editor at the top shows a file named 'requirements.txt' with the following content:

```
1 Flask==2.0.1
2 Flask-SQLAlchemy==2.5.1
3 flask-security-too
4 email_validator
5 bcrypt
6 flask_restful
7 redis
8 celery[redis]
9 flask-sse
10 gunicorn
11 gevent
```

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)

The image consists of three vertically stacked screenshots of a Sublime Text editor window, showing code snippets for different files. The top two screenshots have a light gray background, while the bottom one has a white background.

Screenshot 1 (Top): The title bar says "File Edit Selection Find View Goto Tools Project Preferences Help". The file path is "/code/work/flask-push/code/local_unicorns.sh". The code content is:

```
1 #!/bin/sh
2 echo "=====
3 echo "Welcome to the setup. This will setup the local virtual env."
4 echo "And then it will install all the required python libraries."
5 echo "You can rerun this without any issues."
6 echo -----
7 if [ -d ".env" ];
8 then
9   echo "Enabling virtual env"
10 else
11   echo "No Virtual env. Please run setup.sh first"
12 exit N
13 fi
14
15 # Activate virtual env
16 . .env/bin/activate
17 export ENV=stage
18 gunicorn main:app --worker-class:gevent --bind=127.0.0.1:8000 --workers=4
19 deactivate
20
21
22
```

Screenshot 2 (Middle): The title bar says "File Edit Selection Find View Goto Tools Project Preferences Help". The file path is "/code/work/flask-push/code/application/config.py". The code content is:

```
1 import os
2 basedir = os.path.abspath(os.path.dirname(__file__))
3
4 class Config():
5   DEBUG = False
6   SQLITE_DB_DIR = None
7   SQLALCHEMY_DATABASE_URI = None
8   SQLALCHEMY_TRACK_MODIFICATIONS = False
9   WTF_CSRF_ENABLED = False
10  SECURITY_TOKEN_AUTHENTICATION_HEADER = "Authentication-Token"
11  CELERY_BROKER_URL = "redis://localhost:6379/1"
12  CELERY_RESULT_BACKEND = "redis://localhost:6379/2"
13
14 class LocalDevelopmentConfig(Config):
15   SQLITE_DB_DIR = os.path.join(basedir, "..db_directory")
16   SQLALCHEMY_DATABASE_URI = "sqlite:///{} + os.path.join(SQLITE_DB_DIR, "test.db")
17   DEBUG = True
18   SECRET_KEY = "ash ah secret"
19   SECURITY_PASSWORD_HASH = "bcrypt"
20   SECURITY_PASSWORD_SALT = "really super secret" # Read from ENV in your config
21   SECURITY_REGISTERABLE = True
22   SECURITY_CONFIRMABLE = False
23   SECURITY_SEND_REGISTER_EMAIL = False
24   SECURITY_UNAUTHORIZED_VIEW = None
25   WTF_CSRF_ENABLED = False
26   CELERY_BROKER_URL = "redis://localhost:6379/1"
27   CELERY_RESULT_BACKEND = "redis://localhost:6379/2"
```

Screenshot 3 (Bottom): The title bar says "File Edit Selection Find View Goto Tools Project Preferences Help". The file path is "/code/work/flask-push/code/application/config.py". The code content is identical to Screenshot 2.

```

code/work/flask-push/code/application/config.py (flask-push) - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
    * iit_code
        * iit_env
        * iit_gunicorn_
        * iit_application
            * iit_gunicorn_
                - __init__.py
                - config.py
            - config.py
        - controllers.py
        - database.py
        - models.py
        - tasks.py
        - validation.py
        - views.py
    * iit_db_directory
    * iit_static
    * iit_templates
        - celerybeat_schedule
        - debug.log
        - local_beat.sh
        - local_gunicorn.sh
        - local_uwsgi.sh
        - local_workers.sh
        - main.py
        - README.md
        - requirements.txt
        - README.md
Source: 4 Python
14+ class LocalDevelopmentConfig(Config):
15     SQLITE_DB_DIR = os.path.join(basedir, "../db directory")
16     SQLALCHEMY_DATABASE_URI = "sqlite:///{} + os.path.join(SQLITE_DB_DIR, "te"
17     DEBUG = True
18     SECRET_KEY = "ash ah secre"
19     SECURITY_PASSWORD_HASH = "bcrypt"
20     SECURITY_PASSWORD_SALT = "really super secret" # Read from ENV in your ca
21     SECURITY_REGISTERABLE = True
22     SECURITY_CONFIRMABLE = False
23     SECURITY_SEND_REGISTER_EMAIL = False
24     SECURITY_UNAUTHORIZED_VIEW = None
25     WTF_CSRF_ENABLED = False
26     CELERY_BROKER_URL = "redis://localhost:6379/1"
27     CELERY_RESULT_BACKEND = "redis://localhost:6379/2"
28
29+ class StageConfig(Config):
30     SQLITE_DB_DIR = os.path.join(basedir, "../db directory")
31     SQLALCHEMY_DATABASE_URI = "sqlite:///{} + os.path.join(SQLITE_DB_DIR, "te"
32     DEBUG = True
33     SECRET_KEY = "ash ah secre"
34     SECURITY_PASSWORD_HASH = "bcrypt"
35     SECURITY_PASSWORD_SALT = "really super secret" # Read from ENV in your ca
36     SECURITY_REGISTERABLE = True
37     SECURITY_CONFIRMABLE = False
38     SECURITY_SEND_REGISTER_EMAIL = False
39     SECURITY_UNAUTHORIZED_VIEW = None
40     WTF_CSRF_ENABLED = False
41     CELERY_BROKER_URL = "redis://localhost:6379/1"
42     CELERY_RESULT_BACKEND = "redis://localhost:6379/2"
43     REDIS_URL = "redis://localhost:6379"
44

```

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)

The image shows two screenshots of a Sublime Text editor window. Both screenshots display the same Python code for a Flask application, specifically the file `main.py`. The code defines a `create_app()` function that initializes a Flask application and sets up various configurations based on the environment variable `ENV`.

```
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
• iit-task-push
• all code
• all env
• all _pycache_
• all application
• all __init__.py
• all api
• all config
• all controllers
• all database
• all models
• all tasks
• all static
• all templates
□ celerybeat_schedule
□ devlog.log
▷ local_beat.sh
▷ local_gunicorn.sh
▷ local_nsqd.sh
▷ local_redis.sh
▷ local_workers.sh
• all .gitignore
• requirements.txt
• README.md
• main.py
• tests.py
• validation.py
• workers.py
• db directory
• static
• templates
• celerybeat_schedule
• devlog.log
▷ local_beat.sh
▷ local_gunicorn.sh
▷ local_nsqd.sh
▷ local_redis.sh
▷ local_workers.sh
• main.py
• README.md
• requirements.txt
• README.md

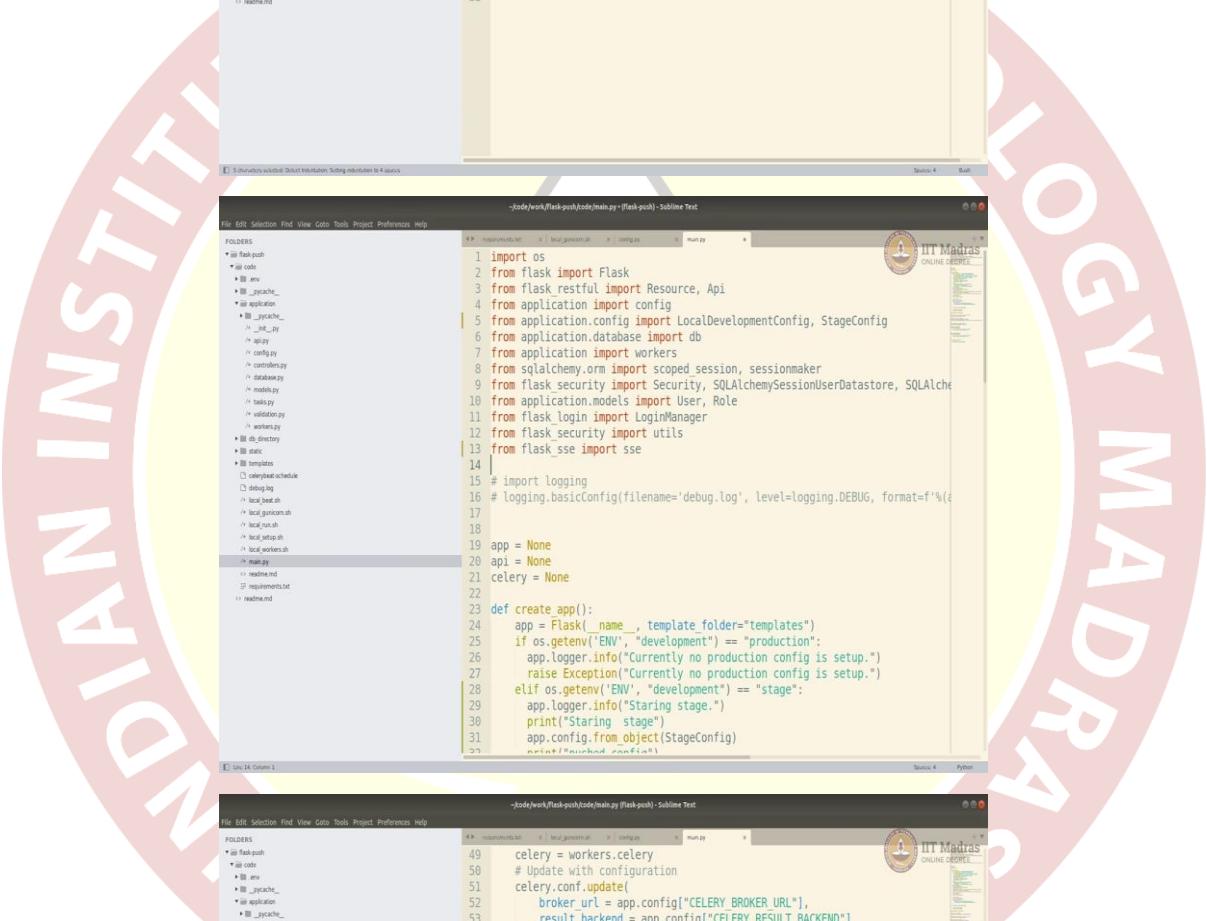
Line 3, Column 67
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
• iit-task-push
• all code
• all env
• all _pycache_
• all application
• all __init__.py
• all api
• all config
• all controllers
• all database
• all models
• all tasks
• all static
• all templates
□ celerybeat_schedule
▷ local_beat.sh
▷ local_gunicorn.sh
▷ local_nsqd.sh
▷ local_redis.sh
▷ local_workers.sh
• all .gitignore
• requirements.txt
• README.md
• main.py
• tests.py
• validation.py
• workers.py
• db directory
• static
• templates
• celerybeat_schedule
• devlog.log
▷ local_beat.sh
▷ local_gunicorn.sh
▷ local_nsqd.sh
▷ local_redis.sh
▷ local_workers.sh
• main.py
• README.md
• requirements.txt
• README.md
```

The code includes logic to handle different environments:

- If `ENV` is "development":
 - Imports `User` and `Role` from `application.models`.
 - Imports `LoginManager` from `flask_login`.
 - Imports `utils` from `flask_security`.
 - Logs a message indicating "Starting Local Development".
 - Prints "Starting Local Development".
 - Configures the app using `LocalDevelopmentConfig`.
 - Initializes the database with `db.init_app(app)`.
 - Registers a context processor to push the user data store.
 - Logs a message indicating "App setup complete".
 - Initializes the session maker with `SQLAlchemySessionUserDatastore(db.session, User, Role)`.
- If `ENV` is "stage":
 - Logs a message indicating "Starting stage".
 - Prints "Starting stage".
 - Configures the app using `StageConfig`.
 - Prints "pushed config".
- Otherwise:
 - Logs a message indicating "Starting Local Development".
 - Prints "Starting Local Development".
 - Configures the app using `LocalDevelopmentConfig`.
 - Initializes the database with `db.init_app(app)`.
 - Registers a context processor to push the user data store.
 - Logs a message indicating "App setup complete".
 - Initializes the session maker with `SQLAlchemySessionUserDatastore(db.session, User, Role)`.

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)



The image shows three screenshots of a Sublime Text editor window displaying code for a Flask application. The code is organized into three files: `local_unicorn.sh`, `main.py`, and `main.py`.

File 1: local_unicorn.sh

```
#!/bin/sh
echo "=====
Welcome 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."
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
```

File 2: main.py

```
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, SQLAlchemyUserDatastore, SQLAlchemy
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'%(asctime)s - %(name)s - %(levelname)s - %(message)s')
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("Starting stage.")
        print("Starting stage")
        app.config.from_object(StageConfig)
    else:
        app.config.from_object(LocalDevelopmentConfig)

    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
```

File 3: main.py

```
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
    return render_template('404.html'), 404
```

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)

The image consists of two screenshots. The top screenshot is a web browser displaying a Flask application's documentation on 'Modular Applications with Blueprints'. The page includes a navigation sidebar with links like 'Overview', 'Why Blueprints?', 'The Concept of Blueprints', etc., and a main content area with sections on 'Why Blueprints?' and 'The Concept of Blueprints'. The bottom screenshot shows a Sublime Text editor with an open file named 'controllers.py'. The code in the file is as follows:

```

1 from flask import Flask, request
2 from flask import render_template
3 from flask import current_app as app
4 from application.models import Article
5 from application import tasks
6 from flask_security import login_required, roles_accepted, roles_required
7 from datetime import datetime
8
9 @app.route("/", methods=["GET", "POST"])
10 def articles():
11     app.logger.info("Inside get all articles using info")
12     articles = Article.query.all()
13     app.logger.debug("Inside get all articles using debug")
14     return render_template("articles.html", articles=articles)
15
16 @app.route("/articles_by<user_name>", methods=["GET", "POST"])
17 @login_required
18 def articles_by_author(user_name):
19     articles = Article.query.filter(Article.authors.any(username=user_name))
20     return render_template("articles_by_author.html", articles=articles, user=user)
21
22
23 @app.route("/article_like<article_id>", methods=["GET", "POST"])
24 def like(article_id):
25     print(article_id)
26     return "OK", 200
27
28
29 @app.route("/feedback", methods=["GET", "POST"])
30 def feedback():
31     if request.method == "GET":

```



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)

The image consists of three vertically stacked screenshots of a Sublime Text editor window, showing code snippets and file explorers.

Screenshot 1: The top screenshot shows the main code editor with the file `main.py` open. The code is a Python script for a Celery application, defining tasks, configuration, and API endpoints. It includes imports for `workers`, `celery`, and `application`. The code handles streaming requests and registers blueprints for article likes and test APIs.

```
49 celery = workers.celery
50 # Update with configuration
51 celery.conf.update(
52     broker_url = app.config["CELERY_BROKER_URL"],
53     result_backend = app.config["CELERY_RESULT_BACKEND"]
54 )
55
56 celery.Task = workers.ContextTask
57
58 app.app_context().push()
59 return app, api, celery
60
61 app,api,celery = create_app()
62
63 # This is for streaming
64 app.register_blueprint(ss, url_prefix='/stream')
65
66 # Import all the controllers so they are loaded
67 from application.controllers import *
68
69 # Add all restful controllers
70 from application.api import ArticleLikesAPI
71 api.add_resource(ArticleLikesAPI, "/api/article_likes/")
72
73
74 from application.api import TestAPI
75 api.add_resource(TestAPI, "/api/test")
76
77 @app.errorhandler(404)
78 def page_not_found(e):
79     # note that we set the 404 status explicitly
80     return render_template('404.html'), 404
```

Screenshot 2: The middle screenshot shows the file explorer sidebar with the file `main.py` selected. The sidebar lists various files and folders such as `celery.py`, `config.py`, `controllers.py`, `database.py`, `models.py`, `tasks.py`, `validation.py`, and `workers.py`.

Screenshot 3: The bottom screenshot shows the file explorer sidebar with the file `show_updates.html` selected. The sidebar lists files like `articles.html`, `articles_by_author.html`, `articles_by_id.html`, `articles_by_title.html`, `feedback.html`, and `thank_you.html`.

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)

The image shows two screenshots of a Sublime Text editor window. Both screenshots display the same code for a file named 'show_update.html' located in a 'flask-push' directory. The code is an HTML file with embedded JavaScript to handle an SSE event source.

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript" src="{{ url_for('static', filename='js/show_alerts.js') }}></script>
<script type="text/javascript" src="{{ url_for('static', filename='vue/vue.min.js') }}></script>
</head>
<title>Flask-SSE Quickstart</title>
<body>
<h1>Flask-SSE Quickstart</h1>
<script>
var source = new EventSource("{{ url_for('sse.stream') }}");
source.addEventListener('greeting', function(event) {
    var data = JSON.parse(event.data);
    alert("The server says " + data.message);
}, false);
source.addEventListener('error', function(event) {
    alert("Failed to connect to event stream. Is Redis running?");
}, false);
</script>
</body>
</html>

```

The bottom screenshot shows the status bar indicating 'Line 23 Col 8: Syntax Error'. The file path shown in the title bar is '/code/work/flask-push/code/templates/show_update.html'.

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)

The screenshot shows a browser window with two tabs. The top tab is titled 'Using server-sent events' and displays a live event stream. The bottom tab is titled 'Flask-SSE Quickstart' and shows a code editor with a Python file named 'show_updates.html'. The browser interface includes a navigation bar, a search bar, and a toolbar with various icons.

Event Stream Content:

```
event: userconnect
data: {"username": "bobby", "time": "02:33:48"}  
b  
event: usermessage
data: {"username": "bobby", "time": "02:34:11", "text": "Hi everyone."}  
event: userdisconnect
data: {"username": "bobby", "time": "02:34:23"}  
event: usermessage
data: {"username": "sean", "time": "02:34:36", "text": "Bye, bobby."}
```

Mixing and matching:
You don't have to use just unnamed messages or typed events; you can mix them together in a single event stream.

```
event: userconnect
data: {"username": "bobby", "time": "02:33:48"}  
data: Here's a system message of some kind that will get used
data: to accomplish some task.  
event: usermessage
data: {"username": "bobby", "time": "02:34:11", "text": "Hi everyone."}
```

Browser compatibility:

EventSource

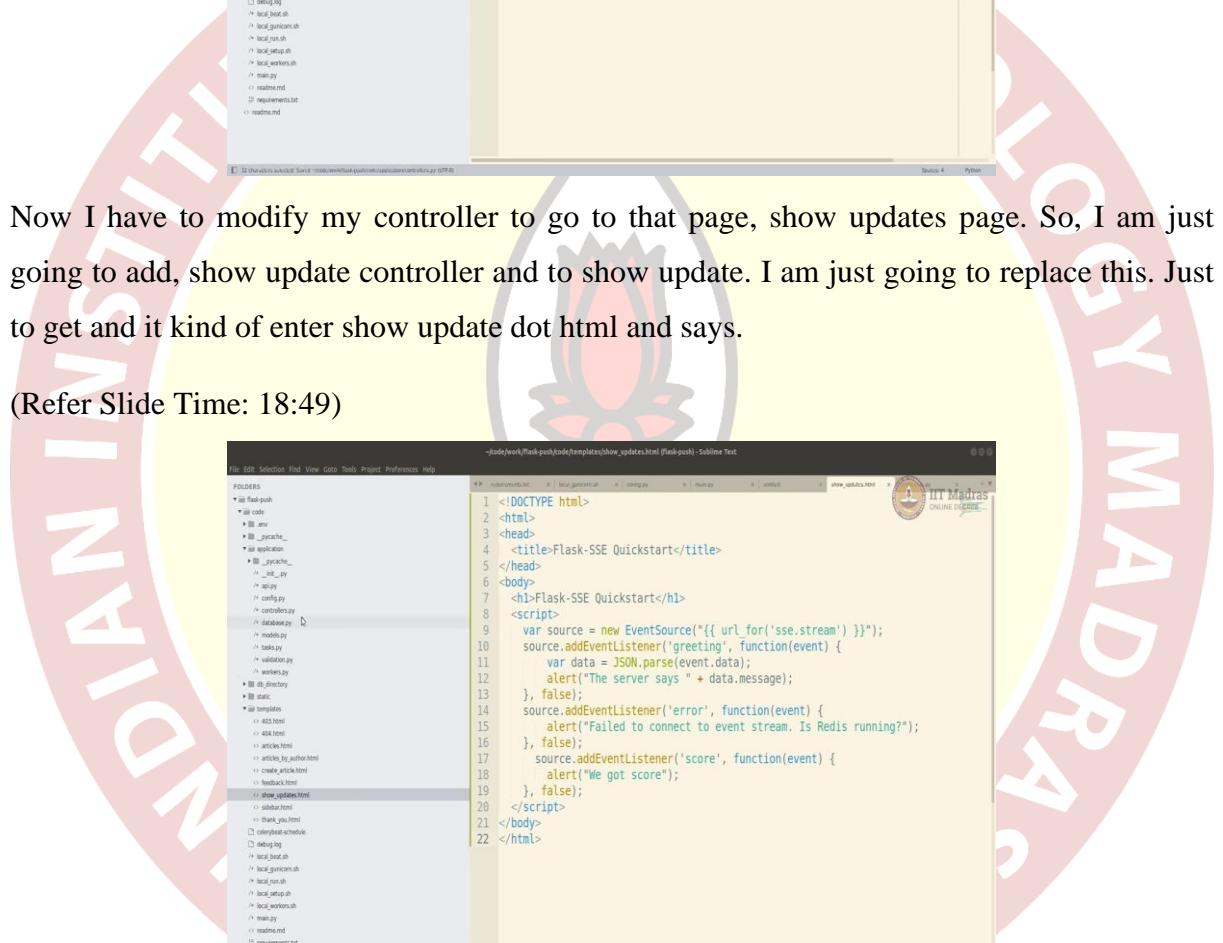
Code Editor Content (show_updates.html):

```
<!DOCTYPE html>
<html>
<head>
<title>Flask-SSE Quickstart</title>
</head>
<body>
<h1>Flask-SSE Quickstart</h1>
<script>
var source = new EventSource("{ url_for('sse.stream') }");
source.addEventListener('greeting', function(event) {
  var data = JSON.parse(event.data);
  alert("The server says " + data.message);
}, false);
source.addEventListener('error', function(event) {
  alert("Failed to connect to event stream. Is Redis running?");
}, false);
source.addEventListener('score', function(event) {
  alert("We got score!");
}, false);
</script>
</body>
</html>
```

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)

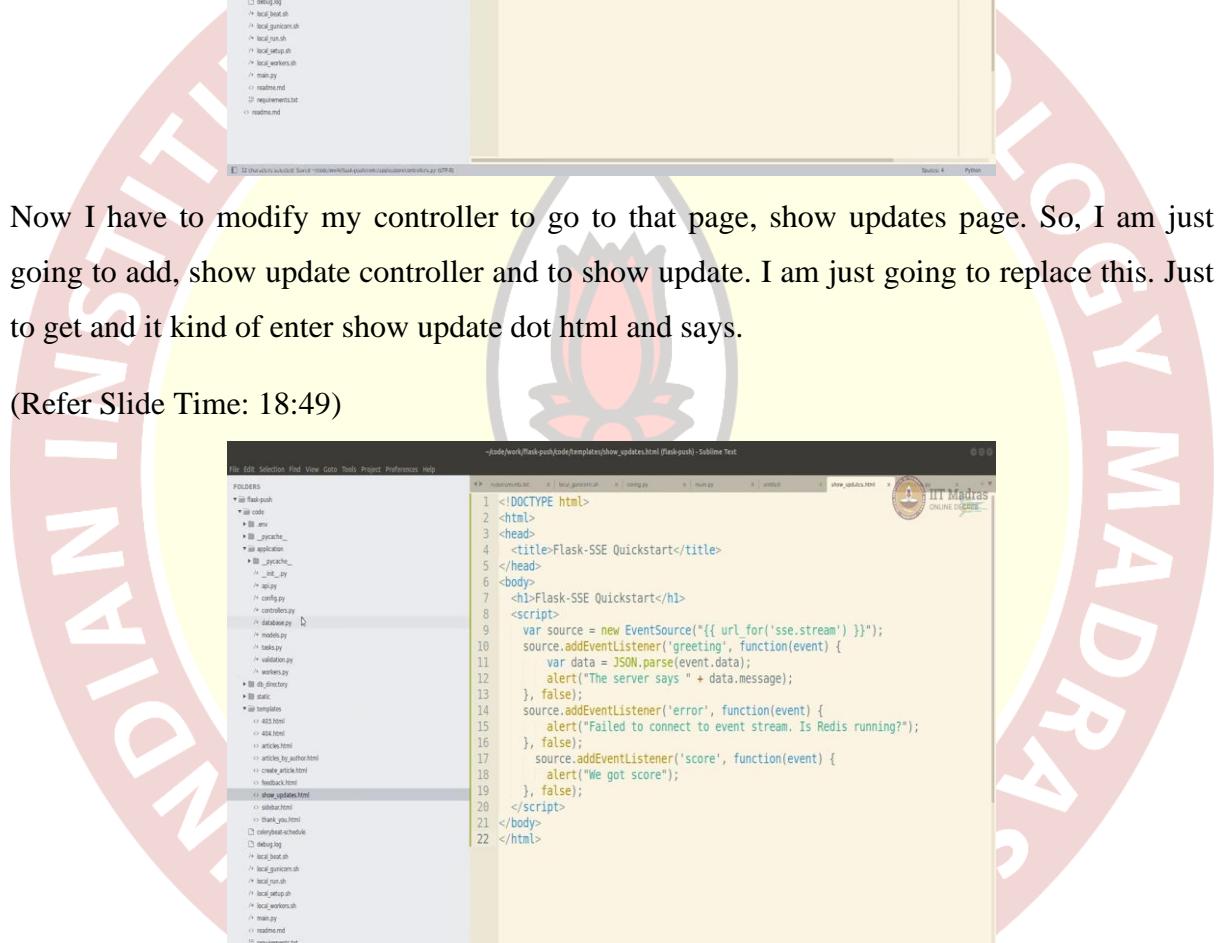


```
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
    ▾ iit
        ▾ flask
            ▾ __init__.py
            ▾ __main__.py
            ▾ __appcache_
            ▾ __application
            ▾ __http_.py
            ▾ __jinja2__
            ▾ __werkzeug__
            ▾ controllers.py
            ▾ database.py
            ▾ models.py
            ▾ tasks.py
            ▾ validation.py
            ▾ workers.py
        ▾ static
            ▾ __templates
                ▾ 403.html
                ▾ 404.html
                ▾ article.html
                ▾ article_by_author.html
                ▾ create_article.html
                ▾ index.html
                ▾ show_update.html
                ▾ update.html
                ▾ thank_you.html
            ▾ __staticfileschedule
            ▾ debug.log
            ▾ local_host.sh
            ▾ local_gunicorn.sh
            ▾ local_uwsgi.sh
            ▾ local_workers.sh
            ▾ main.py
            ▾ README.md
            ▾ requirements.txt
            ▾ README.md

50 def create_article():
51     if request.method == "GET":
52         return render_template("create_article.html", error=None)
53
54
55 @app.route("/hello", methods=["GET", "POST"])
56 def hello():
57     job = tasks.print_current_time_job.apply_async(username)
58     result = job.wait()
59     return str(result), 200
60
61
62 @app.route("/show_updates", methods=["GET"])
63 def show_updates():
64     return render_template("show_updates.html", error=None)
65
66
```

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)

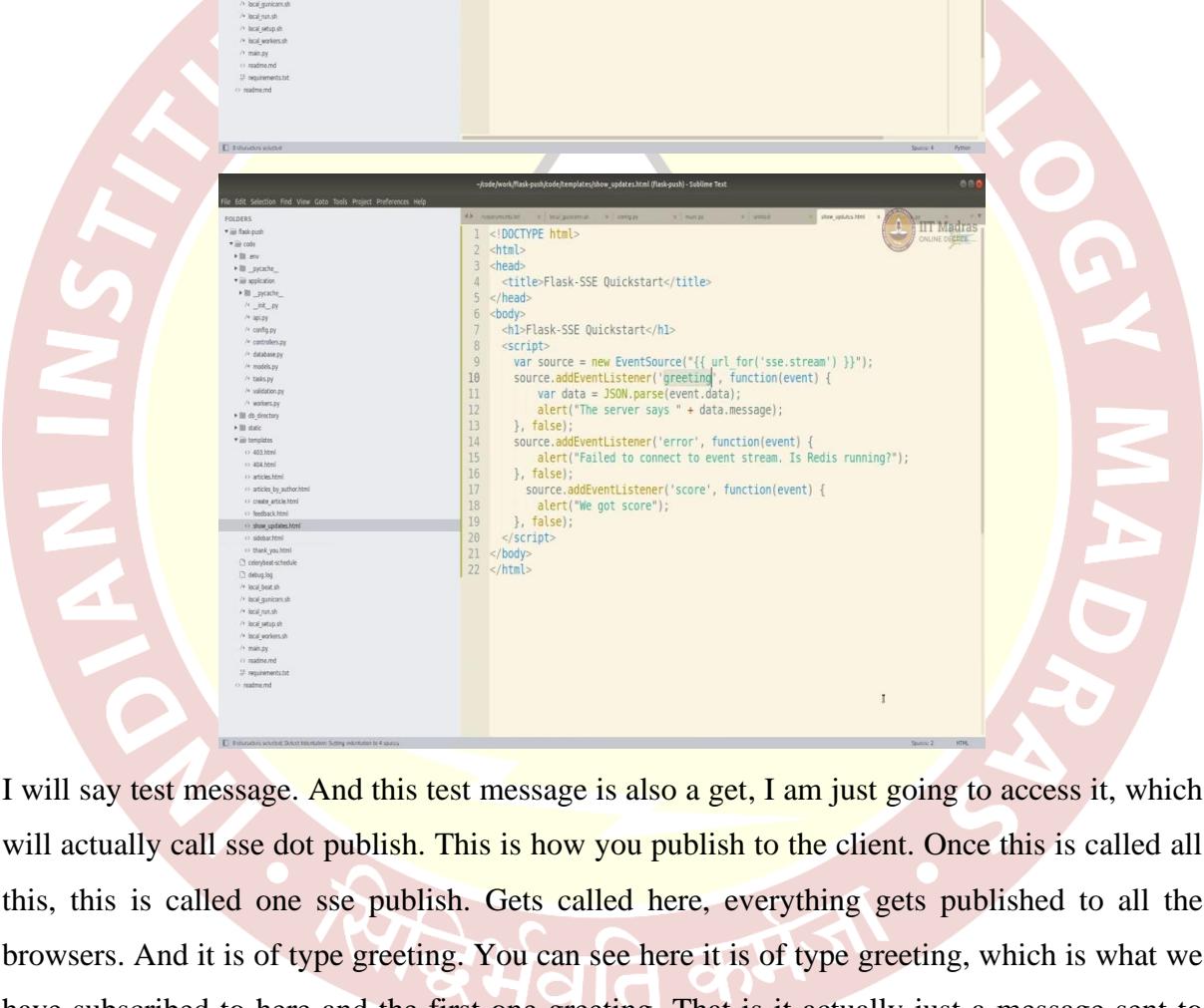


```
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
    ▾ iit
        ▾ flask
            ▾ __init__.py
            ▾ __main__.py
            ▾ __appcache_
            ▾ __application
            ▾ __http_.py
            ▾ __jinja2__
            ▾ __werkzeug__
            ▾ controllers.py
            ▾ database.py
            ▾ models.py
            ▾ tasks.py
            ▾ validation.py
            ▾ workers.py
        ▾ static
            ▾ __templates
                ▾ 403.html
                ▾ 404.html
                ▾ article.html
                ▾ article_by_author.html
                ▾ create_article.html
                ▾ index.html
                ▾ show_update.html
                ▾ update.html
                ▾ thank_you.html
            ▾ __staticfileschedule
            ▾ debug.log
            ▾ local_host.sh
            ▾ local_gunicorn.sh
            ▾ local_uwsgi.sh
            ▾ local_workers.sh
            ▾ main.py
            ▾ README.md
            ▾ requirements.txt
            ▾ README.md

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Flask-SSE Quickstart</title>
5   </head>
6   <body>
7     <h1>Flask-SSE Quickstart</h1>
8     <script>
9       var source = new EventSource("{ url_for('sse.stream') }");
10      source.addEventListener('greeting', function(event) {
11        var data = JSON.parse(event.data);
12        alert("The server says " + data.message);
13      }, false);
14      source.addEventListener('error', function(event) {
15        alert("Failed to connect to event stream. Is Redis running?");
16      }, false);
17      source.addEventListener('score', function(event) {
18        alert("We got score");
19      }, false);
20    </script>
21  </body>
22 </html>
```

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)



The image shows two screenshots of Sublime Text editors. The top screenshot displays a Python file named 'controllers.py' with code related to Flask routes and asynchronous tasks. The bottom screenshot displays an HTML file named 'show_updates.html' containing an EventSource script that connects to a SSE stream and alerts the user with received messages.

controllers.py

```
56 def hello():
57     job = tasks.print_current_time.job.apply_async(username)
58     result = job.wait()
59     return str(result), 200
60
61
62 @app.route("/show_updates", methods=["GET"])
63 def show_updates():
64     return render_template("show_updates.html", error=None)
65
66
67 @app.route("/test_send_message", methods=["GET"])
68 def test_send_message():
69     sse.publish({"message": "Hello!"}, type='greeting')
70     return "Message sent!"
71
```

show_updates.html

```
<!DOCTYPE html>
<html>
<head>
<title>Flask-SSE Quickstart</title>
</head>
<body>
<h1>Flask-SSE Quickstart</h1>
<script>
var source = new EventSource("{ url_for('sse.stream') }");
source.addEventListener('greeting', function(event) {
    var data = JSON.parse(event.data);
    alert("The server says " + data.message);
}, false);
source.addEventListener('error', function(event) {
    alert("Failed to connect to event stream. Is Redis running?");
}, false);
source.addEventListener('score', function(event) {
    alert("We got score");
}, false);
</script>
</body>
</html>
```

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)

The image shows two screenshots of a Sublime Text editor window. The top screenshot displays a terminal session showing the directory structure of a Flask application and its contents. The bottom screenshot shows the code for a Python controller file named 'controllers.py'. The code includes routes for creating articles, sending test messages, and showing updates, along with corresponding templates and logic.

```
-code@uma:~/code/work/flask-push/code$ ls
application  local_beat.sh  main.py      static
celerybeat-schedule  local_run.sh  __pycache__  templates
db_directory  local_workers.sh  README.md
debug.log    local_workers.sh  requirements.txt
thej@uma:~/code/work/flask-push/code$ ls -a
..          .env        main.py      static
application  local_beat.sh  __pycache__  templates
celerybeat-schedule  local_run.sh  README.md
db_directory  local_setup.sh  requirements.txt
thej@uma:~/code/work/flask-push/code$ 

thej@uma:~/code/work/flask-push/code$ cd application
thej@uma:~/code/work/flask-push/code$ ls
__init__.py  celerybeat_schedule  debug.log  local_beat.sh
__main__.py  db_directory        README.md  requirements.txt
app.py      local_run.sh       requirements.txt
config.py  local_setup.sh     static
controller.py  local_workers.sh
database.py  logs
models.py   static
utils.py    templates
views.py   workers.py

thej@uma:~/code/work/flask-push/code$ cd ..
thej@uma:~/code/work/flask-push$ ls
code  flask-push

File Edit Selection Prod View Goto Tools Project Preferences Help
File Edit Selection Prod View Goto Tools Project Preferences Help
thej@uma:~/code/work/flask-push/code/application/controllers.py [Task-push] - Sublime Text
52     return render_template("thank-you.html")
53
54
55 @app.route("/create_article", methods=["GET", "POST"])
56 @login_required
57 @roles_required('admin')
58 def create_article():
59     if request.method == "GET":
60         return render_template("create_article.html", error=None)
61
62
63 @app.route("/test_send_message", methods=["GET", "POST"])
64 def test_send_message():
65     sse.publish({"message": "Hello!", type='greeting'})
66     return "Message sent check the other browser!"
67
68
69 @app.route("/show_updates", methods=["GET"])
70 def show_updates():
71     return render_template("show_updates.html", error=None)
72
```

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)

```
the@uma:~/code/work/flask-push/code
```

```
the@uma:~/code/work/flask-push/code
```

```
the@uma:~/code/work/flask-push/code
```

```
bash: .bourne-app: No such file or directory
bash: .bashpromptrc: No such file or directory
the@uma:~/code/work/flask-push/codes$ ls
application          db_directory local_beat.sh    local_run.sh  local_workers.sh  _pycache_ requirements.txt templates
celerybeat-schedule   debug.log    local_gunicorn.sh local_setup.sh main.py
the@uma:~/code/work/flask-push/codes$ sh local_beat.sh
=====
Welcome 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
Starting Local Development
pushed config
sh db
sh db
sh Init complete
Create app complete
celery beat v5.2.3 (dawn-chorus) is starting.
=====
localTime -> 2022-02-23 23:31:25
Configuration ->
  . broker -> redis://localhost:6379/1
  . loader -> celery.loaders.app.AppLoader
  . scheduler -> celery.beat.PersistentScheduler
  . db -> celerybeat-schedule
  . logfile -> [stderr]@INFO
  . maxinterval -> 1.00 second [1s]
[2022-02-23 23:31:25,239: INFO/MainProcess] beat: Starting...
```



```
the@uma:~/code/work/flask-push/code
```

```
the@uma:~/code/work/flask-push/code
```

```
the@uma:~/code/work/flask-push/code
```

```
bash: .bourne-app: No such file or directory
bash: .bashpromptrc: No such file or directory
the@uma:~/code/work/flask-push/codes$ sh local_workers.sh
=====
Welcome 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
Starting Local Development
pushed config
sh DB
sh DB
sh 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:0x7f7fe915cb90
...     .> 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.
```



```
Flask-SSE Quickstart
```

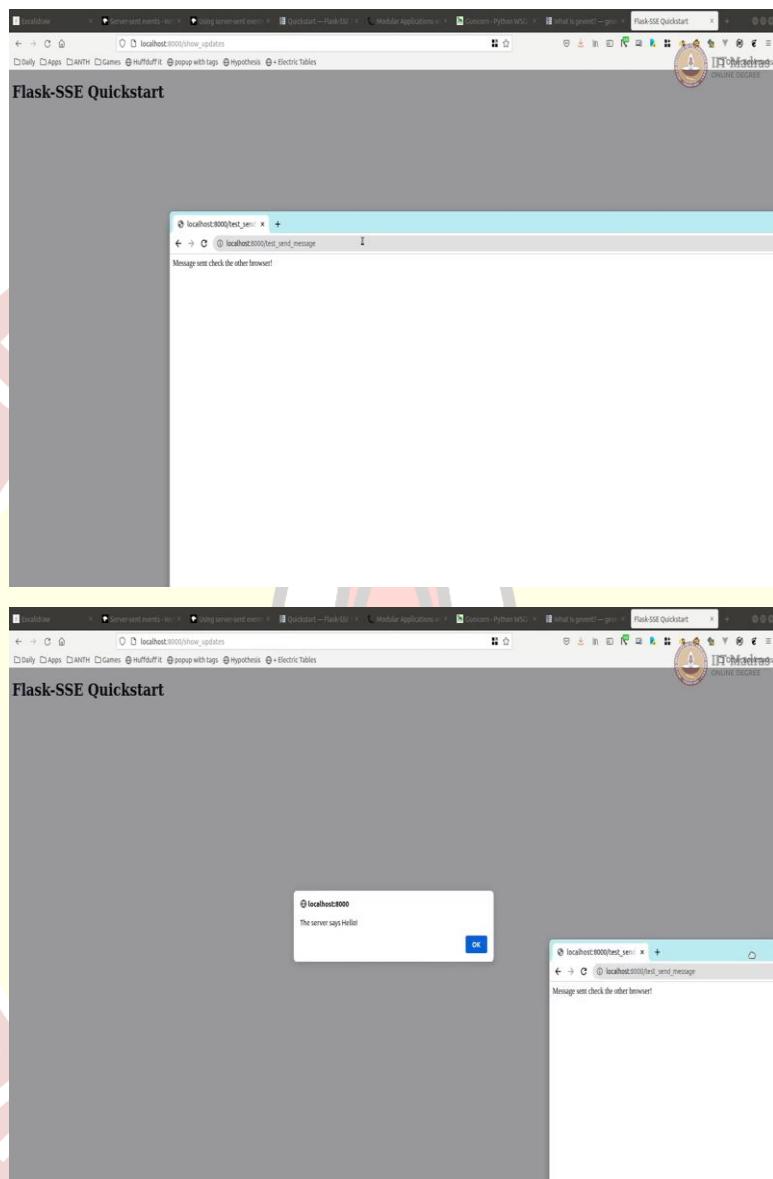
```
localhost:8000/show_updates
```

```
Daily Apps ANTH HuHuFit popup-with-tags Hypothesis Electric Tables
```

```
Flask-SSE Quickstart
```

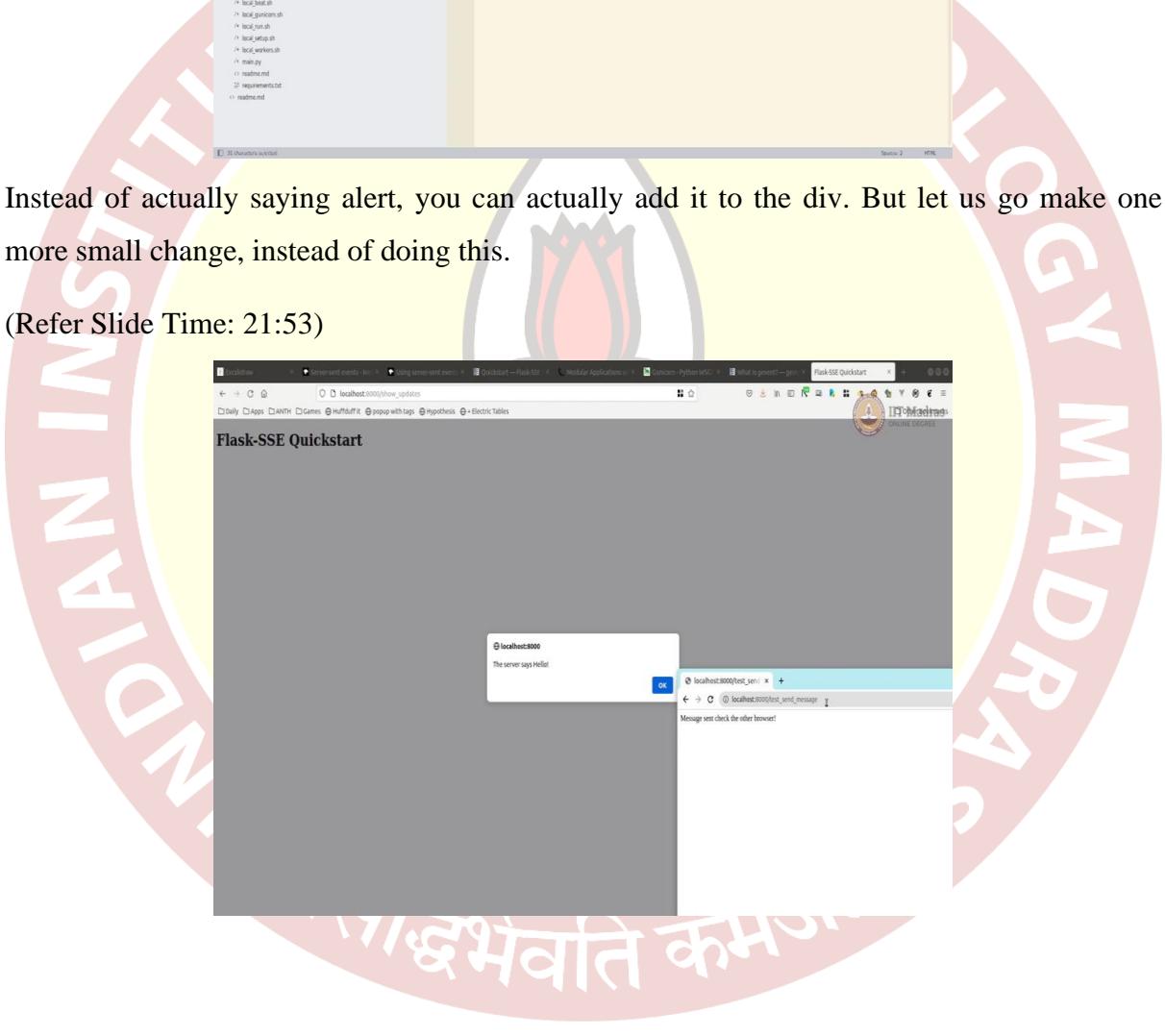
Let us start this. I have also started my celery beat, and celery worker. Now, if I go to this place, it has flask-sse quick start. Now we want to publish something to it.

(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)

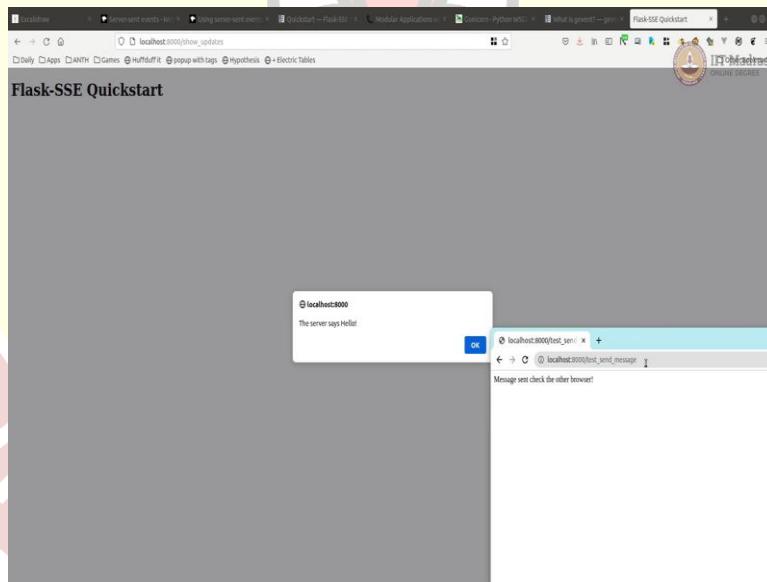


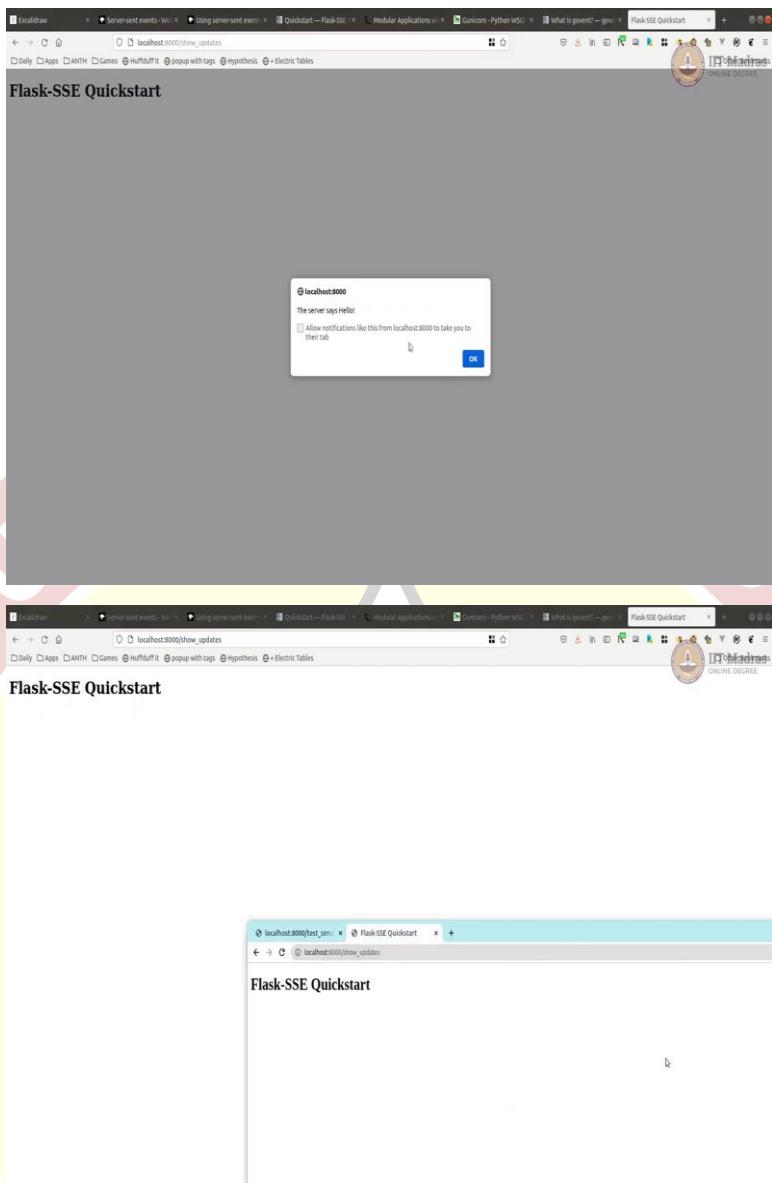
A screenshot of Sublime Text showing the file `show_updates.html`. The code is a simple Flask-SSE application:

```
<!DOCTYPE html>
<html>
<head>
<title>Flask-SSE Quickstart</title>
</head>
<body>
<h1>Flask-SSE Quickstart</h1>
<script>
var source = new EventSource("{ url_for('sse.stream') }");
source.addEventListener('greeting', function(event) {
    var data = JSON.parse(event.data);
    alert("The server says " + data.message);
}, false);
source.addEventListener('error', function(event) {
    alert("Failed to connect to event stream. Is Redis running?");
}, false);
source.addEventListener('score', function(event) {
    alert("We got score");
}, false);
</script>
</body>
</html>
```

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.

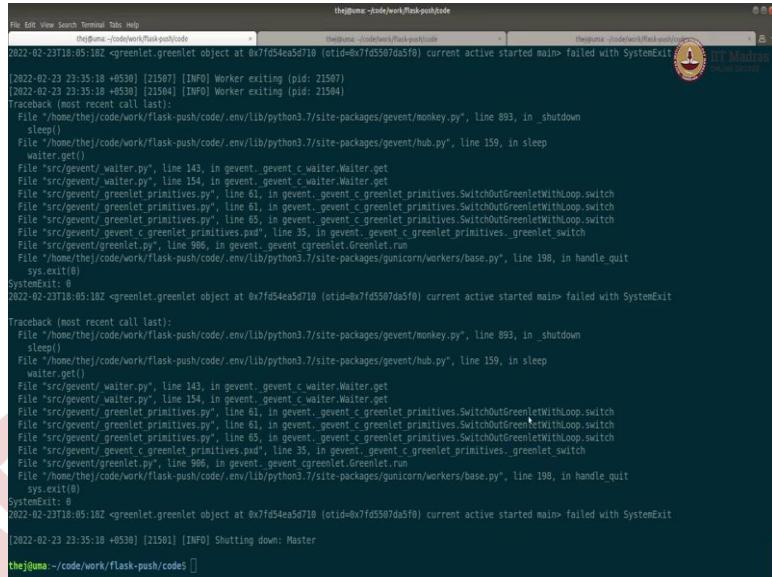
(Refer Slide Time: 21:53)





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)



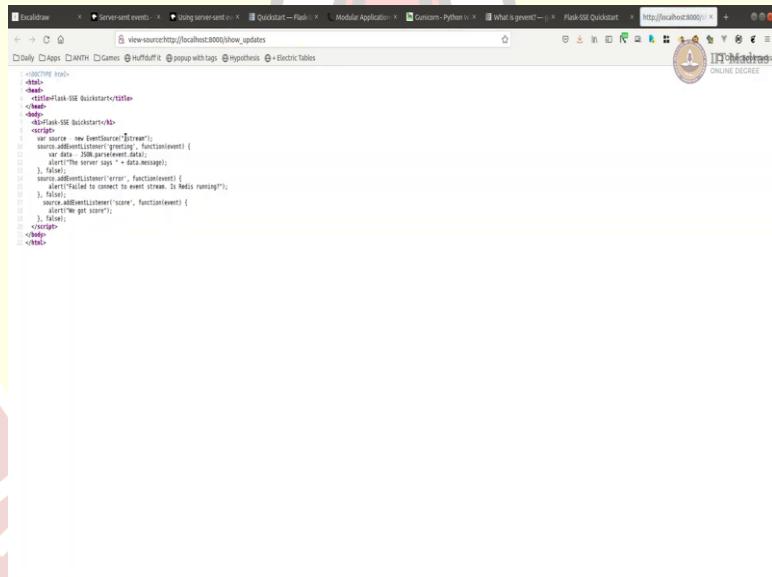
The terminal window shows several tabs open, all displaying the same error message:

```
thej@juna:~/code/work/flask-push/code
thej@juna:~/code/work/flask-push/code
thej@juna:~/code/work/flask-push/code
```

Logs from 2022-02-23 23:35:18 +0530 [21507] [INFO] Worker exiting (pid: 21507)
[2022-02-23 23:35:18 +0530] [21504] [INFO] Worker exiting (pid: 21504)
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/walter.py", line 143, in gevent_c_walter.Walter.get
 File "/src/gevent/walter.py", line 154, in gevent_c_walter.Walter.get
 File "/src/gevent/greenlet_primitives.py", line 61, in gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
 File "/src/gevent/greenlet_primitives.py", line 61, in gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
 File "/src/gevent/greenlet_primitives.py", line 65, in gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
 File "/src/gevent/greenlet_primitives.py", line 35, in gevent_c_greenlet_primitives._greenlet_switch
 File "/src/gevent/greenlet.py", line 906, in gevent_c_greenlet.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/walter.py", line 143, in gevent_c_walter.Walter.get
 File "/src/gevent/walter.py", line 154, in gevent_c_walter.Walter.get
 File "/src/gevent/greenlet_primitives.py", line 61, in gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
 File "/src/gevent/greenlet_primitives.py", line 61, in gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
 File "/src/gevent/greenlet_primitives.py", line 65, in gevent_c_greenlet_primitives.SwitchOutGreenletWithLoop.switch
 File "/src/gevent/greenlet_primitives.py", line 35, in gevent_c_greenlet_primitives._greenlet_switch
 File "/src/gevent/greenlet.py", line 906, in gevent_c_greenlet.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
2022-02-23 23:35:18 +0530 [21503] [INFO] Shutting down: Master
[2022-02-23 23:35:18 +0530] [21503]

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)

The image consists of three vertically stacked screenshots of the Sublime Text code editor, showing different stages of a Vue.js application setup.

Screenshot 1: The file structure shows a directory named 'flask-push'. Inside it are several sub-directories like 'db', 'static', 'templates', and 'workers'. The 'templates' folder contains HTML files such as '403.html', '404.html', 'articles.html', 'article_by_author.html', 'create_article.html', 'feedback.html', 'show_update.html', 'thank_you.html', and 'update.html'. There are also configuration files like '.htaccess', 'config.py', 'database.py', 'models.py', 'tasks.py', 'validation.py', and 'workers.py'. A 'log' directory contains 'log_beat.sh', 'log_gunicorn.sh', 'log_rsys.sh', 'log_setup.sh', and 'log_workers.sh'. Other files include 'main.py', 'readme.md', 'requirements.txt', and 'readme.html'.

Screenshot 2: A file dialog is open in the center of the screen, titled 'Name: show_update_no.html'. It lists files from the 'templates' folder and other locations like 'Desktop', 'Documents', 'Downloads', 'Music', 'Pictures', 'Videos', 'work', 'software', 'backup', and 'code'. The 'show_update.html' file is selected.

Screenshot 3: The code editor window shows the contents of 'static/js/app.js'. The code defines a new Vue instance with an element selector '#app', template delimiters {}, and data properties including 'msg' set to 'Hello World from Vue!'.

Now, let us create another template, new file. We are going to use vue 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)

The image shows two screenshots of a Sublime Text editor. The top screenshot shows a file browser window with a search bar containing 'show alerts.js'. The bottom screenshot shows the code for 'show_alerts.js'.

```

1 var app = new Vue({
2   el: '#app',
3   template:
4     `

5       


6         <li v-for="message in messages"> {{message}} </li>
7       

8

`,
9   data: {
10     messages: []
11   },
12   mounted: function() {
13     source = new EventSource("/stream");
14     source.addEventListener('greeting', event => {
15       let data = JSON.parse(event.data);
16       this.messages.push(data.message)
17     }, false);
18   }
19 });
20

```

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)

The image shows two screenshots of a Sublime Text editor window. The top screenshot displays the contents of 'show_update.vue.html' with the following code:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
<head>
<title>Flask-SSE Quickstart</title>
</head>
<body>
<h1>Flask-SSE Quickstart</h1>
<div class="container" id="app">
</div>
<script type="text/javascript" src="{{ url_for('static', filename='js/show_alerts.js') }}></script>
</body>
</html>
```

The bottom screenshot displays the contents of 'show_alerts.js' with the following code:

```
var app = new Vue({
  el: '#app',
  template: `
    <div>
      <ul>
        <li v-for="message in messages"> {{message}} </li>
      </ul>
    </div>`,
  data: {
    messages: []
  },
  mounted: function() {
    source = new EventSource("/stream");
    source.addEventListener("greeting", event => {
      let data = JSON.parse(event.data);
      this.messages.push(data.message)
    }, false);
  }
});
```

A screenshot of a Sublime Text editor window showing a Python file named 'controllers.py'. The code implements several routes for a web application:

```

52     return render_template("thank-you.html")
53
54
55 @app.route("/create_article", methods=["GET", "POST"])
56 @login_required
57 @roles_required('admin')
58 def create_article():
59     if request.method == "GET":
60         return render_template("create_article.html", error=None)
61
62
63 @app.route("/test_send_message", methods=["GET", "POST"])
64 def test_send_message():
65     sse.publish({"message": "Hello!", type="greeting"})
66     return "Message sent check the other browser!"
67
68
69 @app.route("/show_updates", methods=["GET"])
70 def show_updates():
71     return render_template("show_updates.html", error=None)
72
73
74 @app.route("/show_updates_vue", methods=["GET"])
75 def show_updates_vue():
76     return render_template("show_updates_vue.html", error=None)

```

The left sidebar shows the project structure with files like 'db', 'models.py', 'tasks.py', 'validation.py', 'style.css', 'templates', and various HTML files.

And add everything that is required. I am going to import, it is a simple one, so I am going to import vue 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)

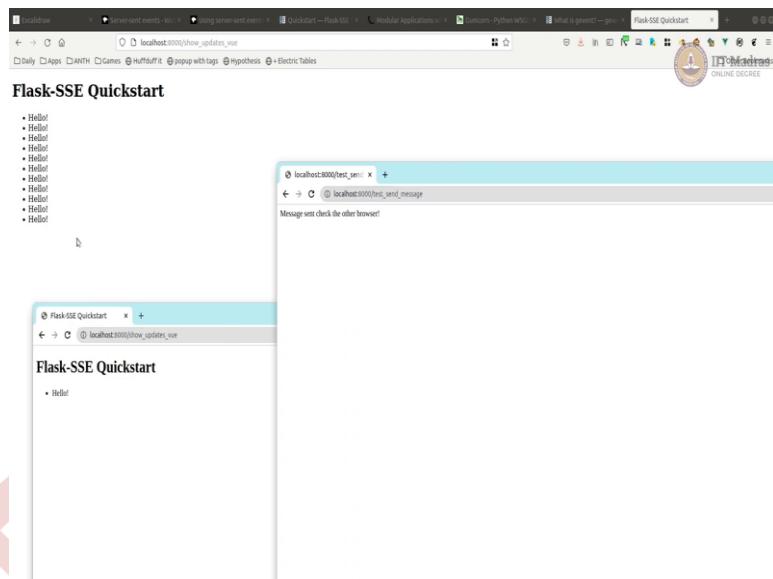
A terminal window showing the setup of a local virtual environment and the starting of a Celery beat process:

```

thej@name:~/code/work/Task-push/code$ bash setup.sh
thej@name:~/code/work/Task-push/code$ ls
application  db directory local_beat.sh  local_run.sh  local_workers.sh  __pycache__  requirements.txt  templates
Celerybeat-schedule  debug.log  local_gunicorn.sh  local_setup.sh  main.py  README.md  static
thej@name:~/code/work/Task-push/code$ sh local_beat.sh
[...]
Welcome 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
developing...
Starting Local Development
pushed config
DB Init
DB Init complete
Create app complete
celery beat v3.2.3 (dawn-chorus) is starting.
[...]
LocalTime -> 2022-02-23 23:31:25
Configuration ->
  . broker -> redis://localhost:6379/1
  . taskurer -> celery.loaders.app.AppLoader
  . scheduler -> celery.beat.PersistentScheduler
  . db -> celerybeat-schedule
  . logfile -> [stderr]@INFO
  . maxinterval -> 1.00 second (1s)
[2022-02-23 23:31:25,239: INFO/MainProcess] beat: Starting...

```

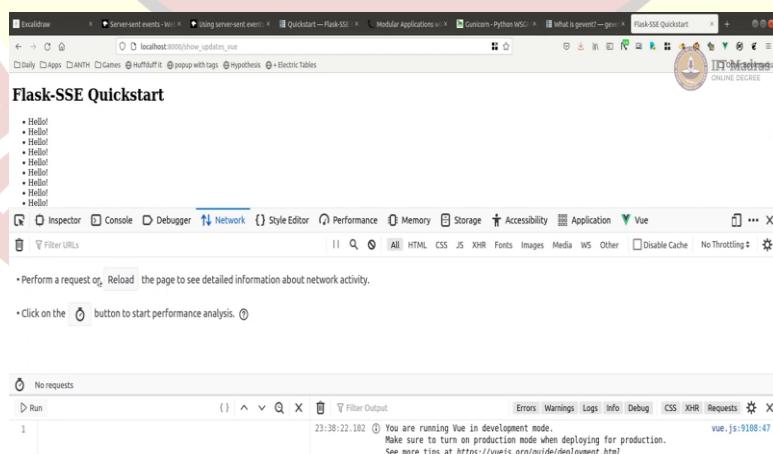


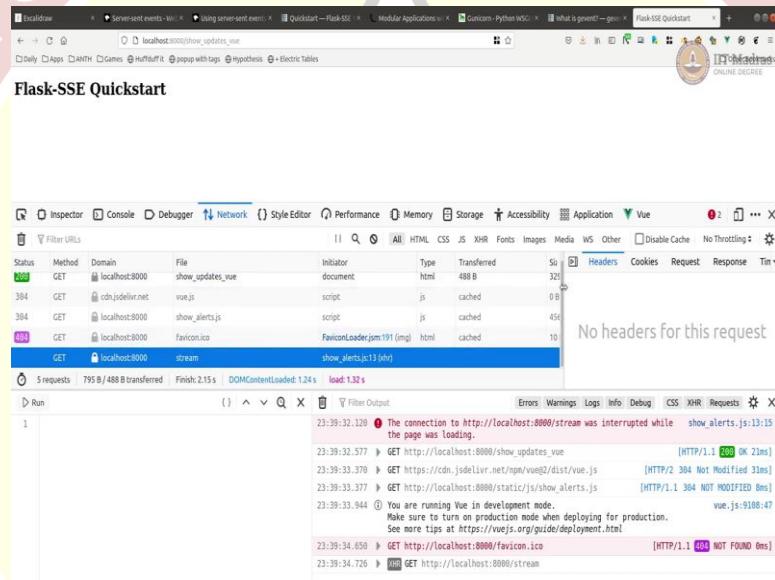
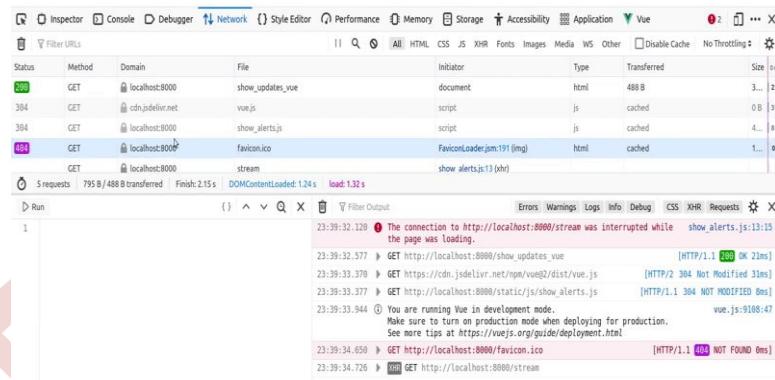
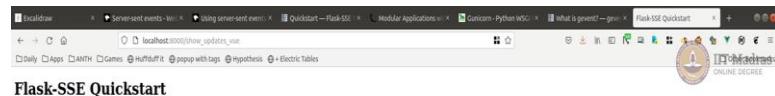


Let us start this again. This is running. This will start again. It will be Show update underscore vue. And now, if I have another browser, can you see? You can see how fast it gets. We will open one more. Open the same browser. Let us assume these are the two people who are watching the score.

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.

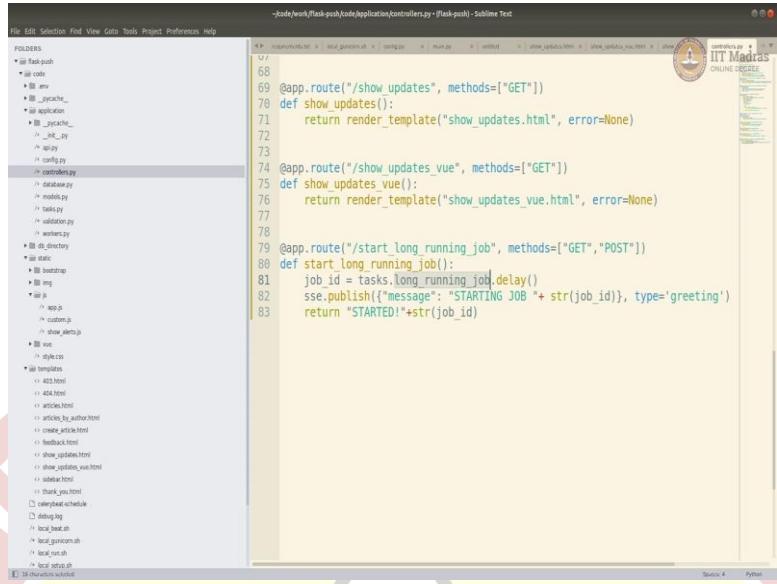
(Refer Slide Time: 26:51)





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)



```

File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
  ▪ iit_madras
    ▪ __init__.py
    ▪ __main__.py
    ▪ __pycache__
      ▪ __init__.py
      ▪ controllers.py
      ▪ database.py
      ▪ models.py
      ▪ tasks.py
      ▪ validation.py
      ▪ workers.py
    ▪ __init__.py
    ▪ static
      ▪ __init__.py
      ▪ img
      ▪ js
        ▪ app.js
        ▪ custom.js
        ▪ show_alerts.js
    ▪ __pycache__.py
    ▪ style.css
  ▪ templates
    ▪ 403.html
    ▪ 404.html
    ▪ article.html
    ▪ article_by_author.html
    ▪ create_article.html
    ▪ edit.html
    ▪ show_update.html
    ▪ show_update_vue.html
    ▪ update.html
    ▪ thank_you.html
  ▪ celerybeat_schedule
  ▪ debug.log
  ▪ local_beat.sh
  ▪ local_beat.sh.in
  ▪ local_setup.sh
  ▪ local_setup.sh.in
  ▪ requirements.txt
  ▪ static

File: controllers.py - (Task-push) - Sublime Text
  68
  69 @app.route("/show_updates", methods=["GET"])
  70 def show_updates():
  71     return render_template("show_updates.html", error=None)
  72
  73
  74 @app.route("/show_updates_vue", methods=["GET"])
  75 def show_updates_vue():
  76     return render_template("show_updates_vue.html", error=None)
  77
  78
  79 @app.route("/start_long_running_job", methods=["GET","POST"])
  80 def start_long_running_job():
  81     job_id = tasks.long_running_job.delay()
  82     sse.publish({"message": "STARTING JOB "+ str(job_id)}, type='greeting')
  83     return "STARTED!" +str(job_id)

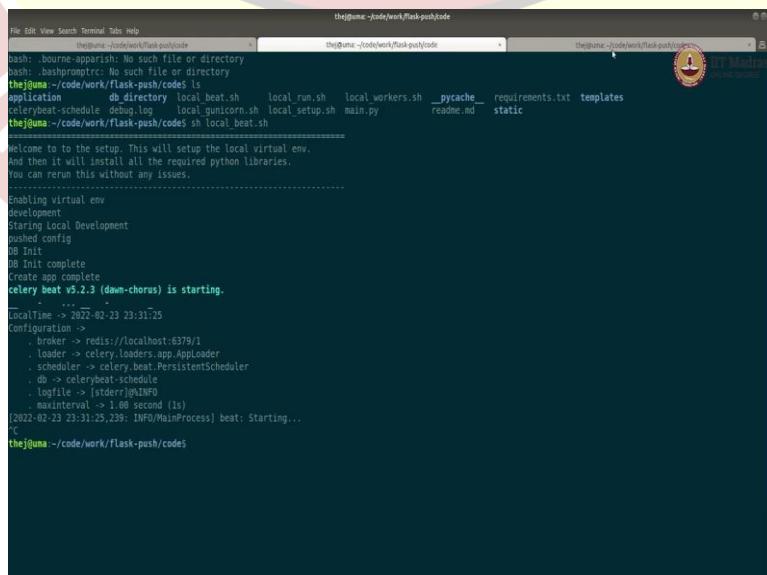
Status: 4 Python

```

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)



```

thej@uma:~/code/work/flask-push/code$ bash
thej@uma:~/code/work/flask-push/code$ cd codes
thej@uma:~/code/work/flask-push/codes$ ls
application  db directory local_beat.sh  local_run.sh  local_workers.sh  __pycache__  requirements.txt  templates
celerybeat-schedule  debug.log  local_unicorn.sh  local_setup.sh  main.py  readme.md
thej@uma:~/code/work/flask-push/codes$ sh local_beat.sh
Welcome 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
Starting Local Development
pushed config
Do Init
Do Init complete
Create app complete
celery beat v3.2.3 (down-chorus) is starting.

[2022-02-23 23:31:25,299: INFO/MainProcess] beat: Starting...
thej@uma:~/code/work/flask-push/code$ 

```

The image shows three vertically stacked screenshots of a Sublime Text code editor window, each displaying Python code related to a Flask application and Celery tasks.

Screenshot 1 (Top):

```

File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
    • iit_task_push
        • __init__.py
        • __env__
        • __pycache__
        • __application
            • __init__.py
            • __app__
            • __tasks__
            • __validation__
            • __utils__
        • __controllers__
        • __models__
        • __database__
        • __utils__
        • __tasks__
        • __validation__
        • __utils__
    • __dbs_directory
    • __static__
    • __bootstrap__
    • __img__
    • __js__
        • app.js
        • custom.js
        • show_alert.js
    • __css__
        • style.css
    • __templates__
        • 403.html
        • 404.html
        • article.html
        • articles_by_author.html
        • create_article.html
        • feedback.html
        • show_update.html
        • show_update_vue.html
        • update.html
            • thank_you.html
        • celerybeat_schedule
        • debug.log
        • local_beat.sh
        • local_gunicorn.sh
        • local_run.sh
        • local_setup.sh
    • __logs__
    • 18_characters_selected

```

```

File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
    • iit_task_push
        • __init__.py
        • __env__
        • __pycache__
        • __application
            • __init__.py
            • __app__
            • __tasks__
            • __validation__
            • __utils__
        • __controllers__
        • __models__
        • __database__
        • __utils__
        • __tasks__
        • __validation__
        • __utils__
    • __dbs_directory
    • __static__
    • __bootstrap__
    • __img__
    • __js__
        • app.js
        • custom.js
        • show_alert.js
    • __css__
        • style.css
    • __templates__
        • 403.html
        • 404.html
        • article.html
        • articles_by_author.html
        • create_article.html
        • feedback.html
        • show_update.html
        • show_update_vue.html
        • update.html
            • thank_you.html
        • celerybeat_schedule
        • debug.log
        • local_beat.sh
        • local_gunicorn.sh
        • local_run.sh
        • local_setup.sh
    • __logs__
    • 18_characters_selected

```

Screenshot 2 (Middle):

```

File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
    • iit_task_push
        • __init__.py
        • __env__
        • __pycache__
        • __application
            • __init__.py
            • __app__
            • __tasks__
            • __validation__
            • __utils__
        • __controllers__
        • __models__
        • __database__
        • __utils__
        • __tasks__
        • __validation__
        • __utils__
    • __dbs_directory
    • __static__
    • __bootstrap__
    • __img__
    • __js__
        • app.js
        • custom.js
        • show_alert.js
    • __css__
        • style.css
    • __templates__
        • 403.html
        • 404.html
        • article.html
        • articles_by_author.html
        • create_article.html
        • feedback.html
        • show_update.html
        • show_update_vue.html
        • update.html
            • thank_you.html
        • celerybeat_schedule
        • debug.log
        • local_beat.sh
        • local_gunicorn.sh
        • local_run.sh
        • local_setup.sh
    • __logs__
    • 18_characters_selected

```

```

19
20
21 @celery.task()
22 def hello_world(name):
23     print("hello {}".format(name))
24
25
26
27 @celery.task()
28 def long_running_job():
29     print("STARTED LONG JOB")
30     now = datetime.now()
31     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")

```

Screenshot 3 (Bottom):

```

File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
    • iit_task_push
        • __init__.py
        • __env__
        • __pycache__
        • __application
            • __init__.py
            • __app__
            • __tasks__
            • __validation__
            • __utils__
        • __controllers__
        • __models__
        • __database__
        • __utils__
        • __tasks__
        • __validation__
        • __utils__
    • __dbs_directory
    • __static__
    • __bootstrap__
    • __img__
    • __js__
        • app.js
        • custom.js
        • show_alert.js
    • __css__
        • style.css
    • __templates__
        • 403.html
        • 404.html
        • article.html
        • articles_by_author.html
        • create_article.html
        • feedback.html
        • show_update.html
        • show_update_vue.html
        • update.html
            • thank_you.html
        • celerybeat_schedule
        • debug.log
        • local_beat.sh
        • local_gunicorn.sh
        • local_run.sh
        • local_setup.sh
    • __logs__
    • 18_characters_selected

```

```

19
20
21 @celery.task()
22 def hello_world(name):
23     print("hello {}".format(name))
24
25
26
27 @celery.task()
28 def long_running_job():
29     print("STARTED LONG JOB")
30     now = datetime.now()
31     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
32     sse.publish({"message": "STARTED EMAILING JOB =" + dt_string}, type='greet')
33     for lp in range(100):
34         now = datetime.now()
35         dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
36         sse.publish({"message": "SENDING EMAIL TO =" + str(lp)}, type='greet')
37         sse.publish({"message": "SENDING EMAIL AT =" + dt_string}, type='greet')
38         #send an email
39         time.sleep(2)
40     now = datetime.now()
41     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
42     sse.publish({"message": "COMPLETED EMAILING JOB =" + dt_string}, type='greet')
43     print("COMPLETE LONG RUN")
44
45
46
47

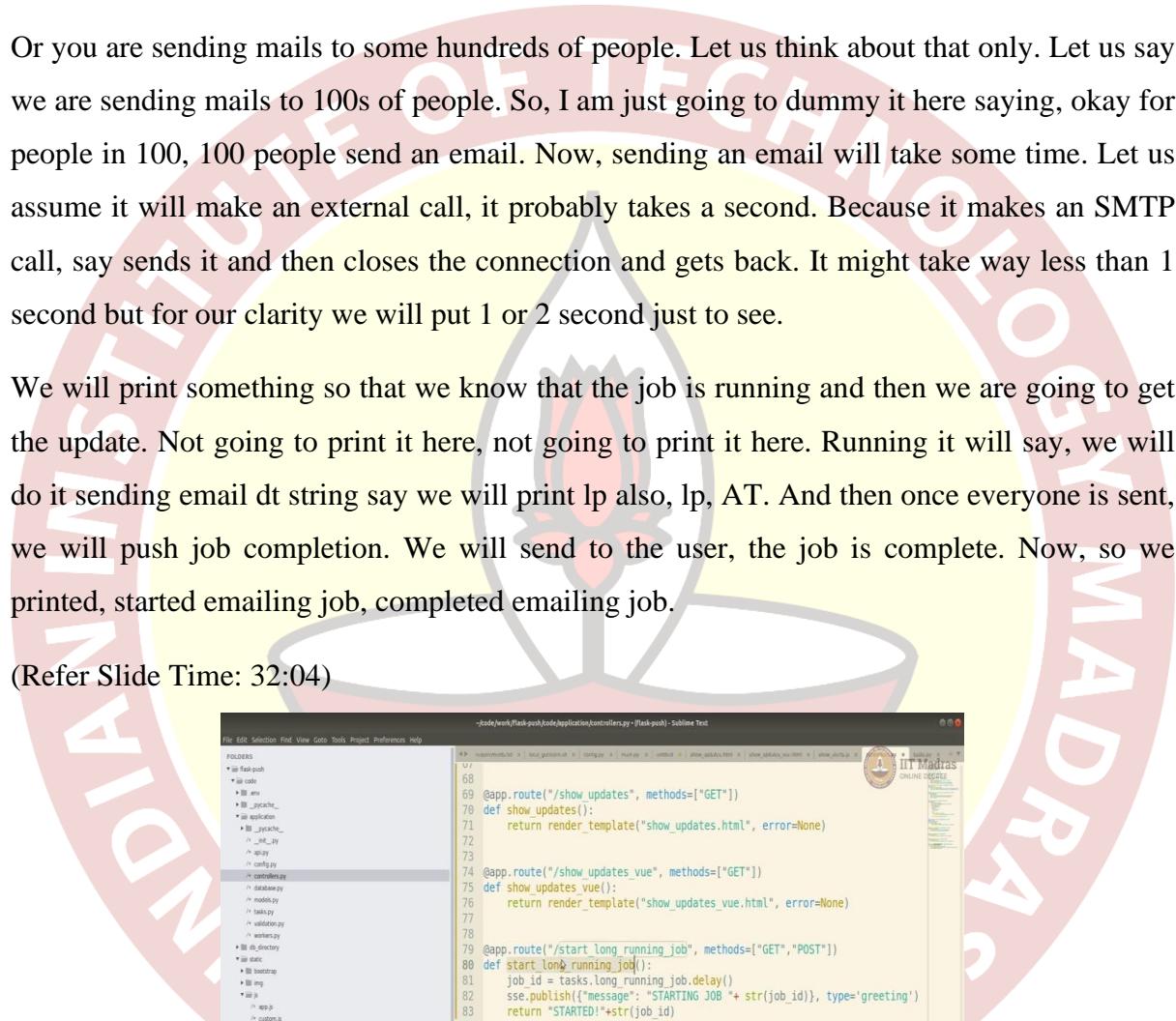
```

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)



```
-jcode\work\flask-push\code\application\controllers.py - [Task-push] - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
• iit_taskpush
• iit_code
• iit_arts
• iit_science
• iit_application
• iit_psychology
• iit_it
• iit_gop
• iit_config
• iit_controller
• iit_models
• iit_tasks
• iit_validation
• iit_workers
• iit_dbs_directory
• iit_static
• iit_bootstrap
• iit_img
• iit_js
• iit_apps
• iit_custom
• iit_show_update
• iit_vuet
• iit_style
• iit_templates
• iit_assets
• iit_index.html
• iit_article.html
• iit_article_by_author.html
• iit_create_article.html
• iit_feedback.html
• iit_show_update.html
• iit_show_update_vue.html
• iit_tasks.html
• iit_greet.html
• iit_cancellation_schedule
• iit_debug_log
• iit_local_beat.sh
• iit_local_gunicorn.sh
• iit_local_npm.sh
• iit_local_setups.sh
• iit_thermometer.html
68
69 @app.route("/show_updates", methods=["GET"])
70 def show_updates():
71     return render_template("show_updates.html", error=None)
72
73
74 @app.route("/show_updates_vue", methods=["GET"])
75 def show_updates_vue():
76     return render_template("show_updates_vue.html", error=None)
77
78
79 @app.route("/start_long_running_job", methods=["GET", "POST"])
80 def start_long_running_job():
81     job_id = tasks.long_running_job.delay()
82     sse.publish({"message": "STARTING JOB "+ str(job_id), type='greeting'})
83     return "STARTED!" +str(job_id)
```

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)

The image displays three screenshots of a Sublime Text editor window, showing code snippets for different Python files. The top two screenshots show the 'tasks.py' file, and the bottom one shows the 'controllers.py' file.

tasks.py (Top Screenshot):

```
19
20     @celery.task()
21     def hello_world(name):
22         print("hello {}".format(name))
23
24
25
26
27     @celery.task()
28     def long_running_job():
29         print("STARTED LONG JOB")
30         now = datetime.now()
31         dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
32         sse.publish({"message": "STARTED EMAILING JOB =" + dt_string}, type='greet')
33         for lp in range(100):
34             now = datetime.now()
35             dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
36             sse.publish({"message": "SENDING EMAIL TO =" + str(lp)}, type='greet')
37             sse.publish({"message": "SENDING EMAIL AT =" + dt_string}, type='greet')
38             #send an email
39             time.sleep(2)
40         now = datetime.now()
41         dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
42         sse.publish({"message": "COMPLETED EMAILING JOB =" + dt_string}, type='greet')
43
44
45
46
47
```

tasks.py (Middle Screenshot):

```
1 import time
2 from application.workers import celery
3 from datetime import datetime
4 from flask_sse import sse
5 from celery.schedules import crontab
6
7 # @celery.on_after_finalize.connect
8 # def setup_periodic_tasks(sender, **kwargs):
9 #     sender.add_periodic_task(10.0, print_current_time_job(), name='At every 10 seconds')
10
11
12 @celery.task()
13 def print_current_time_job():
14     print("START")
15     now = datetime.now()
16     print("now in task =", now)
17     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
18     print("date and time =", dt_string)
19     print("COMPLETE")
20     return dt_string
21
22
23 @celery.task()
24 def hello_world(name):
25     print("hello {}".format(name))
26
27
28
29 @celery.task()
30 def long_running_job():
31     print("STARTED LONG JOB")
32     now = datetime.now()
33
```

controllers.py (Bottom Screenshot):

```
1
2
3     @app.route("/test_send_message", methods=["GET", "POST"])
4     def test_send_message():
5         sse.publish({"message": "Hello!"}, type='greeting')
6         return "Message sent check the other browser!"
7
8
9     @app.route("/show_updates", methods=["GET"])
10    def show_updates():
11        return render_template("show_updates.html", error=None)
12
13
14    @app.route("/show_updates_vue", methods=["GET"])
15    def show_updates_vue():
16        return render_template("show_updates_vue.html", error=None)
17
18
19    @app.route("/start_long_running_job", methods=["GET"])
20    def start_long_running_job():
21        job_id = tasks.long_running_job.delay()
22        sse.publish({"message": "STARTING JOB " + str(job_id)}, type='greeting')
23        return "STARTED!"+str(job_id)
24
25
26
```

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)

```
the@juna:~/code/work/flask-push/code
```

```
File Edit View Search Terminal Tabs Help
the@juna:~/code/work/flask-push/code
the@juna:~/code/work/flask-push/code
the@juna:~/code/work/flask-push/code
```

```
pushed config
DB Init
DB Init complete
Create app complete
in controller app <Flask 'main'>

..... celery@juna 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:0x7f6c70e2c90
... > transport: redis://localhost:6379/1
... > results: redis://localhost:6379/2
... * ... > concurrency: 4 (prefork)
... *** ... > task events: OFF (enable -t 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 23:45:29,049: INFO/MainProcess] mangle: searching for neighbors
[2022-02-23 23:45:29,065: INFO/MainProcess] mangle: all alone
[2022-02-23 23:45:29,095: INFO/MainProcess] celery@juna ready.
[2022-02-23 23:45:29,095: INFO/MainProcess] Task application.tasks.long_running_job[8f33cdad-0698-4ef2-b4f8-cdce4e2f0bb1] received
[2022-02-23 23:47:46,557: WARNING/ForkPoolWorker-4] STARTED LONG JOB
[2022-02-23 23:47:46,557: WARNING/ForkPoolWorker-4] 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
: None
the@juna:~/code/work/flask-push/code
```



```
the@juna:~/code/work/flask-push/code
```

```
File Edit View Search Terminal Tabs Help
the@juna:~/code/work/flask-push/code
the@juna:~/code/work/flask-push/code
the@juna:~/code/work/flask-push/code
```

```
Welcome to the iPython 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
[2022-02-23 23:47:40 +0530] [23944] [INFO] Starting unicorn 20.1.0
[2022-02-23 23:47:40 +0530] [23944] [INFO] Listening at: http://127.0.0.1:8000 (23944)
[2022-02-23 23:47:40 +0530] [23944] [INFO] Using worker: gevent
[2022-02-23 23:47:40 +0530] [23947] [INFO] Booting worker with pid: 23947
[2022-02-23 23:47:40 +0530] [23948] [INFO] Booting worker with pid: 23948
[2022-02-23 23:47:40 +0530] [23949] [INFO] Booting worker with pid: 23949
[2022-02-23 23:47:40 +0530] [23950] [INFO] Booting worker with pid: 23950
stage
Starting stage
pushed config
DB Init
DB Init complete
Create app complete
in controller app <Flask 'main'>
stage
Starting stage
pushed config
DB Init
DB Init complete
Create app complete
in controller app <Flask 'main'>
stage
Starting stage
pushed config
DB Init
DB Init complete
Create app complete
in controller app <Flask 'main'>
stage
Starting stage
pushed config
DB Init
DB Init complete
Create app complete
in controller app <Flask 'main'>
stage
Starting stage
pushed config
DB Init
DB Init complete
Create app complete
in controller app <Flask 'main'>
```

```

thej@uma:~/code/work/flask-push/code$ thej@uma:~/code/work/flask-push/code
[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/cnv/lib/python3.7/site-packages/gevent/monkey.py", line 893, in _shutdown
    sleep()
  File "/home/thej/code/work/flask-push/code/cnv/lib/python3.7/site-packages/gevent/hub.py", line 159, in sleep
    waiter.get()
  File "/src/gevent/_event/_waiter.py", line 143, in gevent_c_watcher_Waiter_get
  File "/src/gevent/_event/_waiter.py", line 154, in gevent_c_watcher_Waiter_get
  File "/src/gevent/_event/_greenlet_primitives_px4", line 61, in gevent_c_greenlet_primitives_SwitchOutGreenletWithLoop_switch
  File "/src/gevent/_greenlet_primitives_py", line 61, in gevent_c_greenlet_primitives_SwitchOutGreenletWithLoop_switch
  File "/src/gevent/_greenlet_primitives_px4", line 65, in gevent_c_greenlet_primitives_SwitchOutGreenletWithLoop_switch
  File "/src/gevent/_greenlet_primitives_py", line 986, in gevent_c_greenlet_primitives_Greenlet_run
  File "/home/thej/code/work/flask-push/code/cnv/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/cnv/lib/python3.7/site-packages/gevent/monkey.py", line 893, in _shutdown
    sleep()
  File "/home/thej/code/work/flask-push/code/cnv/lib/python3.7/site-packages/gevent/hub.py", line 159, in sleep
    waiter.get()
  File "/src/gevent/_event/_waiter.py", line 143, in gevent_c_watcher_Waiter_get
  File "/src/gevent/_event/_waiter.py", line 154, in gevent_c_watcher_Waiter_get
  File "/src/gevent/_greenlet_primitives_px4", line 35, in gevent_c_greenlet_primitives_Greenlet_run
  File "/src/gevent/_greenlet_primitives_py", line 61, in gevent_c_greenlet_primitives_SwitchOutGreenletWithLoop_switch
  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$ sh local_gunicorn.sh
[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/cnv/lib/python3.7/site-packages/gevent/monkey.py", line 893, in _shutdown
    sleep()
  File "/home/thej/code/work/flask-push/code/cnv/lib/python3.7/site-packages/gevent/hub.py", line 159, in sleep
    waiter.get()
  File "/src/gevent/_event/_waiter.py", line 143, in gevent_c_watcher_Waiter_get
  File "/src/gevent/_event/_waiter.py", line 154, in gevent_c_watcher_Waiter_get
  File "/src/gevent/_event/_greenlet_primitives_px4", line 61, in gevent_c_greenlet_primitives_SwitchOutGreenletWithLoop.switch
  File "/src/gevent/_greenlet_primitives_py", line 61, in gevent_c_greenlet_primitives_SwitchOutGreenletWithLoop.switch
  File "/src/gevent/_greenlet_primitives_px4", line 35, in gevent_c_greenlet_primitives_Greenlet_run
  File "/src/gevent/_greenlet_primitives_py", line 986, in gevent_c_greenlet_primitives_Greenlet_run
  File "/home/thej/code/work/flask-push/code/cnv/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_workers.sh
[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-cde4e2f0bb1] received
[2022-02-23 23:47:46.757: WARNING/ForkPoolWorker-4] STARTED LONG JOB
[2022-02-23 23:47:46.757: WARNING/ForkPoolWorker-4] COMPLETE LONG RUN
[2022-02-23 23:51:07.897: INFO/ForkPoolWorker-4] Task application.tasks.long_running_job[8f33cdad-0698-4ef2-b4f8-cde4e2f0bb1] succeeded in 200.94103766104672
thej@uma:~/code/work/flask-push/code$ sh local_workers.sh

```

```

the@uma:~/code/work/flask-push/code$ sh local_workers.sh
=====
Welcome 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
Starting Local Development
pushed config
do Init
do Init complete
Create app complete
in controller app <Flask 'main'>

..... celery@uma v5.2.3 (down-chorus)
.... ****
... Linux-5.4.0-99-generic-x86_64-with-Ubuntu-18.04-bionic 2022-02-23 23:51:52
... * ...
... [config]
...     .> app: Application Jobs:0x7ffefad94bc50
...     .> 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

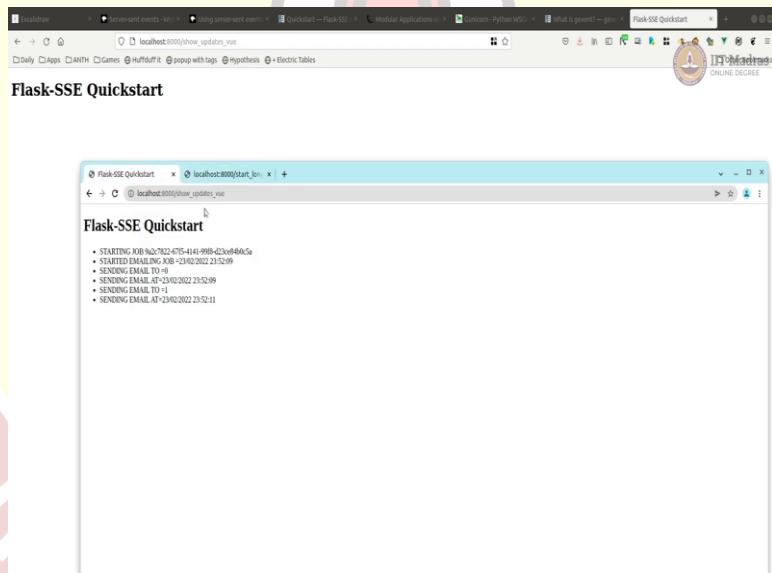
[tasks]
.. application.tasks.hello_world
.. application.tasks.long_running_job
.. application.tasks.print_current_time_job

[2022-02-23 23:51:53,264: INFO/MainProcess] Connected to redis://localhost:6379/1
[2022-02-23 23:51:53,272: INFO/MainProcess] mingle: searching for neighbors
[2022-02-23 23:51:54,389: INFO/MainProcess] mingle: all alone
[2022-02-23 23:51:54,332: INFO/MainProcess] celery@uma ready.

```

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)



The screenshot shows a computer desktop with two windows open. The top window is a web browser displaying a log of email sending events from a Flask application. The bottom window is a code editor showing Python code for a Celery task named 'long_running_job'. The code includes logic to publish messages to a socket.io connection every 2 seconds while performing a long-running task.

```

Flask-SSE Quickstart
• SENDING EMAIL TO=1
• SENDING EMAIL AT=23/02/2022 23:52:17
• SENDING EMAIL TO=5
• SENDING EMAIL AT=23/02/2022 23:52:19
• SENDING EMAIL TO=6
• SENDING EMAIL AT=23/02/2022 23:52:21
• SENDING EMAIL TO=7
• SENDING EMAIL AT=23/02/2022 23:52:23
• SENDING EMAIL TO=8
• SENDING EMAIL AT=23/02/2022 23:52:25
• SENDING EMAIL TO=9
• SENDING EMAIL AT=23/02/2022 23:52:27
• SENDING EMAIL TO=10
• SENDING EMAIL AT=23/02/2022 23:52:29
• SENDING EMAIL TO=11
• SENDING EMAIL AT=23/02/2022 23:52:31
• SENDING EMAIL TO=12
• SENDING EMAIL AT=23/02/2022 23:52:33
• SENDING EMAIL TO=13
• SENDING EMAIL AT=23/02/2022 23:52:35

STARTING JOB NO=25-414-998-023x860c5a
• STARTED EMAILING JOB AT=23/02/2022 23:52:39
• SENDING EMAIL TO=4
• SENDING EMAIL AT=23/02/2022 23:52:39
• SENDING EMAIL TO=14
• SENDING EMAIL AT=23/02/2022 23:52:41
• SENDING EMAIL TO=15
• SENDING EMAIL AT=23/02/2022 23:52:43
• SENDING EMAIL TO=16
• SENDING EMAIL AT=23/02/2022 23:52:45
• SENDING EMAIL TO=17
• SENDING EMAIL AT=23/02/2022 23:52:47
• SENDING EMAIL TO=18
• SENDING EMAIL AT=23/02/2022 23:52:49
• SENDING EMAIL TO=19
• SENDING EMAIL AT=23/02/2022 23:52:51
• SENDING EMAIL TO=20
• SENDING EMAIL AT=23/02/2022 23:52:53
• SENDING EMAIL TO=21
• SENDING EMAIL AT=23/02/2022 23:52:55
• SENDING EMAIL TO=22
• SENDING EMAIL AT=23/02/2022 23:52:57
• SENDING EMAIL TO=23
• SENDING EMAIL AT=23/02/2022 23:52:59
• SENDING EMAIL TO=24
• SENDING EMAIL AT=23/02/2022 23:53:01
• SENDING EMAIL TO=25
• SENDING EMAIL AT=23/02/2022 23:53:03
• SENDING EMAIL TO=26
• SENDING EMAIL AT=23/02/2022 23:53:05
• SENDING EMAIL TO=27
• SENDING EMAIL AT=23/02/2022 23:53:07
• SENDING EMAIL TO=28
• SENDING EMAIL AT=23/02/2022 23:53:09
• SENDING EMAIL TO=29
• SENDING EMAIL AT=23/02/2022 23:53:11
• SENDING EMAIL TO=30
• SENDING EMAIL AT=23/02/2022 23:53:13
• SENDING EMAIL TO=31
• SENDING EMAIL AT=23/02/2022 23:53:15
• SENDING EMAIL TO=32
• SENDING EMAIL AT=23/02/2022 23:53:17
• SENDING EMAIL TO=33
• SENDING EMAIL AT=23/02/2022 23:53:19
• SENDING EMAIL TO=34
• SENDING EMAIL AT=23/02/2022 23:53:21
• SENDING EMAIL TO=35
• SENDING EMAIL AT=23/02/2022 23:53:23
• SENDING EMAIL TO=36
• SENDING EMAIL AT=23/02/2022 23:53:25
• SENDING EMAIL TO=37
• SENDING EMAIL AT=23/02/2022 23:53:27
• SENDING EMAIL TO=38
• SENDING EMAIL AT=23/02/2022 23:53:29
• SENDING EMAIL TO=39
• SENDING EMAIL AT=23/02/2022 23:53:31
• SENDING EMAIL TO=40
• SENDING EMAIL AT=23/02/2022 23:53:33
• SENDING EMAIL TO=41
• SENDING EMAIL AT=23/02/2022 23:53:35

File Edit Selection Find Go Tools Project Preferences Help
File Edit Selection Find Go Tools Project Preferences Help
19     print("COMPLETE")
20     return dt_string
21
22
23 @celery.task()
24 def hello_world(name):
25     print("hello {}".format(name))
26
27
28 @celery.task()
29 def long_running_job():
30     print("STARTED LONG JOB")
31     now = datetime.now()
32     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
33     sse.publish({"message": "STARTED EMAILING JOB =" + dt_string}, type='greet')
34     for lp in range(100):
35         now = datetime.now()
36         dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
37         sse.publish({"message": "SENDING EMAIL TO =" + str(lp)}, type='greet')
38         sse.publish({"message": "SENDING EMAIL AT=" + dt_string}, type='greet')
39         time.sleep(2)
40         send an email
41         time.sleep(2)
42     now = datetime.now()
43     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
44     sse.publish({"message": "COMPLETED EMAILING JOB =" + dt_string}, type='greet')
45     print("COMPLETE LONG RUN")
46
47
48
49

```

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)

The image consists of three vertically stacked screenshots of a Sublime Text editor window, showing code snippets for a Python application named 'task-push'. The top screenshot shows the main application logic, while the middle and bottom screenshots show the output of a 'Flask-SSE Quickstart' command.

Top Screenshot (Code View):

```

19: int("COMPLETE")
20: return dt_string
21:
22:
23:     y.task()
24:     hello_world(name):
25:         int("hello {}".format(name))
26:
27:
28:
29:     y.task()
30:     msg_running_job():
31:         int("STARTED LONG JOB")
32:         now = datetime.now()
33:         _string = now.strftime("%d/%m/%Y %H:%M:%S")
34:         ie.publish({"message": "STARTED EMAILING JOB =" + dt_string}, type='greeting')
35:         lp in range(100):
36:             now = datetime.now()
37:             dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
38:             sse.publish({"message": "SENDING EMAIL TO =" + str(lp)}, type='greeting')
39:             sse.publish({"message": "SENDING EMAIL AT =" + dt_string}, type='greeting')
40:             #send an email
41:             time.sleep(2)
42:         now = datetime.now()
43:         _string = now.strftime("%d/%m/%Y %H:%M:%S")
44:         ie.publish({"message": "COMPLETED EMAILING JOB =" + dt_string}, type='greeting'
45:         int("COMPLETE LONG RUN")
46:
47:
48:
49:

```

Middle Screenshot (Output View):

```

19: int("COMPLETE")
20: STARTED JOB ID:7024744+4995-d23x490c5a
21: STARTED EMAIL TO -0
22: SENDING EMAIL TO -0
23: SENDING EMAIL TO -1
24: SENDING EMAIL TO -2
25: SENDING EMAIL TO -3
26: SENDING EMAIL TO -4
27: SENDING EMAIL TO -5
28: SENDING EMAIL TO -6
29: SENDING EMAIL TO -7
30: SENDING EMAIL TO -8
31: SENDING EMAIL TO -9
32: SENDING EMAIL TO -10
33: SENDING EMAIL TO -11
34: SENDING EMAIL TO -12
35: SENDING EMAIL TO -13
36: SENDING EMAIL TO -14
37: SENDING EMAIL TO -15
38: SENDING EMAIL TO -16
39: SENDING EMAIL TO -17
40: SENDING EMAIL TO -18
41: SENDING EMAIL TO -19
42: SENDING EMAIL TO -20
43: SENDING EMAIL TO -21
44: SENDING EMAIL TO -22
45: SENDING EMAIL TO -23
46: SENDING EMAIL TO -24
47: SENDING EMAIL TO -25
48: SENDING EMAIL TO -26
49: SENDING EMAIL TO -27
50: SENDING EMAIL TO -28
51: SENDING EMAIL TO -29
52: SENDING EMAIL TO -30
53: SENDING EMAIL TO -31
54: SENDING EMAIL TO -32
55: SENDING EMAIL TO -33
56: SENDING EMAIL TO -34
57: SENDING EMAIL TO -35
58: SENDING EMAIL TO -36
59: SENDING EMAIL TO -37
60: SENDING EMAIL TO -38
61: SENDING EMAIL TO -39
62: SENDING EMAIL TO -40
63: SENDING EMAIL TO -41
64: SENDING EMAIL TO -42
65: SENDING EMAIL TO -43
66: SENDING EMAIL TO -44
67: SENDING EMAIL TO -45
68: SENDING EMAIL TO -46
69: SENDING EMAIL TO -47
70: SENDING EMAIL TO -48
71: SENDING EMAIL TO -49
72: SENDING EMAIL TO -50
73: SENDING EMAIL TO -51
74: SENDING EMAIL TO -52
75: SENDING EMAIL TO -53
76: SENDING EMAIL TO -54
77: SENDING EMAIL TO -55
78: SENDING EMAIL TO -56
79: SENDING EMAIL TO -57
80: SENDING EMAIL TO -58
81: SENDING EMAIL TO -59
82: SENDING EMAIL TO -60
83: SENDING EMAIL TO -61
84: SENDING EMAIL TO -62
85: SENDING EMAIL TO -63
86: SENDING EMAIL TO -64
87: SENDING EMAIL TO -65
88: SENDING EMAIL TO -66
89: SENDING EMAIL TO -67
90: SENDING EMAIL TO -68
91: SENDING EMAIL TO -69
92: SENDING EMAIL TO -70
93: SENDING EMAIL TO -71
94: SENDING EMAIL TO -72
95: SENDING EMAIL TO -73
96: SENDING EMAIL TO -74
97: SENDING EMAIL TO -75
98: SENDING EMAIL TO -76
99: SENDING EMAIL TO -77
100: SENDING EMAIL TO -78
101: SENDING EMAIL TO -79
102: SENDING EMAIL TO -80
103: SENDING EMAIL TO -81
104: SENDING EMAIL TO -82
105: SENDING EMAIL TO -83
106: SENDING EMAIL TO -84
107: SENDING EMAIL TO -85
108: SENDING EMAIL TO -86
109: SENDING EMAIL TO -87
110: SENDING EMAIL TO -88
111: SENDING EMAIL TO -89
112: SENDING EMAIL TO -90
113: SENDING EMAIL TO -91
114: SENDING EMAIL TO -92
115: SENDING EMAIL TO -93
116: SENDING EMAIL TO -94
117: SENDING EMAIL TO -95
118: SENDING EMAIL TO -96
119: SENDING EMAIL TO -97
120: SENDING EMAIL TO -98
121: SENDING EMAIL TO -99
122: SENDING EMAIL TO -100
123: COMPLETED

```

Bottom Screenshot (Output View):

```

19: int("COMPLETE")
20: STARTED JOB ID:7024744+4995-d23x490c5a
21: STARTED EMAILING JOB @ 23/02/2022 23:52:09
22: SENDING EMAIL TO -0
23: SENDING EMAIL TO -1
24: SENDING EMAIL TO -2
25: SENDING EMAIL TO -3
26: SENDING EMAIL TO -4
27: SENDING EMAIL TO -5
28: SENDING EMAIL TO -6
29: SENDING EMAIL TO -7
30: SENDING EMAIL TO -8
31: SENDING EMAIL TO -9
32: SENDING EMAIL TO -10
33: SENDING EMAIL TO -11
34: SENDING EMAIL TO -12
35: SENDING EMAIL TO -13
36: SENDING EMAIL TO -14
37: SENDING EMAIL TO -15
38: SENDING EMAIL TO -16
39: SENDING EMAIL TO -17
40: SENDING EMAIL TO -18
41: SENDING EMAIL TO -19
42: SENDING EMAIL TO -20
43: SENDING EMAIL TO -21
44: SENDING EMAIL TO -22
45: SENDING EMAIL TO -23
46: SENDING EMAIL TO -24
47: SENDING EMAIL TO -25
48: SENDING EMAIL TO -26
49: SENDING EMAIL TO -27
50: SENDING EMAIL TO -28
51: SENDING EMAIL TO -29
52: SENDING EMAIL TO -30
53: SENDING EMAIL TO -31
54: SENDING EMAIL TO -32
55: SENDING EMAIL TO -33
56: SENDING EMAIL TO -34
57: SENDING EMAIL TO -35
58: SENDING EMAIL TO -36
59: SENDING EMAIL TO -37
60: SENDING EMAIL TO -38
61: SENDING EMAIL TO -39
62: SENDING EMAIL TO -40
63: SENDING EMAIL TO -41
64: SENDING EMAIL TO -42
65: SENDING EMAIL TO -43
66: SENDING EMAIL TO -44
67: SENDING EMAIL TO -45
68: SENDING EMAIL TO -46
69: SENDING EMAIL TO -47
70: SENDING EMAIL TO -48
71: SENDING EMAIL TO -49
72: SENDING EMAIL TO -50
73: SENDING EMAIL TO -51
74: SENDING EMAIL TO -52
75: SENDING EMAIL TO -53
76: SENDING EMAIL TO -54
77: SENDING EMAIL TO -55
78: SENDING EMAIL TO -56
79: SENDING EMAIL TO -57
80: SENDING EMAIL TO -58
81: SENDING EMAIL TO -59
82: SENDING EMAIL TO -60
83: SENDING EMAIL TO -61
84: SENDING EMAIL TO -62
85: SENDING EMAIL TO -63
86: SENDING EMAIL TO -64
87: SENDING EMAIL TO -65
88: SENDING EMAIL TO -66
89: SENDING EMAIL TO -67
90: SENDING EMAIL TO -68
91: SENDING EMAIL TO -69
92: SENDING EMAIL TO -70
93: SENDING EMAIL TO -71
94: SENDING EMAIL TO -72
95: SENDING EMAIL TO -73
96: SENDING EMAIL TO -74
97: SENDING EMAIL TO -75
98: SENDING EMAIL TO -76
99: SENDING EMAIL TO -77
100: SENDING EMAIL TO -78
101: SENDING EMAIL TO -79
102: SENDING EMAIL TO -80
103: SENDING EMAIL TO -81
104: SENDING EMAIL TO -82
105: SENDING EMAIL TO -83
106: SENDING EMAIL TO -84
107: SENDING EMAIL TO -85
108: SENDING EMAIL TO -86
109: SENDING EMAIL TO -87
110: SENDING EMAIL TO -88
111: SENDING EMAIL TO -89
112: SENDING EMAIL TO -90
113: SENDING EMAIL TO -91
114: SENDING EMAIL TO -92
115: SENDING EMAIL TO -93
116: SENDING EMAIL TO -94
117: SENDING EMAIL TO -95
118: SENDING EMAIL TO -96
119: SENDING EMAIL TO -97
120: SENDING EMAIL TO -98
121: SENDING EMAIL TO -99
122: SENDING EMAIL TO -100
123: COMPLETED

```

Flask-SSE Quickstart

```

• SENDING EMAIL TO=1
• SENDING EMAIL AT=23/02/2022 23:52:17
• SENDING EMAIL TO=5
• SENDING EMAIL AT=23/02/2022 23:52:19
• SENDING EMAIL TO=6
• SENDING EMAIL AT=23/02/2022 23:52:21
• SENDING EMAIL TO=7
• SENDING EMAIL AT=23/02/2022 23:52:23
• SENDING EMAIL AT=23/02/2022 23:52:25
• SENDING EMAIL TO=9
• SENDING EMAIL AT=23/02/2022 23:52:27
• SENDING EMAIL AT=23/02/2022 23:52:29
• SENDING EMAIL TO=11
• SENDING EMAIL AT=23/02/2022 23:52:31
• SENDING EMAIL AT=23/02/2022 23:52:33
• SENDING EMAIL TO=13
• SENDING EMAIL AT=23/02/2022 23:52:35
• SENDING EMAIL TO=14
• SENDING EMAIL AT=23/02/2022 23:52:37
• SENDING EMAIL TO=15
• SENDING EMAIL AT=23/02/2022 23:52:39
• SENDING EMAIL TO=16
• SENDING EMAIL AT=23/02/2022 23:52:41
• SENDING EMAIL AT=23/02/2022 23:52:44
• SENDING EMAIL TO=18
• SENDING EMAIL AT=23/02/2022 23:52:46
• SENDING EMAIL TO=19
• SENDING EMAIL AT=23/02/2022 23:52:48
• SENDING EMAIL TO=20
• SENDING EMAIL AT=23/02/2022 23:52:50
• SENDING EMAIL TO=21
• SENDING EMAIL AT=23/02/2022 23:52:52
• SENDING EMAIL TO=22
• SENDING EMAIL AT=23/02/2022 23:52:54
• SENDING EMAIL TO=23
• SENDING EMAIL AT=23/02/2022 23:52:56
• SENDING EMAIL TO=24
• SENDING EMAIL AT=23/02/2022 23:52:58
• SENDING EMAIL TO=25
• SENDING EMAIL AT=23/02/2022 23:53:00
• SENDING EMAIL TO=26
• SENDING EMAIL AT=23/02/2022 23:53:02
• SENDING EMAIL AT=23/02/2022 23:53:04

```

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)

File Edit Selection Find View Goto Tools Project Preferences Help

Tasks.py

```

19     print("COMPLETE")
20     return dt_string
21
22
23 @celery.task()
24 def hello_world(name):
25     print("hello {}".format(name))
26
27
28
29 @celery.task()
30 def long_running_job():
31     print("STARTED LONG JOB")
32     now = datetime.now()
33     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
34     sse.publish({"message": "STARTED EMAILING JOB =" + dt_string }, type='greet'
35     for lp in range(100):
36         now = datetime.now()
37         dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
38         sse.publish({"message": "SENDING EMAIL TO =" + str(lp) }, type='greet'
39         sse.publish({"message": "SENDING EMAIL AT =" + dt_string }, type='greet'
40         #send an email
41         time.sleep(2)
42         now = datetime.now()
43         dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
44         sse.publish({"message": "COMPLETED EMAILING JOB =" + dt_string }, type='greet'
45         print("COMPLETE LONG RUN")
46
47
48
49

```



File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS

- iit flask push
- iit code
- iit env
- iit_celache_
- iit application
- iit_celache_/_init_.py
- iit config.py
- iit controllers.py
- iit database.py
- iit models.py
- iit tasks.py
- iit validation.py
- iit workers.py
- iit __init__.py
- iit db_directory
- iit static
- iit img
- iit js
- iit app.js
- iit custom.js
- iit show_alert.js
- iit css
- iit templates
- iit 403.html
- iit 404.html
- iit article.html
- iit article_by_author.html
- iit create_article.html
- iit feedback.html
- iit show_update.html
- iit show_update_vue.html
- iit update.html
- iit thank_you.html
- iit_celache_schedule
- iit_debug_log
- iit local_init.sh
- iit local_gunicorn.sh
- iit local_run.sh
- iit log_setup.sh

```
1 import time
2 from application.workers import celery
3 from datetime import datetime
4 from flask_sse import sse
5 from celery.schedules import crontab
6
7 # celery.on_after_finalize.connect
8 # def setup_periodic_tasks(sender, **kwargs):
9 #     sender.add_periodic_task(10.0, print_current_time_job.s(), name='At-every-10-seconds')
10
11
12 @celery.task()
13 def print_current_time_job():
14     print("START")
15     now = datetime.now()
16     print("now in task =", now)
17     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
18     print("date and time =", dt_string)
19     print("COMPLETE")
20     return dt_string
21
22
23 @celery.task()
24 def hello_world(name):
25     print("hello {}".format(name))
26
27
28 @celery.task()
29 def long_running_job():
30     print("STARTED LONG JOB")
31     now = datetime.now()
32     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
33     sse.publish({"message": "STARTED EMAILING JOB =" + dt_string}, type='greet')
34     for lp in range(100):
35         now = datetime.now()
36         dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
37         sse.publish(lp, type='greet')
38     sse.publish("message", "SENDING EMAIL TO ="+str(lp), type='greet')
39     sse.publish("message", "SENDING EMAIL AT=" + dt_string, type='greet')
40     #send an email
41     time.sleep(2)
42     now = datetime.now()
43     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
44     sse.publish("message", "COMPLETED EMAILING JOB =" + dt_string, type='greet')
45     print("COMPLETE LONG RUN")
```

Source 4 Python

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS

- iit flask push
- iit code
- iit env
- iit_celache_
- iit application
- iit_celache_/_init_.py
- iit config.py
- iit controllers.py
- iit database.py
- iit models.py
- iit tasks.py
- iit validation.py
- iit workers.py
- iit __init__.py
- iit db_directory
- iit static
- iit img
- iit js
- iit app.js
- iit custom.js
- iit show_alert.js
- iit css
- iit templates
- iit 403.html
- iit 404.html
- iit article.html
- iit article_by_author.html
- iit create_article.html
- iit feedback.html
- iit show_update.html
- iit show_update_vue.html
- iit update.html
- iit thank_you.html
- iit_celache_schedule
- iit_debug_log
- iit local_init.sh
- iit local_gunicorn.sh
- iit local_run.sh
- iit log_setup.sh

```
7 # celery.on_after_finalize.connect
8 # def setup_periodic_tasks(sender, **kwargs):
9 #     sender.add_periodic_task(10.0, print_current_time_job.s(), name='At-every-10-seconds')
10
11
12 @celery.task()
13 def print_current_time_job():
14     print("START")
15     now = datetime.now()
16     print("now in task =", now)
17     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
18     print("date and time =", dt_string)
19     print("COMPLETE")
20     return dt_string
21
22
23 @celery.task()
24 def hello_world(name):
25     print("hello {}".format(name))
26
27
28 @celery.task()
29 def long_running_job():
30     print("STARTED LONG JOB")
31     now = datetime.now()
32     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
33     sse.publish({"message": "STARTED EMAILING JOB =" + dt_string}, type='greet')
34     for lp in range(100):
35         now = datetime.now()
36         dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
37         sse.publish(lp, type='greet')
38     sse.publish("message", "SENDING EMAIL TO ="+str(lp), type='greet')
39     sse.publish("message", "SENDING EMAIL AT=" + dt_string, type='greet')
40     #send an email
41     time.sleep(2)
42     now = datetime.now()
43     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
44     sse.publish("message", "COMPLETED EMAILING JOB =" + dt_string, type='greet')
45     print("COMPLETE LONG RUN")
```

Source 4 Python

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS

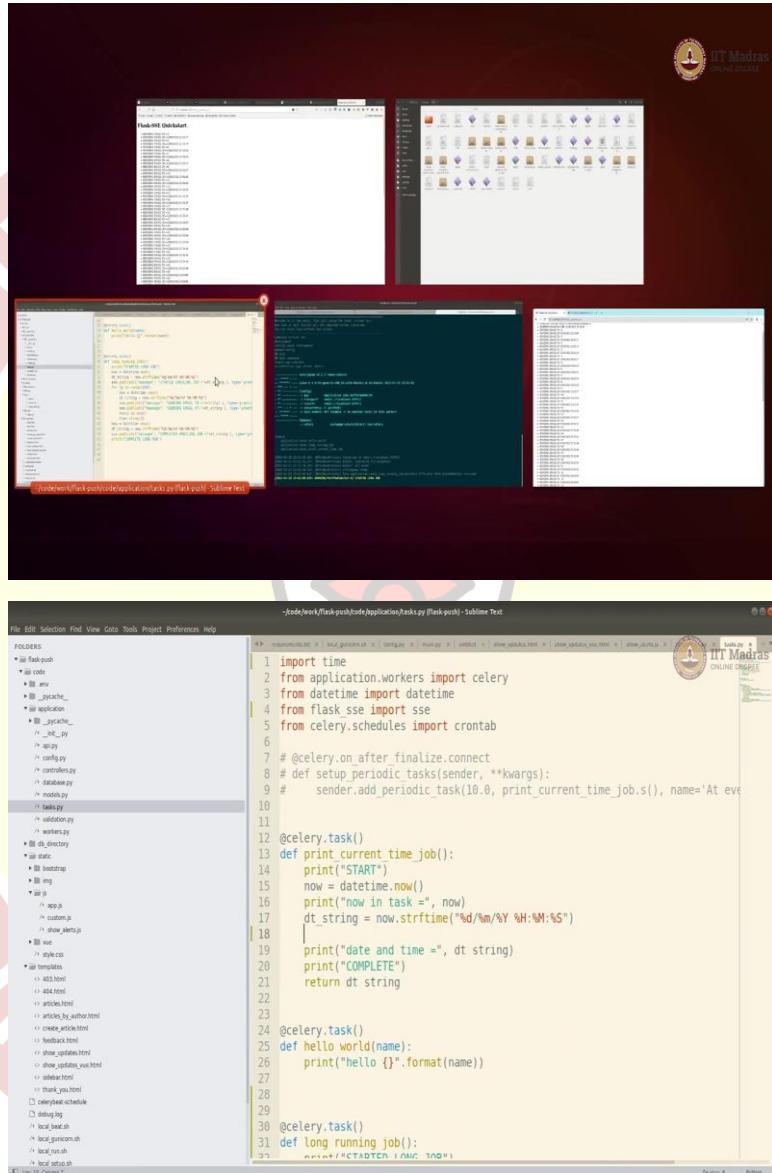
- iit flask push
- iit code
- iit env
- iit_celache_
- iit application
- iit_celache_/_init_.py
- iit config.py
- iit controllers.py
- iit database.py
- iit models.py
- iit tasks.py
- iit validation.py
- iit workers.py
- iit __init__.py
- iit db_directory
- iit static
- iit img
- iit js
- iit app.js
- iit custom.js
- iit show_alert.js
- iit css
- iit templates
- iit 403.html
- iit 404.html
- iit article.html
- iit article_by_author.html
- iit create_article.html
- iit feedback.html
- iit show_update.html
- iit show_update_vue.html
- iit update.html
- iit thank_you.html
- iit_celache_schedule
- iit_debug_log
- iit local_init.sh
- iit local_gunicorn.sh
- iit local_run.sh
- iit log_setup.sh

```
22
23 @celery.task()
24 def hello_world(name):
25     print("hello {}".format(name))
26
27
28 @celery.task()
29 def long_running_job():
30     print("STARTED LONG JOB")
31     now = datetime.now()
32     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
33     sse.publish({"message": "STARTED EMAILING JOB =" + dt_string}, type='greet')
34     for lp in range(100):
35         now = datetime.now()
36         dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
37         sse.publish(lp, type='greet')
38     sse.publish("message", "SENDING EMAIL TO ="+str(lp), type='greet')
39     sse.publish("message", "SENDING EMAIL AT=" + dt_string, type='greet')
40     #send an email
41     time.sleep(2)
42     now = datetime.now()
43     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
44     sse.publish("message", "COMPLETED EMAILING JOB =" + dt_string, type='greet')
45     print("COMPLETE LONG RUN")
```

Source 4 Python

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

```

1 import time
2 from application.workers import celery
3 from datetime import datetime
4 from flask_sse import sse
5 from celery.schedules import crontab
6
7 # @celery.on_after_finalize.connect
8 # def setup_periodic_tasks(sender, **kwargs):
9 #     sender.add_periodic_task(10.0, print_current_time_job.s(), name='At eve
10
11
12 @celery.task()
13 def print_current_time_job():
14     print("START")
15     now = datetime.now()
16     print("now in task =", now)
17     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
18     sse.publish({"message": "STARTED EMAILING JOB "+dt_string}, type='greet
19     print("date and time =", dt_string)
20     print("COMPLETE")
21     return dt_string
22
23
24 @celery.task()
25 def hello_world(name):
26     print("hello {}".format(name))
27
28
29
30 @celery.task()
31 def long_running_job():
32     print("STARTED LONG JOB")

```

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS

- flask-push
- file code
- file env
- file __init__.py
- file application
- file __init__.py
- file config.py
- file controllers.py
- file database.py
- file models.py
- file tasks.py
- file validation.py
- file workers.py
- file __init__.py
- file static
- file bootstrap
- file img
- file app.js
- file custom.js
- file show_alerts.js
- file __init__.py
- file style.css
- file templates
- file 403.html
- file 404.html
- file article.html
- file article_by_author.html
- file create_article.html
- file feedback.html
- file show_updates.html
- file show_updates_vue.html
- file update.html
- file thank-you.html
- file celerybeat_schedule
- file debug.log
- file local_beat.sh
- file local_gunicorn.sh
- file local_run.sh
- file local_setup.sh

20 characters selected

-code/work/flask-push/code/application/tasks.py (Flask-push) - Sublime Text

```

1 import time
2 from application.workers import celery
3 from datetime import datetime
4 from flask_sse import sse
5 from celery.schedules import crontab
6
7 # @celery.on_after_finalize.connect
8 # def setup_periodic_tasks(sender, **kwargs):
9 #     sender.add_periodic_task(10.0, print_current_time_job.s(), name='At eve
10
11
12 @celery.task()
13 def print_current_time_job():
14     print("START")
15     now = datetime.now()
16     print("now in task =", now)
17     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
18     sse.publish({"message": "Current Time ="+dt_string}, type='greet
19     print("date and time =", dt_string)
20     print("COMPLETE")
21     return dt_string
22
23
24 @celery.task()
25 def hello_world(name):
26     print("hello {}".format(name))
27
28
29
30 @celery.task()
31 def long_running_job():
32     print("STARTED LONG JOB")

```

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS

- flask-push
- file code
- file env
- file __init__.py
- file application
- file __init__.py
- file config.py
- file controllers.py
- file database.py
- file models.py
- file tasks.py
- file validation.py
- file workers.py
- file __init__.py
- file static
- file bootstrap
- file img
- file app.js
- file custom.js
- file show_alerts.js
- file __init__.py
- file style.css
- file templates
- file 403.html
- file 404.html
- file article.html
- file article_by_author.html
- file create_article.html
- file feedback.html
- file show_updates.html
- file show_updates_vue.html
- file update.html
- file thank-you.html
- file celerybeat_schedule
- file debug.log
- file local_beat.sh
- file local_gunicorn.sh
- file local_run.sh
- file local_setup.sh

20 characters selected. Source: /code/work/flask-push/code/application/tasks.py (Flask-push)

-code/work/flask-push/code/application/tasks.py (Flask-push) - Sublime Text

Flask-SSE Quickstart

localhost:8000/start_log | +

```

+ C @ localhost:8000/start_log | +
  SENDING EMAIL ID=0
  SENDING EMAIL ID=23/02/2022 23:53:54
  SENDING EMAIL ID=23/02/2022 23:53:56
  SENDING EMAIL ID=23/02/2022 23:53:58
  SENDING EMAIL ID=23/02/2022 23:54:00
  SENDING EMAIL ID=23/02/2022 23:54:02
  SENDING EMAIL ID=23/02/2022 23:54:04
  SENDING EMAIL ID=23/02/2022 23:54:06
  SENDING EMAIL ID=23/02/2022 23:54:08
  SENDING EMAIL ID=23/02/2022 23:54:08
  SENDING EMAIL ID=23/02/2022 23:54:10
  SENDING EMAIL ID=23/02/2022 23:54:12
  SENDING EMAIL ID=23/02/2022 23:54:14
  SENDING EMAIL ID=23/02/2022 23:54:16
  SENDING EMAIL ID=23/02/2022 23:54:18
  SENDING EMAIL ID=23/02/2022 23:54:20
  SENDING EMAIL ID=23/02/2022 23:54:22
  SENDING EMAIL ID=23/02/2022 23:54:24
  SENDING EMAIL ID=23/02/2022 23:54:26
  SENDING EMAIL ID=23/02/2022 23:54:28
  SENDING EMAIL ID=23/02/2022 23:54:30
  SENDING EMAIL ID=23/02/2022 23:54:32
  SENDING EMAIL ID=23/02/2022 23:54:34
  SENDING EMAIL ID=23/02/2022 23:54:36
  SENDING EMAIL ID=23/02/2022 23:54:38
  SENDING EMAIL ID=23/02/2022 23:54:40
  SENDING EMAIL ID=23/02/2022 23:54:42
  SENDING EMAIL ID=23/02/2022 23:54:42

```

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS

- flask-push
- file code
- file env
- file __init__.py
- file application
- file __init__.py
- file config.py
- file controllers.py
- file database.py
- file models.py
- file tasks.py
- file validation.py
- file workers.py
- file __init__.py
- file static
- file bootstrap
- file img
- file app.js
- file custom.js
- file show_alerts.js
- file __init__.py
- file style.css
- file templates
- file 403.html
- file 404.html
- file article.html
- file article_by_author.html
- file create_article.html
- file feedback.html
- file show_updates.html
- file show_updates_vue.html
- file update.html
- file thank-you.html
- file celerybeat_schedule
- file debug.log
- file local_beat.sh
- file local_gunicorn.sh
- file local_run.sh
- file local_setup.sh

20 characters selected

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)



The image shows two screenshots of a Sublime Text editor window. Both screenshots display the same Python code in a file named 'tasks.py'. The code is part of a Flask application and uses Celery for periodic tasks. The code includes imports for celery, datetime, flask_sse, and celery.schedules. It defines a task to print the current time and another to send a hello message. A long-running job is also defined. The code is as follows:

```

1 import time
2 from application.workers import celery
3 from datetime import datetime
4 from flask_sse import sse
5 from celery.schedules import crontab
6
7 @celery.on_after_finalize.connect
8 def setup_periodic_tasks(sender, **kwargs):
9     ...: sender.add_periodic_task(10.0, print_current_time_job.s(), name='At every')
10
11
12 @celery.task()
13 def print_current_time_job():
14     print("START")
15     now = datetime.now()
16     print("now in task =", now)
17     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
18     sse.publish({"message": "Current Time =" + dt_string}, type='greeting')
19     print("date and time =", dt_string)
20     print("COMPLETE")
21     return dt_string
22
23
24 @celery.task()
25 def hello_world(name):
26     print("hello {}".format(name))
27
28
29
30 @celery.task()
31 def long_running_job():
32     ...: print("CTADTCN | ANC TAD")

```

The left screenshot shows the file tree on the left side of the editor, while the right screenshot shows the file tree on the right side. The code is identical in both instances.

The image shows two screenshots of a Sublime Text editor window, both displaying the same Python code for a Flask application's tasks.py file. The code defines several Celery tasks, including one that prints the current time and sends it via SSE to connected users.

```

1 import time
2 from application.workers import celery
3 from datetime import datetime
4 from flask_sse import sse
5 from celery.schedules import crontab
6
7 @celery.on_after_finalize.connect
8 def setup_periodic_tasks(sender, **kwargs):
9     sender.add_periodic_task(10.0, print_current_time_job.s(), name='At every 10 seconds')
10
11 @celery.task()
12 def print_current_time_job():
13     print("START")
14     now = datetime.now()
15     print("now in task =", now)
16     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
17     sse.publish({"message": "Current Time =" + dt_string}, type='greeting')
18     print("date and time =", dt_string)
19     print("COMPLETE")
20     return dt_string
21
22 @celery.task()
23 def hello_world(name):
24     print("hello {}".format(name))
25
26 @celery.task()
27 def long_running_job():
28     print("STARTED LONG JOB")
29
30 @celery.task()
31 def long_running_job():
32     print("STARTED LONG JOB")

```

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 image consists of three vertically stacked screenshots of a terminal window showing the output of a Flask-SSE application. The terminal is running on port 8000.

Screenshot 1: The terminal shows the command `python flask_sse/app.py` being run. The output log shows multiple `SENDING EMAIL` messages with IDs ranging from 70-54 to 70-99. The log ends with `COMPLETED EMAILING JOB` at 23:55:30.

```

Code Work /Flask-SSE/flask_sse/app.py [Flask-py] - Sublime Text
File Edit Selection Find View Goto Tools Run Help
+ C @ localhost:8000/start_log + 
Flask-SSE Quickstart
SENDING EMAIL ID=70-54
SENDING EMAIL ID=70-55
SENDING EMAIL ID=70-56
SENDING EMAIL ID=70-57
SENDING EMAIL ID=70-58
SENDING EMAIL ID=70-59
SENDING EMAIL ID=70-60
SENDING EMAIL ID=70-61
SENDING EMAIL ID=70-62
SENDING EMAIL ID=70-63
SENDING EMAIL ID=70-64
SENDING EMAIL ID=70-65
SENDING EMAIL ID=70-66
SENDING EMAIL ID=70-67
SENDING EMAIL ID=70-68
SENDING EMAIL ID=70-69
SENDING EMAIL ID=70-70
SENDING EMAIL ID=70-71
SENDING EMAIL ID=70-72
SENDING EMAIL ID=70-73
SENDING EMAIL ID=70-74
SENDING EMAIL ID=70-75
SENDING EMAIL ID=70-76
SENDING EMAIL ID=70-77
SENDING EMAIL ID=70-78
SENDING EMAIL ID=70-79
SENDING EMAIL ID=70-80
SENDING EMAIL ID=70-81
SENDING EMAIL ID=70-82
SENDING EMAIL ID=70-83
SENDING EMAIL ID=70-84
SENDING EMAIL ID=70-85
SENDING EMAIL ID=70-86
SENDING EMAIL ID=70-87
SENDING EMAIL ID=70-88
SENDING EMAIL ID=70-89
SENDING EMAIL ID=70-90
SENDING EMAIL ID=70-91
SENDING EMAIL ID=70-92
SENDING EMAIL ID=70-93
SENDING EMAIL ID=70-94
SENDING EMAIL ID=70-95
SENDING EMAIL ID=70-96
SENDING EMAIL ID=70-97
SENDING EMAIL ID=70-98
SENDING EMAIL ID=70-99
[...]
[23:55:30]

```

Screenshot 2: The terminal shows the command `curl http://localhost:8000/start_log` being run. The output log shows the same sequence of `SENDING EMAIL` messages, starting from ID 70-54 and ending with ID 70-99. The log ends with `COMPLETED EMAILING JOB` at 23:55:30.

```

curl http://localhost:8000/start_log
[23:55:30]

```

Screenshot 3: The terminal shows the command `curl http://localhost:8000/static/js/show_alerts.js` being run. The output log shows the same sequence of `SENDING EMAIL` messages, starting from ID 70-54 and ending with ID 70-99. The log ends with `COMPLETED EMAILING JOB` at 23:55:30.

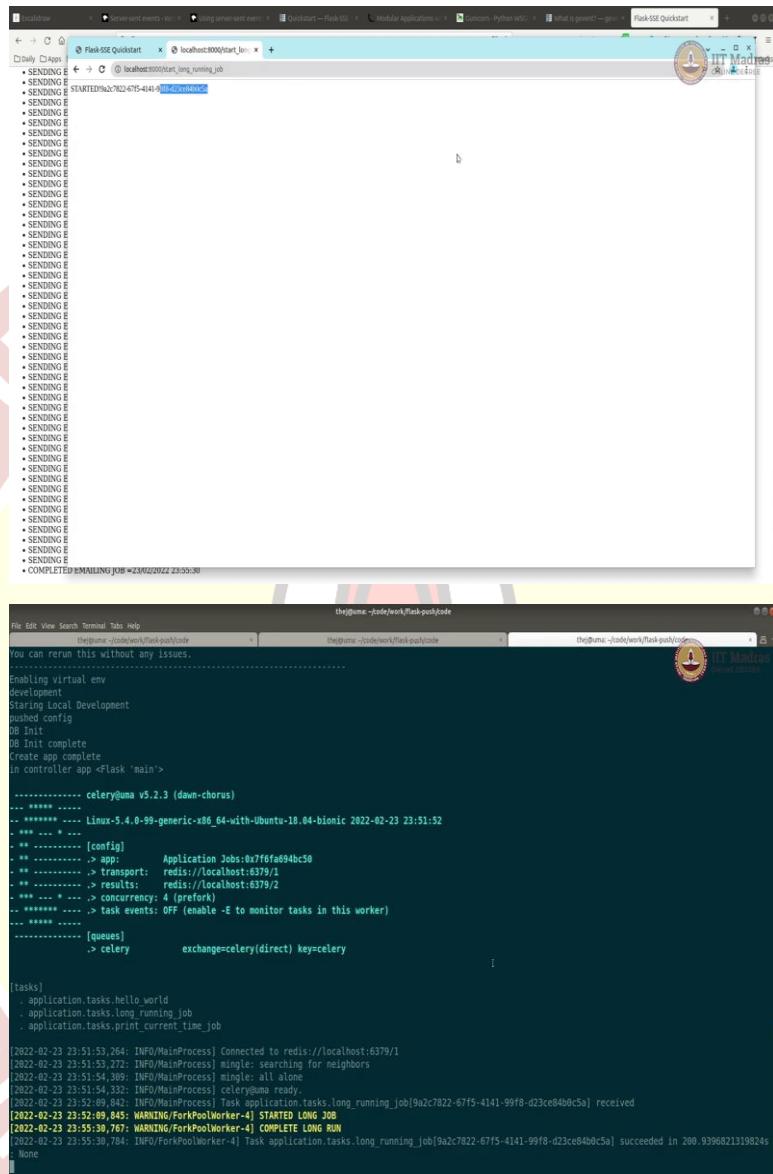
```

curl http://localhost:8000/static/js/show_alerts.js
[23:55:30]

```

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)





```

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

```

pushed config
DB Init
DB Init complete
Create app complete
ls controller app <flask 'main'

..... celery@uma v5.2.3 (down-chorus)
... ****
... ===== Linux-5.4.0-99-generic-x86_64-with-Ubuntu-18.04-bionic 2022-02-23 23:51:52
... *** * ...
... ** .. [config]
... ** ... .> app: Application Jobs:0x7f6fa694bc59
... ** ... .> 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 23:51:53,204: INFO/MainProcess] Connected to redis://localhost:6379/1
[2022-02-23 23:51:53,272: INFO/MainProcess] mingle: searching for neighbors
[2022-02-23 23:51:53,309: INFO/MainProcess] mingle: all alone
[2022-02-23 23:51:53,332: INFO/MainProcess] celery@uma ready.
[2022-02-23 23:52:09,842: INFO/MainProcess] Task application.tasks.long_running_job[9a2c7822-67f5-4141-99f8-d23ce840c5a] received
[2022-02-23 23:52:09,845: WARNING/ForkPoolWorker-4] STARTED LONG JOB
[2022-02-23 23:55:39,767: WARNING/ForkPoolWorker-4] COMPLETE LONG RUN
[2022-02-23 23:55:39,768: INFO/ForkPoolWorker-4] Task application.tasks.long_running_job[9a2c7822-67f5-4141-99f8-d23ce840c5a] succeeded in 200.9396821319824s
: None
`C
worker: Hitting Ctrl+C again will terminate all running tasks!
worker: Warm shutdown (MainProcess)

thej@uma:~/code/work/flask-push/code$ sh local_workers.sh
```



```

thej@uma:~/code/work/flask-push/code$ sh local_workers.sh
```

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\$

```

Welcome 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
developing
Starting Local Development
pushed config
DB Init
DB Init complete
Create app complete
ls controller app <flask 'main'

..... celery@uma v5.2.3 (down-chorus)
... ****
... ===== Linux-5.4.0-99-generic-x86_64-with-Ubuntu-18.04-bionic 2022-02-23 23:56:06
... *** * ...
... ** .. [config]
... ** ... .> app: Application Jobs:0x7ff6192af3b90
... ** ... .> 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 23:56:07,215: INFO/MainProcess] Connected to redis://localhost:6379/1
[2022-02-23 23:56:07,219: INFO/MainProcess] mingle: searching for neighbors
[2022-02-23 23:56:08,244: INFO/MainProcess] mingle: all alone
[2022-02-23 23:56:08,263: INFO/MainProcess] celery@uma ready.
```

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

(Refer Slide Time: 36:48)

```
thej@uma:~/code/work/flask-push/code
```

```
#0 Init complete
# Create app complete
# in controller app <flask 'main'>
celery beat v5.2.3 (dawn-chorus) is starting.

[2022-02-23 23:45:25,94] INFO [MainProcess] beat: Starting...
thej@uma:~/code/work/flask-push/code$ sh local_beat.sh
=====
welcome 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
Starting Local Development
pushed config
#0 Init
#0 Init complete
# Create app complete
# in controller app <flask 'main'>
celery beat v5.2.3 (dawn-chorus) is starting.

[2022-02-23 23:56:14,13] INFO [MainProcess] beat: Starting...
thej@uma:~/code/work/flask-push/code$ sh local_beat.sh
=====
welcome 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
Starting Local Development
pushed config
#0 Init
#0 Init complete
# Create app complete
# in controller app <flask 'main'>
celery@uma v5.2.3 (dawn-chorus)
...
[config]
... * -> app: Application Jobs:0x7f6192af3b90
... * -> 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 23:56:07,215] INFO [MainProcess] Connected to redis://localhost:6379/
[2022-02-23 23:56:07,219] INFO [MainProcess] mingle: searching for neighbors
[2022-02-23 23:56:08,244] INFO [MainProcess] mingle: all alone
[2022-02-23 23:56:08,263] INFO [MainProcess] celery@uma ready.

thej@uma:~/code/work/flask-push/code$ sh local_workers.sh
=====
welcome 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
Starting Local Development
pushed config
#0 Init
#0 Init complete
# Create app complete
# in controller app <flask 'main'>
celery@uma v5.2.3 (dawn-chorus)
...
[config]
... * -> app: Application Jobs:0x7f6192af3b90
... * -> 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 23:56:07,215] INFO [MainProcess] Connected to redis://localhost:6379/
[2022-02-23 23:56:07,219] INFO [MainProcess] mingle: searching for neighbors
[2022-02-23 23:56:08,204] INFO [MainProcess] mingle: all alone
[2022-02-23 23:56:08,233] INFO [MainProcess] celery@uma ready.
[2022-02-23 23:56:24,498] INFO [MainProcess] Task application.tasks.print_current_time_job[fe9713fb-3ad8-4a60-8cda-e107d7f79080] received
[2022-02-23 23:56:24,501] WARNING[ForkPoolWorker-4] START
[2022-02-23 23:56:24,502] WARNING[ForkPoolWorker-4] now in task =
[2022-02-23 23:56:24,502] WARNING[ForkPoolWorker-4] 2022-02-23 23:56:24.502221
[2022-02-23 23:56:24,506] WARNING[ForkPoolWorker-4] date and time =
[2022-02-23 23:56:24,506] WARNING[ForkPoolWorker-4]
[2022-02-23 23:56:24,506] WARNING[ForkPoolWorker-4] 23/02/2022 23:56:24
[2022-02-23 23:56:24,506] WARNING[ForkPoolWorker-4] COMPLETE
[2022-02-23 23:56:24,518] INFO[ForkPoolWorker-4] Task application.tasks.print_current_time_job[fe9713fb-3ad8-4a60-8cda-e107d7f79080] succeeded in 0.0176484979
[2022-02-23 23:56:24,518] INFO[ForkPoolWorker-4] Task application.tasks.print_current_time_job[fe9713fb-3ad8-4a60-8cda-e107d7f79080] succeeded in 0.0176484979
```


The image shows two terminal windows side-by-side. Both terminals are running on a Linux system with the command `the@hnu:~/code/work/task-push/code`. The left terminal displays Celery task logs, and the right terminal displays Flask-SSE logs.

```

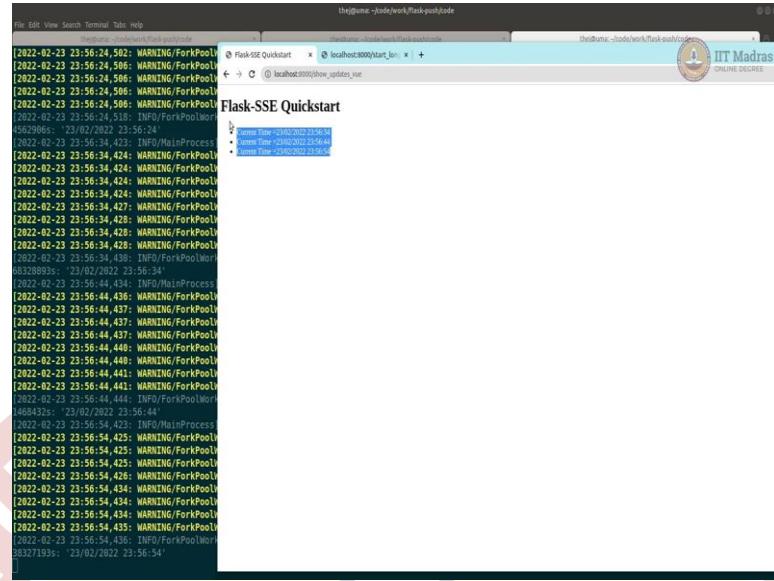
the@hnu:~/code/work/task-push/code
[2022-02-23 23:56:07,215: INFO/MainProcess] Connected to redis://localhost:6379/1
[2022-02-23 23:56:07,216: INFO/MainProcess] mingle: searching for neighbors
[2022-02-23 23:56:08,244: INFO/MainProcess] mingle: all alone
[2022-02-23 23:56:24,498: INFO/MainProcess] celery:uma ready.
[2022-02-23 23:56:24,501: INFO/MainProcess] Task application.tasks.print_current_time_job[fe9713fb-3ad8-4a60-8cda-e107d7f79088] received
[2022-02-23 23:56:24,501: WARNING/ForkPoolWorker-4] START
[2022-02-23 23:56:24,502: WARNING/ForkPoolWorker-4] now in task =
[2022-02-23 23:56:24,502: WARNING/ForkPoolWorker-4]
[2022-02-23 23:56:24,502: WARNING/ForkPoolWorker-4] date and time =
[2022-02-23 23:56:24,504: WARNING/ForkPoolWorker-4]
[2022-02-23 23:56:24,506: WARNING/ForkPoolWorker-4] 23/02/2022 23:56:24
[2022-02-23 23:56:24,506: WARNING/ForkPoolWorker-4] COMPLETE
[2022-02-23 23:56:24,518: INFO/ForkPoolWorker-4] Task application.tasks.print_current_time_job[fe9713fb-3ad8-4a60-8cda-e107d7f79088] succeeded in 0.0176484979
[45290683: 23/02/2022 23:56:24]
[2022-02-23 23:56:34,423: INFO/MainProcess] Task application.tasks.print_current_time_job[01a72ab4-1895-4350-ba2c-a7d111919f5] received
[2022-02-23 23:56:34,424: INFO/MainProcess] Task application.tasks.print_current_time_job[01a72ab4-1895-4350-ba2c-a7d111919f5] received
[2022-02-23 23:56:34,424: WARNING/ForkPoolWorker-4] START
[2022-02-23 23:56:34,424: WARNING/ForkPoolWorker-4] now in task =
[2022-02-23 23:56:34,424: WARNING/ForkPoolWorker-4]
[2022-02-23 23:56:34,424: WARNING/ForkPoolWorker-4] date and time =
[2022-02-23 23:56:34,427: WARNING/ForkPoolWorker-4]
[2022-02-23 23:56:34,428: WARNING/ForkPoolWorker-4] 23/02/2022 23:56:34
[2022-02-23 23:56:34,428: WARNING/ForkPoolWorker-4] COMPLETE
[2022-02-23 23:56:34,436: INFO/ForkPoolWorker-4] Task application.tasks.print_current_time_job[01a72ab4-1895-4350-ba2c-a7d111919f5] succeeded in 0.0063248829
[45328893: 23/02/2022 23:56:34]

the@hnu:~/code/work/task-push/code
[2022-02-23 23:56:07,215: INFO/MainProcess] Connected to redis://localhost:6379/1
[2022-02-23 23:56:07,216: INFO/MainProcess] mingle: searching for neighbors
[2022-02-23 23:56:08,244: INFO/MainProcess] mingle: all alone
[2022-02-23 23:56:24,498: INFO/MainProcess] celery:uma ready.
[2022-02-23 23:56:24,501: INFO/MainProcess] Task application.tasks.long_running_job[01a72ab4-1895-4350-ba2c-a7d111919f5] received
[2022-02-23 23:56:24,501: INFO/MainProcess] Task application.tasks.long_running_job[01a72ab4-1895-4350-ba2c-a7d111919f5] received
[2022-02-23 23:56:24,501: WARNING/ForkPoolWorker-4] START
[2022-02-23 23:56:24,502: WARNING/ForkPoolWorker-4] now in task =
[2022-02-23 23:56:24,502: WARNING/ForkPoolWorker-4]
[2022-02-23 23:56:24,502: WARNING/ForkPoolWorker-4] date and time =
[2022-02-23 23:56:24,504: WARNING/ForkPoolWorker-4]
[2022-02-23 23:56:24,506: WARNING/ForkPoolWorker-4] 23/02/2022 23:56:24
[2022-02-23 23:56:24,506: WARNING/ForkPoolWorker-4] COMPLETE
[2022-02-23 23:56:24,518: INFO/ForkPoolWorker-4] Task application.tasks.long_running_job[01a72ab4-1895-4350-ba2c-a7d111919f5] succeeded in 0.0176484979
[45290683: 23/02/2022 23:56:24]
[2022-02-23 23:56:34,423: INFO/MainProcess] Task application.tasks.long_running_job[01a72ab4-1895-4350-ba2c-a7d111919f5] received
[2022-02-23 23:56:34,424: INFO/MainProcess] Task application.tasks.long_running_job[01a72ab4-1895-4350-ba2c-a7d111919f5] received
[2022-02-23 23:56:34,424: WARNING/ForkPoolWorker-4] START
[2022-02-23 23:56:34,424: WARNING/ForkPoolWorker-4] now in task =
[2022-02-23 23:56:34,424: WARNING/ForkPoolWorker-4]
[2022-02-23 23:56:34,424: WARNING/ForkPoolWorker-4] date and time =
[2022-02-23 23:56:34,427: WARNING/ForkPoolWorker-4]
[2022-02-23 23:56:34,428: WARNING/ForkPoolWorker-4] 23/02/2022 23:56:34
[2022-02-23 23:56:34,428: WARNING/ForkPoolWorker-4] COMPLETE
[2022-02-23 23:56:34,436: INFO/ForkPoolWorker-4] Task application.tasks.long_running_job[01a72ab4-1895-4350-ba2c-a7d111919f5] succeeded in 0.0063248829
[45328893: 23/02/2022 23:56:34]

```

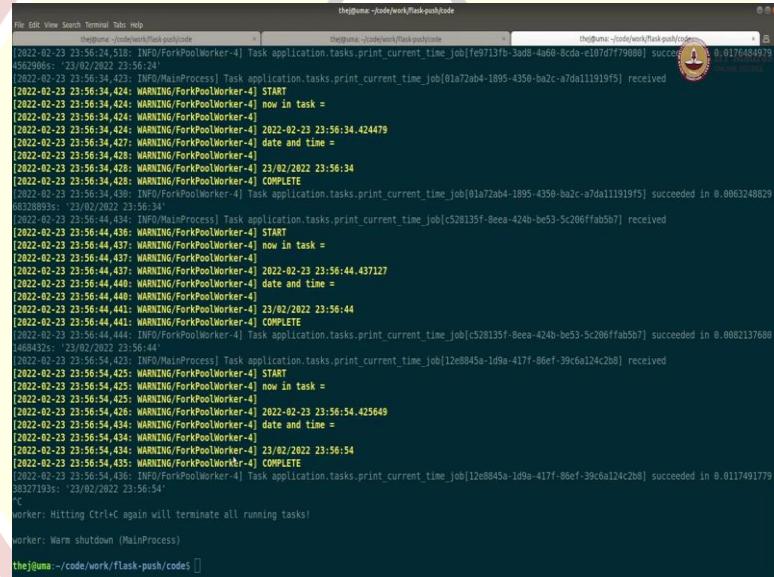
But I think let us see there is continue getting thin, there we go, rate for, there you go. This one is every 10 seconds to wait.

(Refer Slide Time: 37:29)



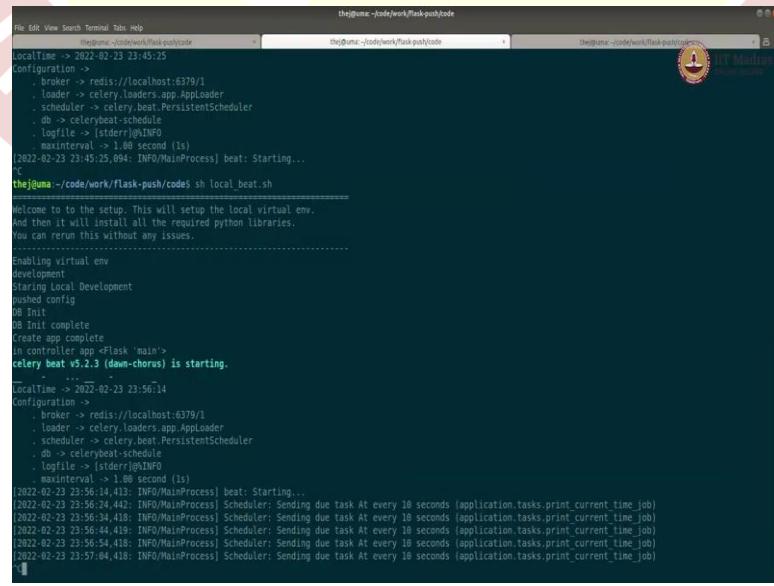
The terminal window displays log output from the Flask-SSE Quickstart application. It shows multiple entries for 'WARNING/ForkPoolWorker' with timestamp variations like 23:56:24, 24, 24.92, 24.96, and 24.98. Other log entries include 'INFO/MainProcess' and 'INFO/ForkPoolWorker' messages.

```
[the@uma:~/code/work/flask-push/code]$ flask SSE Quickstart
[2022-02-23 23:56:24,92: WARNING/ForkPoolWorker] received [2022-02-23 23:56:24,96: WARNING/ForkPoolWorker] 2022-02-23 23:56:24,96: WARNING/ForkPoolWorker
[2022-02-23 23:56:24,96: WARNING/ForkPoolWorker] 2022-02-23 23:56:24,98: WARNING/ForkPoolWorker
[2022-02-23 23:56:24,98: WARNING/ForkPoolWorker]
[the@uma:~/code/work/flask-push/code]$ the@uma:~/code/work/flask-push/code$ flask SSE Quickstart
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] received [2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker]
[the@uma:~/code/work/flask-push/code]$ flask SSE Quickstart
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] received [2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker]
[the@uma:~/code/work/flask-push/code]$ flask SSE Quickstart
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] received [2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker]
[the@uma:~/code/work/flask-push/code]$ flask SSE Quickstart
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] received [2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker]
[the@uma:~/code/work/flask-push/code]$ flask SSE Quickstart
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] received [2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker]
[the@uma:~/code/work/flask-push/code]$ flask SSE Quickstart
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] received [2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker]
[the@uma:~/code/work/flask-push/code]$ flask SSE Quickstart
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] received [2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker] 2022-02-23 23:56:24,98: INFO/ForkPoolWorker
[2022-02-23 23:56:24,98: INFO/ForkPoolWorker]
```



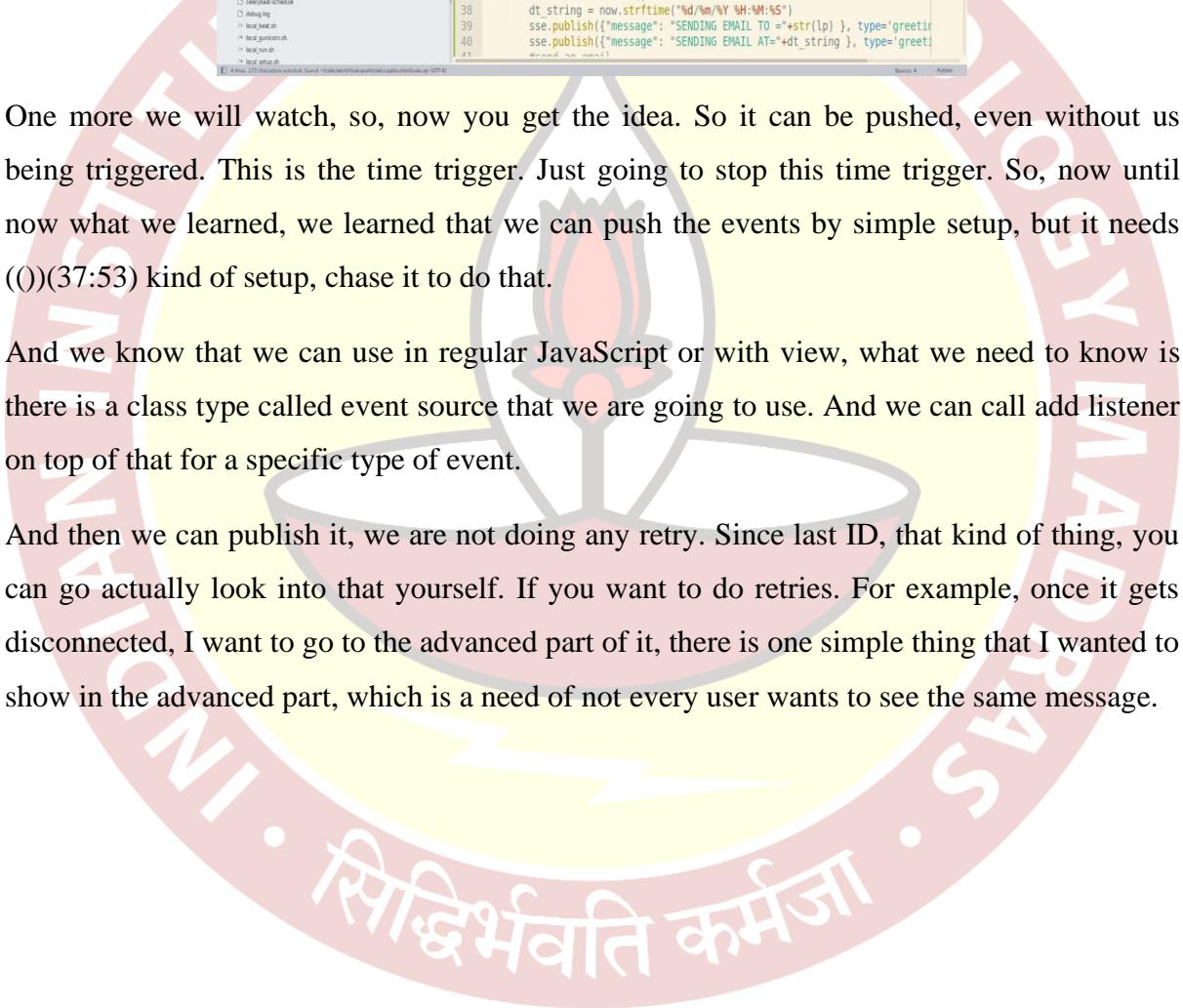
The terminal window displays log output from the Task application.tasks.print_current_time_job task. It shows multiple entries for 'WARNING/ForkPoolWorker-4' with timestamp variations like 23:56:34, 42, 42.95, and 43.0. Other log entries include 'INFO/MainProcess' and 'INFO/ForkPoolWorker-4' messages.

```
[the@uma:~/code/work/flask-push/code]$ flask SSE Quickstart
[2022-02-23 23:56:34,05: INFO/ForkPoolWorker-4] Task application.tasks.print_current_time_job[01a7ab4-1b95-4350-ba2c-a7d111919f5] succeeded in 0.0176461979
[2022-02-23 23:56:34,05: INFO/ForkPoolWorker-4] received [2022-02-23 23:56:34,42: WARNING/ForkPoolWorker-4] START
[2022-02-23 23:56:34,42: WARNING/ForkPoolWorker-4] now in task =
[2022-02-23 23:56:34,42: WARNING/ForkPoolWorker-4] 2022-02-23 23:56:34.424479
[2022-02-23 23:56:34,42: WARNING/ForkPoolWorker-4] date and time =
[2022-02-23 23:56:34,42: WARNING/ForkPoolWorker-4] 2022-02-23 23:56:34.424479
[2022-02-23 23:56:34,42: WARNING/ForkPoolWorker-4] Task application.tasks.print_current_time_job[01a7ab4-1b95-4350-ba2c-a7d111919f5] succeeded in 0.0176461979
[2022-02-23 23:56:34,42: WARNING/ForkPoolWorker-4] received [2022-02-23 23:56:34,43: INFO/MainProcess] Task application.tasks.print_current_time_job[c520135f-8eea-424b-be53-5c206ffab5b7] received [2022-02-23 23:56:34,43: INFO/ForkPoolWorker-4] now in task =
[2022-02-23 23:56:34,43: INFO/ForkPoolWorker-4] 2022-02-23 23:56:34.43127
[2022-02-23 23:56:34,43: INFO/ForkPoolWorker-4] date and time =
[2022-02-23 23:56:34,43: INFO/ForkPoolWorker-4] 2022-02-23 23:56:34.43127
[2022-02-23 23:56:34,43: INFO/ForkPoolWorker-4] Task application.tasks.print_current_time_job[c520135f-8eea-424b-be53-5c206ffab5b7] succeeded in 0.0063248829
[2022-02-23 23:56:34,43: INFO/ForkPoolWorker-4] received [2022-02-23 23:56:34,44: WARNING/ForkPoolWorker-4] COMPLETE
[2022-02-23 23:56:34,44: WARNING/ForkPoolWorker-4] now in task =
[2022-02-23 23:56:34,44: WARNING/ForkPoolWorker-4] 2022-02-23 23:56:34.441
[2022-02-23 23:56:34,44: WARNING/ForkPoolWorker-4] Task application.tasks.print_current_time_job[c520135f-8eea-424b-be53-5c206ffab5b7] succeeded in 0.0082137680
[2022-02-23 23:56:34,44: WARNING/ForkPoolWorker-4] received [2022-02-23 23:56:34,45: INFO/MainProcess] Task application.tasks.print_current_time_job[12e0845a-1d9a-417f-8def-39c6a124c2b8] received [2022-02-23 23:56:34,45: INFO/ForkPoolWorker-4] now in task =
[2022-02-23 23:56:34,45: INFO/ForkPoolWorker-4] 2022-02-23 23:56:34.452649
[2022-02-23 23:56:34,45: INFO/ForkPoolWorker-4] date and time =
[2022-02-23 23:56:34,45: INFO/ForkPoolWorker-4] 2022-02-23 23:56:34.452649
[2022-02-23 23:56:34,45: INFO/ForkPoolWorker-4] Task application.tasks.print_current_time_job[12e0845a-1d9a-417f-8def-39c6a124c2b8] succeeded in 0.0082137680
[2022-02-23 23:56:34,45: INFO/ForkPoolWorker-4] received [2022-02-23 23:56:34,46: WARNING/ForkPoolWorker-4] COMPLETE
[2022-02-23 23:56:34,46: WARNING/ForkPoolWorker-4] now in task =
[2022-02-23 23:56:34,46: WARNING/ForkPoolWorker-4] 2022-02-23 23:56:34.461
[2022-02-23 23:56:34,46: WARNING/ForkPoolWorker-4] Task application.tasks.print_current_time_job[12e0845a-1d9a-417f-8def-39c6a124c2b8] succeeded in 0.0117491779
[2022-02-23 23:56:34,46: WARNING/ForkPoolWorker-4] received [2022-02-23 23:56:34,47: INFO/MainProcess] Task application.tasks.print_current_time_job[12e0845a-1d9a-417f-8def-39c6a124c2b8] received [2022-02-23 23:56:34,47: INFO/ForkPoolWorker-4] now in task =
[2022-02-23 23:56:34,47: INFO/ForkPoolWorker-4] 2022-02-23 23:56:34.471
[2022-02-23 23:56:34,47: INFO/ForkPoolWorker-4] date and time =
[2022-02-23 23:56:34,47: INFO/ForkPoolWorker-4] 2022-02-23 23:56:34.471
[2022-02-23 23:56:34,47: INFO/ForkPoolWorker-4] Task application.tasks.print_current_time_job[12e0845a-1d9a-417f-8def-39c6a124c2b8] succeeded in 0.0117491779
[2022-02-23 23:56:34,47: INFO/ForkPoolWorker-4] received [2022-02-23 23:56:34,48: WARNING/ForkPoolWorker-4] COMPLETE
[2022-02-23 23:56:34,48: WARNING/ForkPoolWorker-4] now in task =
[2022-02-23 23:56:34,48: WARNING/ForkPoolWorker-4] 2022-02-23 23:56:34.481
[2022-02-23 23:56:34,48: WARNING/ForkPoolWorker-4] Task application.tasks.print_current_time_job[12e0845a-1d9a-417f-8def-39c6a124c2b8] succeeded in 0.0117491779
[2022-02-23 23:56:34,48: WARNING/ForkPoolWorker-4] received
```



The terminal window displays log output from the local_beat.sh script. It shows configuration details for Celery, including Redis connection, loader (celery.loaders.app.AppLoader), scheduler (celery.beat.PersistentScheduler), and beat interval (1.00 second). The beat task starts at 23:56:14.

```
[the@uma:~/code/work/flask-push/code]$ local_beat.sh
=====
Welcome 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
Activating env
Starting Local Development
pushed config
DB Init
DB Init complete
Create app complete
in controller app <flask 'main'>
celery beat v5.2.3 (dawn-chorus) is starting.
=====
LocalTime -> 2022-02-23 23:56:14
Configuration -
  . broker -> redis://localhost:6379/1
  . loader -> celery.loaders.app.AppLoader
  . scheduler -> celery.beat.PersistentScheduler
  . db -> celerybeat-schedule
  . logfile -> [stderr@INFO
  maxinterval -> 1.00 second (1s)
[2022-02-23 23:45:25,094: INFO/MainProcess] beat: Starting...
`C
[the@uma:~/code/work/flask-push/code]$ sh local_beat.sh
=====
LocalTime -> 2022-02-23 23:56:14
Configuration -
  . broker -> redis://localhost:6379/1
  . loader -> celery.loaders.app.AppLoader
  . scheduler -> celery.beat.PersistentScheduler
  . db -> celerybeat-schedule
  . logfile -> [stderr@INFO
  maxinterval -> 1.00 second (1s)
[2022-02-23 23:56:14,13: INFO/MainProcess] beat: Starting...
[2022-02-23 23:56:24,44: INFO/MainProcess] Scheduler: Sending due task At every 10 seconds (application.tasks.print_current_time_job)
[2022-02-23 23:56:34,41: INFO/MainProcess] Scheduler: Sending due task At every 10 seconds (application.tasks.print_current_time_job)
[2022-02-23 23:56:44,41: INFO/MainProcess] Scheduler: Sending due task At every 10 seconds (application.tasks.print_current_time_job)
[2022-02-23 23:56:54,41: INFO/MainProcess] Scheduler: Sending due task At every 10 seconds (application.tasks.print_current_time_job)
[2022-02-23 23:57:04,41: INFO/MainProcess] Scheduler: Sending due task At every 10 seconds (application.tasks.print_current_time_job)
```



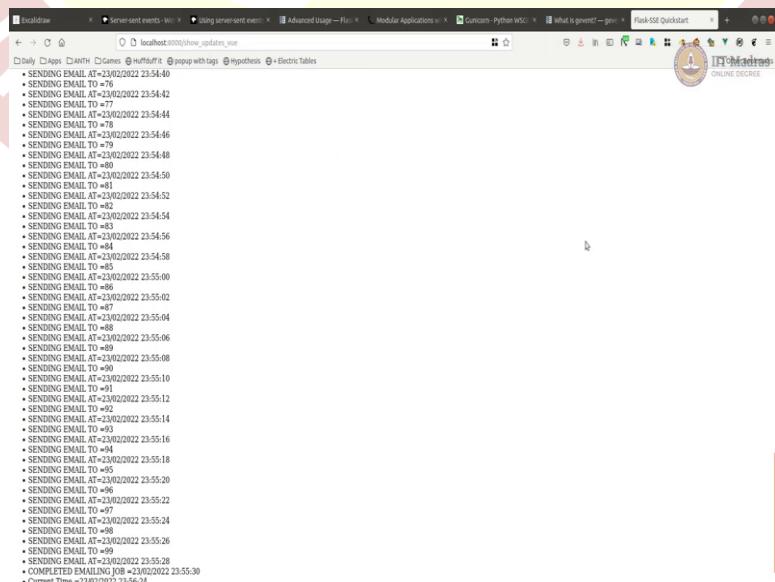
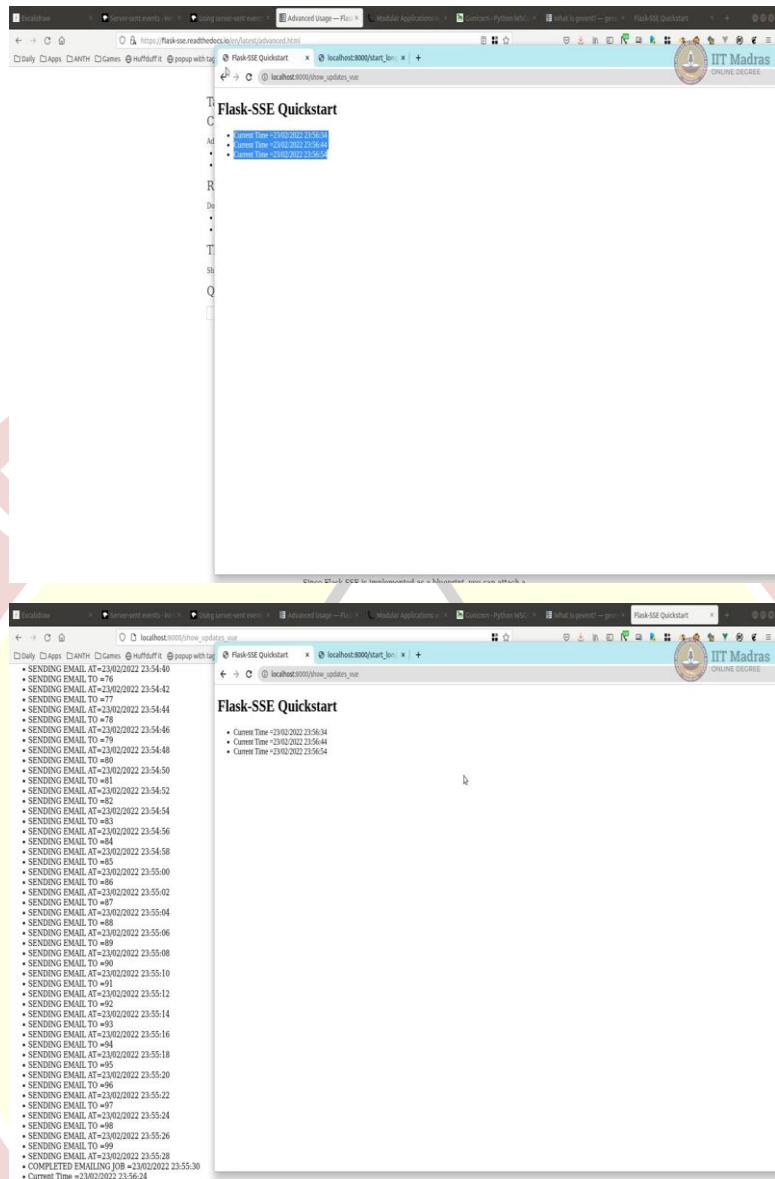
```
-/code/work/flask-push/code/application/tasks.py [flask-push] - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
• ill-fish-pgh
• ill-code
• ill-env
• ill_celache_
• ill_application
• ill_zcache_
• ill_zhttpd_py
• ill_zhttpd
• ill_zhttp
• ill_zcache
• ill_zcache_py
• ill_controllers
• ill_database
• ill_models
• ill_tasks
• ill_validation
• ill_zhttp_py
• ill_zhttpd_directory
• ill_zhttpd
• ill_zhttpd_bootstrap
• ill_zhttpd_img
• ill_zhttpd_js
• ill_zhttpd_app_js
• ill_zhttpd_custom_js
• ill_zhttpd_show_alert_js
• ill_zhttpd_style_css
• ill_zhttpd_templates
• ill_zhttpd_403_html
• ill_zhttpd_404_html
• ill_zhttpd_articles_html
• ill_zhttpd_by_author_html
• ill_zhttpd_error_404_html
• ill_zhttpd_feedback_html
• ill_zhttpd_github_html
• ill_zhttpd_update_vue_html
• ill_zhttpd_update_html
• ill_zhttpd_thank_you_html
• ill_zhttpd_schedule
• ill_zhttpd_debug_js
• ill_zhttpd_all_js
• ill_zhttpd_gunicorn_js
• ill_zhttpd_gunicorn_sh
• ill_zhttpd_setup_js
• ill_zhttpd_teardown_js
10
11
12 @celery.task()
13 def print_current_time_job():
14     print("START")
15     now = datetime.now()
16     print("now in task =", now)
17     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
18     sse.publish({"message": "Current Time =" +dt_string }, type='greeting')
19     print("date and time =", dt_string)
20     print("COMPLETE")
21     return dt_string
22
23
24 @celery.task()
25 def hello_world(name):
26     print("hello {}".format(name))
27
28
29
30 @celery.task()
31 def long_running_job():
32     print("STARTED LONG JOB")
33     now = datetime.now()
34     dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
35     sse.publish({"message": "STARTED EMAILING JOB =" +dt_string }, type='greet')
36     for lp in range(100):
37         now = datetime.now()
38         dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
39         sse.publish({"message": "SENDING EMAIL TO =" +str(lp) }, type='greet')
40         sse.publish({"message": "SENDING EMAIL AT=" +dt_string }, type='greet')
41 #end on point
```

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)

The screenshot shows the gevent project's homepage. The main heading is "What is gevent?". Below it, a paragraph explains that gevent is a coroutine-based Python networking library that uses greenlet to provide a high-level synchronous API on top of the libev or libuv event loop. A bulleted list of features includes:

- Fast event loop based on libev or libuv.
- Lightweight execution units based on greenlets.
- API that reuses concepts from the Python standard library (for example there are events and queues).
- Cooperative sockets with SSL support.
- Cooperative DNS queries performed through a threadpool, dmapython, or c-ares.
- Monkey patching utility to get 3rd party modules to become cooperative.
- TORNADO/IRITIMER servers.
- Subprocess support (through `gevent.subprocess`)
- Thread pools

Below this, a note states: "gevent is inspired by eventlet but features a more consistent API, simpler implementation and better performance. Read why others use gevent and check out the list of the open source projects based on gevent."

At the bottom of the page, it says: "gevent was written by Denis Bevko. Since version 1.1, gevent is maintained by Jason Madden for NextThought with help from the contributors and is licensed under the MIT license. See what's new in the latest major release. Check out the detailed changelog for this version. Continue reading > If you like gevent, donate to support the development." Navigation links at the bottom include: Blog | Code | Docs | Download | Mailing list | Issue tracker | IRC | Page Top ↑

The screenshot shows the Flask-SSE documentation page. The main heading is "Advanced Usage". Below it, a section titled "Channels" contains the following text: "Sometimes, you may not want all events to be published to all clients. For example, a client that cares about receiving the latest updates in their social network probably doesn't care about receiving the latest statistics about how many users are online across the entire site, and vice versa. When publishing an event, you can select which channel to direct the event to. If you do, only clients that are checking that particular channel will receive the event. For example, this event will be sent to the "users.social" channel." Below this, code examples are shown:

```
sse.publish("user": "alice", "status": "Life is good!", channel="users.social")
```

And this event will be sent to the "analytics" channel:

```
sse.publish("active users": 100, channel="analytics")
```

Channel names can be any string you want, and are created dynamically as soon as they are referenced. The default channel name that Flask-SSE uses is "sse". For more information, see the documentation for the Redis publish system.

To subscribe to a channel, the client only needs to provide a channel query parameter when connecting to the event stream. For example, if your event stream is at /stream, you can connect to the "users.social" channel by using the URL /stream?channel=users.social. You can also use Flask's `url_for()` function to generate this query parameter, like so:

```
url_for("sse.stream", channel="users.social")
```

By default, all channels are publicly accessible to all users. However, see the next section to change that.

-code/work/task-push/code/application/controllers.py (task-push) - Sublime Text

```

FOLDERS
├── ill_code
│   ├── ill_env
│   ├── ill_gunicorn_
│   ├── ill_application
│   └── ill_ml_
├── ill_ml_
├── ill_tasks_
├── ill_validation_
└── ill_views_
├── ill_dbs_directory
├── ill_static
├── ill_bootstrap
├── ill_img
├── ill_js
    ├── app.js
    ├── custom.js
    ├── show_alerts.js
└── ill_styles
    ├── style.css
    └── templates
        ├── 403.html
        ├── 404.html
        ├── article.html
        ├── article_by_author.html
        ├── article_by_id.html
        ├── feedback.html
        ├── show_gist.html
        ├── show_updates.html
        └── update.html
            ├── thank_you.html
            ├── calendar_schedule
            ├── debug_log
            ├── local_ml.html
            ├── local_gunicorn.sh
            ├── local_ml.sh
            └── local_setup.sh

File Edit Selection Find View Goto Tools Project Preferences Help

```

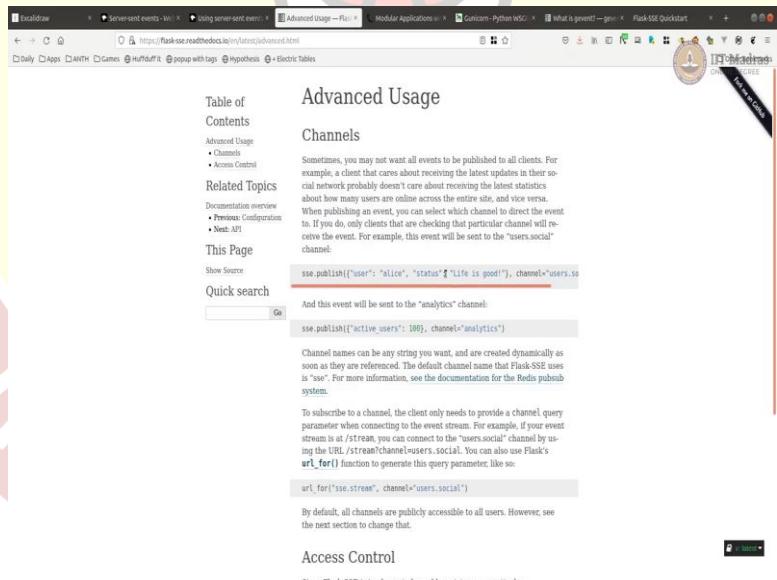
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83

```

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)



```

File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
  • iit_taskpush
    • __init__.py
    • __code__
    • __env__
    • __cache__
    • __application__
      • __init__.py
      • __init__.py
    • __controllers__
      - controllers.py
    • __database__
      - database.py
    • __models__
      - models.py
    • __tasks__
      - tasks.py
    • __validation__
      - validation.py
    • __views__
      - views.py
  • __init__.py
  • __static__
  • __templates__
    • 403.html
    • 404.html
    • article.html
    • article_by_author.html
    • article_by_id.html
    • feedback.html
    • show_update.html
    • show_update_vue.html
    • update.html
    • thank_you.html
  • config.py
  • env.py
  • __bootstrap__
  • __img__
  • __js__
    • app.js
    • custom.js
    • show_alert.js
  • __css__
    - style.css
  • __templates__
    • 403.html
    • 404.html
    • article.html
    • article_by_author.html
    • article_by_id.html
    • feedback.html
    • show_update.html
    • show_update_vue.html
    • update.html
    • thank_you.html
  • config.py
  • env.py
  • __bootstrap__
  • __img__
  • __js__
    • app.js
    • custom.js
    • show_alert.js
  • __css__
    - style.css

app.route("/test_send_message", methods=["GET", "POST"])
def test_send_message():
    user_name = "thejeshgn"
    sse.publish({"message": "Hello!", type='greeting', channel=user_name})
    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)

```

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)

The screenshot shows a browser window displaying a Flask SSE Quickstart page. The URL is <https://flask-sse.readthedocs.io/en/latest/quickstart.html>. The page contains code examples for publishing events to a channel:

```

sse.publish({'active_users': 100}, channel="analytics")

```

A note explains that channel names can be any string you want, and are created dynamically as soon as they are referenced. The default channel name that Flask-SSE uses is "sse".

```

url_for('sse.stream', channel='users.social')

```

A note states that by default, all channels are publicly accessible to all users. However, see the next section to change that.

Access Control

Since Flask-SSE is implemented as a blueprint, you can attach a `before_request()` handler to implement access control. For example:

```

@socketio.event
def check_access():
    if request.args.get('channel') == 'analytics' and not g.user.is_admin:
        abort(403)

app.register_blueprint(sse, url_prefix='/sse')

```

Warning:

When defining a `before_request()` handler, the blueprint must be registered after the handler is defined! Otherwise, the handler will have no effect.

INDIAN INSTITUTE

LOGY MADRAS

```
-jcode/work/task-push/code/main.py [Task-push] - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
└─ application
    └─ __init__.py
    └─ celery.py
    └─ config.py
    └─ database.py
    └─ models.py
    └─ tasks.py
    └─ validation.py
    └─ workers.py
    └─ static
        └─ bootstrap
        └─ img
            └─ app.js
            └─ custom.js
            └─ show_alerts.js
        └─ style.css
    └─ templates
        └─ 403.html
        └─ 404.html
        └─ article.html
        └─ article_by_author.html
        └─ create_article.html
        └─ index.html
        └─ show_update.html
        └─ update.html
        └─ thank_you.html
    └─ celerybeat_schedule
    └─ debug.log
    └─ local_auth.sh
    └─ local_gunicorn.sh
    └─ local_gun.sh
    └─ local_setup.sh
    └─ local_workers.sh
    └─ main.py
    └─ README.md
    └─ requirements.txt
    └─ README.md

[Line 79, Column 1]
```

```
result = celery.Task = workers.ContextTask
m荪 = app.app_context().push()
print("Create app complete")
return app, api, celery

app, api, celery = create_app()

# 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")

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

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

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

```
-jcode/work/task-push/code/main.py [Task-push] - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
└─ application
    └─ __init__.py
    └─ celery.py
    └─ config.py
    └─ controllers.py
    └─ database.py
    └─ models.py
    └─ workers.py
    └─ static
        └─ bootstrap
        └─ img
            └─ app.js
            └─ custom.js
            └─ show_alerts.js
        └─ style.css
    └─ templates
        └─ 403.html
        └─ 404.html
        └─ article.html
        └─ article_by_author.html
        └─ create_article.html
        └─ feedback.html
        └─ show_update.html
        └─ update.html
        └─ update.html
        └─ update.html
    └─ celerybeat_schedule
    └─ debug.log
    └─ local_auth.sh
    └─ local_gunicorn.sh
    └─ local_gun.sh
    └─ local_setup.sh
    └─ local_workers.sh
    └─ main.py
    └─ README.md
    └─ requirements.txt
    └─ README.md

[Line 79, Column 1]
```

```
celery.Task = workers.ContextTask
m荪 = app.app_context().push()
print("Create app complete")
return app, api, celery

app, api, celery = create_app()

# 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")

@sse.before_request
def check_access():
    if request.args.get("channel") == "analytics" and not g.user.is_admin():
        abort(403)

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

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

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

```
-jcode/work/task-push/code/main.py [Task-push] - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
└─ application
    └─ __init__.py
    └─ celery.py
    └─ config.py
    └─ controllers.py
    └─ database.py
    └─ models.py
    └─ tasks.py
    └─ validation.py
    └─ workers.py
    └─ static
        └─ bootstrap
        └─ img
            └─ app.js
            └─ custom.js
            └─ show_alerts.js
        └─ style.css
    └─ templates
        └─ 403.html
        └─ 404.html
        └─ article.html
        └─ article_by_author.html
        └─ create_article.html
        └─ feedback.html
        └─ show_update.html
        └─ update.html
        └─ update.html
        └─ update.html
    └─ celerybeat_schedule
    └─ debug.log
    └─ local_auth.sh
    └─ local_gunicorn.sh
    └─ local_gun.sh
    └─ local_setup.sh
    └─ local_workers.sh
    └─ main.py
    └─ README.md
    └─ requirements.txt
    └─ README.md

[Line 79, Column 1]
```

```
celery.Task = workers.ContextTask
m荪 = app.app_context().push()
print("Create app complete")
return app, api, celery

app, api, celery = create_app()

# 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")

@sse.before_request
def check_access():
    if request.args.get("channel") == "analytics" and not g.user.is_admin():
        abort(403)

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

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

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

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)

```

File Edit Selection Find View Goto Tools Project Preferences Help
File application
  - __init__.py
  - config.py
  - controllers.py
  - database.py
  - models.py
  - tasks.py
  - validation.py
  - workers.py
  - db_directory
  - static
    - bootstrap
    - img
    - js
      - app.js
      - custom.js
      - show_alerts.js
  - templates
    - 403.html
    - 404.html
    - article.html
    - articles_by_author.html
    - create_article.html
    - feedback.html
    - show_update_use.html
    - update.html
    - thank_you.html
    - celery_beat_schedule
    - debug_log
    - local_beat.sh
    - local_gunicorn.sh
    - local_uwsgi.sh
    - local_workers.sh
    - migrate.sh
    - misc.py
    - README.md
    - requirements.txt
    - requirements.md
  - tests

```

```

Line 79, Column 5
File Edit Selection Find View Goto Tools Project Preferences Help
File application
  - __init__.py
  - config.py
  - controllers.py
  - database.py
  - models.py
  - tasks.py
  - validation.py
  - workers.py
  - db_directory
  - static
    - bootstrap
    - img
    - js
      - app.js
      - custom.js
      - show_alerts.js
  - templates
    - 403.html
    - 404.html
    - article.html
    - articles_by_author.html
    - create_article.html
    - feedback.html
    - show_update_use.html
    - update.html
    - thank_you.html
    - celery_beat_schedule
    - debug_log
    - local_beat.sh
    - local_gunicorn.sh
    - local_uwsgi.sh
    - local_workers.sh
    - misc.py
    - README.md
    - requirements.txt
    - requirements.md
  - tests

Line 79, Column 5

```

```

+--jude/work/Task-push/code/main.py - [Task-push] - Sublime Text
+--jude/work/Task-push/code/main.py - [Task-push] - Sublime Text
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/")
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
80     if request.args.get("channel") == "analytics" and not g.user.is_admin():
81         abort(403)
82
83 # This is for streaming
84 app.register_blueprint(sse, url_prefix='/stream')
85
86 # Import all the controllers so they are loaded
87 from application.controllers import *
88
89 @app.errorhandler(404)
90 def page_not_found(e):
91     # note that we set the 404 status explicitly

```

```

+--jude/work/Task-push/code/main.py - [Task-push] - Sublime Text
+--jude/work/Task-push/code/main.py - [Task-push] - Sublime Text
60     celery_task = current_app.CELERY_TASK
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/")
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
80     if request.args.get("channel") == user_name
81     ...if request.args.get("channel") == "analytics" and not g.user.is_admin():
82         abort(403)
83
84 # This is for streaming
85 app.register_blueprint(sse, url_prefix='/stream')
86
87 # Import all the controllers so they are loaded
88 from application.controllers import *
89
90 @app.errorhandler(404)
91 def page_not_found(e):
92     # note that we set the 404 status explicitly

```



```

File Edit Selection Find View Goto Tools Project Preferences Help
File application
    -> __init__.py
    -> celery.py
    -> config.py
    -> controllers.py
    -> database.py
    -> models.py
    -> validation.py
    -> workers.py
    -> wsgi.py
    -> wsgi_directory
    -> wsgi_error
    -> wsgi_log
    -> wsgi.py
    -> app.js
    -> custom.js
    -> show_urls.js
    -> style.css
    -> templates
        -> 403.html
        -> 404.html
        -> article.html
        -> article_by_author.html
        -> create_article.html
        -> login.html
        -> show_update.html
        -> show_update_yml.html
        -> update.html
        -> thank_you.html
        -> celerybeat_schedule
        -> debug.log
        -> log_error.log
        -> log_gunicorn.log
        -> log_gunicorn.sh
        -> local_setup.sh
        -> local_workers.sh
    -> __main__.py
    -> README.md
    -> requirements.txt
    -> requirements.md

```

```

Line 90, Column 19
60     celery_task = celery.Task()
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",
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") == user_name:
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

```

Or it could go with like if channel security is equal to user name. Only then allow hitl, 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)



The screenshot shows a web browser window displaying a Flask-SSE Quickstart guide. The page content includes:

- A snippet of code showing how to publish data to a specific channel:

```
sse.publish({"active_users": 100}, channel="analytics")
```

- An explanation of channel names:

Channel names can be any string you want, and are created dynamically as soon as they are referenced. The default channel name that Flask-SSE uses is "sse". For more information, see the documentation for the Redis pubsub system.

- A note about access control:

To subscribe to a channel, the client only needs to provide a channel query parameter when connecting to the event stream. For example, if your event stream is at /stream, you can connect to the "users/social" channel by using the URL /stream?channel=users.social. You can also use Flask's `url_for` function to generate this query parameter, like so:

```
url_for("sse.stream", channel="users.social")
```

- A warning about defining `before_request` handlers:

By default, all channels are publicly accessible to all users. However, see the next section to change that.

Access Control

Since Flask-SSE is implemented as a blueprint, you can attach a `before_request` handler to implement access control. For example:

```
def before_request():
    def check_access():
        if request.args.get("channel") == "analytics" and not g.user.is_admin:
            abort(403)

app.register_blueprint(sse, url_prefix="/sse")
```

Warning:
When defining a `before_request()` handler, the blueprint must be registered after the handler is defined! Otherwise, the handler will have no effect.

And this event will be sent to the "analytics" channel:

```
sse.publish("active_users", 100), channel="analytics"
```

Channel names can be any string you want, and are created dynamically as soon as they are referenced. The default channel name that Flask-SSE uses is "sse". For more information, see the documentation for the Redis pubsub system.

To subscribe to a channel, the client only needs to provide a channel query parameter when connecting to the event stream. For example, if your event stream is at /stream, you can connect to the "users.social" channel by using the URL /stream?channel=users.social. You can also use Flask's `url_for()` function to generate this query parameter, like so:

```
url_for('sse.stream', channel='users.social')
```

By default, all channels are publicly accessible to all users. However, see the next section to change that.

Access Control

Since Flask-SSE is implemented as a blueprint, you can attach a `before_request()` handler to implement access control. For example:

```
@sse.before_request
def check_access():
    if request.args.get("channel") == "analytics" and not g.user.is_admin():
        abort(403)

app.register_blueprint(sse, url_prefix='/sse')
```

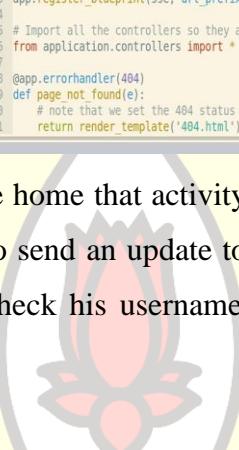
Warning:

When defining a `before_request()` handler, the blueprint must be registered after the handler is defined! Otherwise, the handler will have no effect.

Copyright 2016, David Beazley. Created using sphinx.

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)

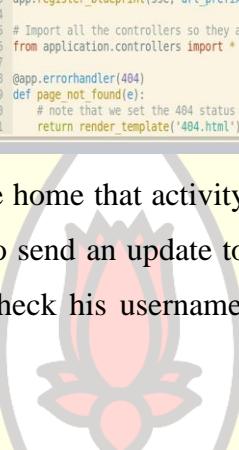


A screenshot of a Sublime Text editor window. The title bar says "code/work/flask-push/code/main.py [flask-push] - Sublime Text". The left sidebar shows a file tree with files like __init__.py, celery.py, app.py, controllers.py, database.py, models.py, tasks.py, validation.py, workers.py, config.py, static.css, and templates (404.html, article.html, articles_by_author.html, create_article.html, delete.html, show_update.html, show_update_vue.html, update.html). The main pane contains Python code:

```
50    celery = Celery('task-pushing')
51    app.app_context().push()
52    print("Create app complete")
53    return app, api, celery
54
55 app, api, celery = create_app()
56
57 # Add all restful controllers
58 from application.api import ArticleLikesAPI
59 api.add_resource(ArticleLikesAPI, "/api/article_likes", "/api/article_likes")
60
61 from application.api import TestAPI
62 api.add_resource(TestAPI, "/api/test")
63
64 @sse.before_request
65 def check_access():
66     if request.args.get("channel") == user_name:
67         abort(403)
68
69     # This is for streaming
70     app.register_blueprint(sse, url_prefix='/stream')
71
72     # Import all the controllers so they are loaded
73     from application.controllers import *
74
75 @app.errorhandler(404)
76 def page_not_found(e):
77     # note that we set the 404 status explicitly
78     return render_template('404.html'), 404
```

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)



A screenshot of a Sublime Text editor window. The title bar says "code/work/flask-push/code/templates/show_update_vue.html [flask-push] - Sublime Text". The left sidebar shows a file tree with files like __init__.py, celery.py, app.py, controllers.py, database.py, models.py, tasks.py, validation.py, workers.py, config.py, static.css, and templates (404.html, article.html, articles_by_author.html, create_article.html, delete.html, show_update.html, show_update_vue.html, update.html). The main pane contains HTML code:

```
<!DOCTYPE html>
<html>
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
</head>
<title>Flask-SSE Quickstart</title>
</head>
<body>
<h1>Flask-SSE Quickstart</h1>
<div class="container" id="app">
</div>
</body>
<script type="text/javascript" src="{{ url_for('static', filename='js/show_alerts.js') }}></script>
</html>
```

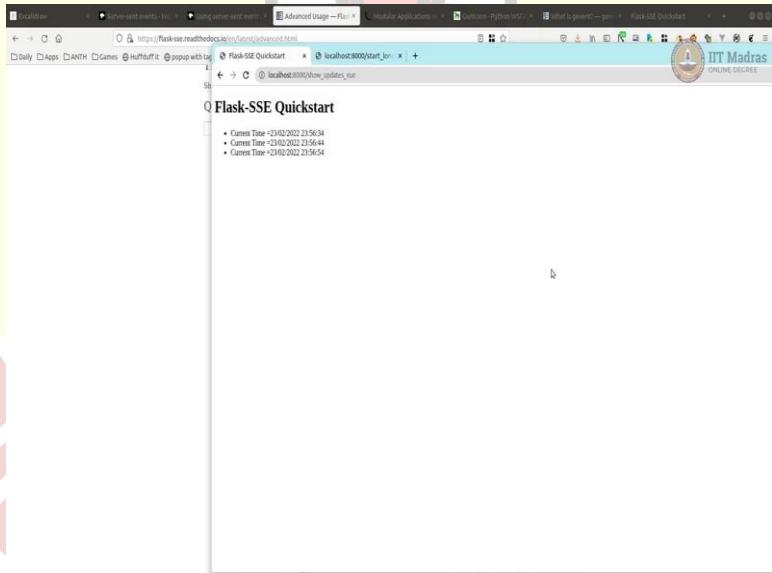
```
File Edit Selection Find View Goto Tools Project Preferences Help
  ▾ application
    ▾ __init__.py
    ▾ config.py
    ▾ database.py
    ▾ models.py
    ▾ routes.py
    ▾ validation.py
    ▾ views.py
  ▾ __init__.py
  ▾ __main__.py
  ▾ __pycache__
    ▾ __init__.pyc
    ▾ config.pyc
    ▾ database.pyc
    ▾ models.pyc
    ▾ routes.pyc
    ▾ validation.pyc
    ▾ views.pyc
  ▾ static
    ▾ bootstrap
    ▾ img
    ▾ js
      ▾ app.js
      ▾ custom.js
      ▾ show_user.js
    ▾ style.css
  ▾ templates
    ▾ 403.html
    ▾ 404.html
    ▾ article.html
    ▾ articles_by_author.html
    ▾ create_article.html
    ▾ index.html
    ▾ show_update.html
    ▾ show_update_use.html
    ▾ update.html
    ▾ user.html
  ▾ config.py
  ▾ database.py
  ▾ __main__.py
  ▾ requirements.txt
  ▾ README.md
```

Sublime Text - /code/work/flask-push/code/templates/show_update.html (flask-push) - Sublime Text

```
<!DOCTYPE html>
<html>
<head>
<title>Flask-SSE Quickstart</title>
</head>
<body>
<h1>Flask-SSE Quickstart</h1>
<script>
var source = new EventSource('{{ url_for('sse.stream') }}');
source.addEventListener('greeting', function(event) {
  var data = JSON.parse(event.data);
  alert("The server says " + data.message);
}, false);
source.addEventListener('error', function(event) {
  alert("Failed to connect to event stream. Is Redis running?");
}, false);
source.addEventListener('score', function(event) {
  alert("We got score");
}, false);
</script>
</body>
</html>
```

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)



The image shows two screenshots of a Sublime Text editor window. Both windows are titled 'show_updates.html • (task-push) - Sublime Text'. The left window shows the file structure on the left pane, which includes files like __init__.py, config.py, controllers.py, models.py, validation.py, style.css, templates, static, img, and various HTML files such as 403.html, 404.html, article.html, article_by_author.html, create_article.html, feedback.html, show_updates.html, and show_updates_vue.html. The right pane displays the code for 'show_updates.html'. The code is a JavaScript snippet using EventSource to connect to a Redis stream named 'sse.stream' under the channel 'users.social'. It handles three types of events: 'greeting', 'error', and 'score'. The 'greeting' event triggers an alert with the server's message. The 'error' event triggers an alert if the connection fails. The 'score' event triggers an alert with the message 'We got score'. The code is as follows:

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Flask-SSE Quickstart</title>
5 </head>
6 <body>
7   <h1>Flask-SSE Quickstart</h1>
8   <script>
9     var source = new EventSource("{ url_for('sse.stream', channel='users.
social') }");
10    source.addEventListener('greeting', function(event) {
11      var data = JSON.parse(event.data);
12      alert("The server says " + data.message);
13    }, false);
14    source.addEventListener('error', function(event) {
15      alert("Failed to connect to event stream. Is Redis running?");
16    }, false);
17    source.addEventListener('score', function(event) {
18      alert("We got score");
19    }, false);
20  </script>
21 </body>
22 </html>

```

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)

The screenshot shows a series of three vertically stacked browser windows, each displaying a different section of the Flask-SSE documentation. The top window shows the 'Advanced Usage' page, the middle window shows the 'Access Control' page, and the bottom window shows another 'Advanced Usage' page. Each window contains code snippets, explanatory text, and a warning message about the placement of access control handlers.

Top Window: Advanced Usage

```
publish("user": "alice", "status": "Life is good!", channel="users.social")  
sse.publish("active_users": 100), channel="analytics")
```

And this event will be sent to the "analytics" channel:

```
url_for("sse.stream", channel="users.social")
```

Channel names can be any string you want, and are created dynamically as soon as they are referenced. The default channel name that Flask-SSE uses is "sse". For more information, see the documentation for the Redis pubsub system.

To subscribe to a channel, the client only needs to provide a channel query parameter when connecting to the event stream. For example, if your event stream is at /stream, you can connect to the "users.social" channel by using the URL /stream?channel=users.social. You can also use Flask's url_for() function to generate this query parameter, like so:

```
url_for("sse.stream", channel="users.social")
```

By default, all channels are publicly accessible to all users. However, see the next section to change that.

Middle Window: Access Control

Since Flask-SSE is implemented as a blueprint, you can attach a `before_request()` handler to implement access control. For example:

```
@sse.before_request  
def check_access():  
    if request.args.get("channel") == "analytics" and not g.user.is_admin():  
        abort(403)  
  
app.register_blueprint(sse, url_prefix="/sse")
```

Bottom Window: Advanced Usage

```
publish("user": "alice", "status": "Life is good!", channel="users.social")  
sse.publish("active_users": 100), channel="analytics")
```

And this event will be sent to the "analytics" channel:

```
url_for("sse.stream", channel="users.social")
```

Channel names can be any string you want, and are created dynamically as soon as they are referenced. The default channel name that Flask-SSE uses is "sse". For more information, see the documentation for the Redis pubsub system.

To subscribe to a channel, the client only needs to provide a channel query parameter when connecting to the event stream. For example, if your event stream is at /stream, you can connect to the "users.social" channel by using the URL /stream?channel=users.social. You can also use Flask's url_for() function to generate this query parameter, like so:

```
url_for("sse.stream", channel="users.social")
```

By default, all channels are publicly accessible to all users. However, see the next section to change that.

Bottom Bottom Window: Access Control

Since Flask-SSE is implemented as a blueprint, you can attach a `before_request()` handler to implement access control. For example:

```
@sse.before_request  
def check_access():  
    if request.args.get("channel") == "analytics" and not g.user.is_admin():  
        abort(403)  
  
app.register_blueprint(sse, url_prefix="/sse")
```

Warning:

When defining a `before_request()` handler, the blueprint must be registered after the handler is defined! Otherwise, the handler will have no effect.

© Copyright 2016, David Beazley. Created using Sphinx.

I'm 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)

The image shows two screenshots of a web browser displaying the Flask-SSE documentation. Both screenshots are identical and show the 'Advanced Usage' page at <https://flask-sse.readthedocs.io/en/latest/advanced.html>. The browser tabs include 'Server-sent events - VI', 'Using server-sent events', 'Advanced Usage - Fl...', 'Module Applications', 'Gunicorn - Python WSG...', 'what is gevent - gen...', and 'Flask-SSE Quickstart'. The main content on the page discusses publishing events to channels and includes code snippets for defining a 'before_request' handler to check if a user has the 'analytics' role before publishing to the 'analytics' channel. A warning note states that the 'before_request' handler must be registered after the blueprint is defined. The bottom of the page includes a copyright notice: '© Copyright 2016, David Beazley. Created using Sphinx.'

```
def before_request():
    def check_access():
        if request.args.get('channel') == "analytics" and not g.user.is_admin():
            abort(403)

    app.register_blueprint(sse, url_prefix='/sse')
```

```
def before_request():
    def check_access():
        if request.args.get('channel') == "analytics" and not g.user.is_admin():
            abort(403)

    app.register_blueprint(sse, url_prefix='/sse')
```



A screenshot of a web browser showing a Flask-SSE documentation page. The page contains code examples for publishing events to a channel and subscribing to a channel using a URL query parameter. It also includes a section on access control with a warning about the registration of blueprints.

```
sse.publish({"active_users": 100}, channel="analytics")  
  
Channel names can be any string you want, and are created dynamically as soon as they are referenced. The default channel name that Flask-SSE uses is "sse". For more information, see the documentation for the Redis pubsub system.  
  
To subscribe to a channel, the client only needs to provide a channel query parameter when connecting to the event stream. For example, if your event stream is at /stream, you can connect to the "users.social" channel by using the URL /stream?channel=users.social. You can also use Flask's url_for() function to generate this query parameter, like so:  
  
url_for('sse.stream', channel='users.social')  
  
By default, all channels are publicly accessible to all users. However, see the next section to change that.  
  
Access Control  
Since Flask-SSE is implemented as a blueprint, you can attach a before_request() handler to implement access control. For example:  
  
app.before_request  
def check_access():  
    if request.args.get('channel') == 'analytics' and not g.user.is_authenticated:  
        abort(403)  
  
app.register_blueprint(sse, url_prefix='/sse')  
  
Warning:  
When defining a before_request() handler, the blueprint must be registered after the handler is defined! Otherwise, the handler will have no effect.  
  
© Copyright 2016, David Beazley. Created using Sphinx.
```

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.