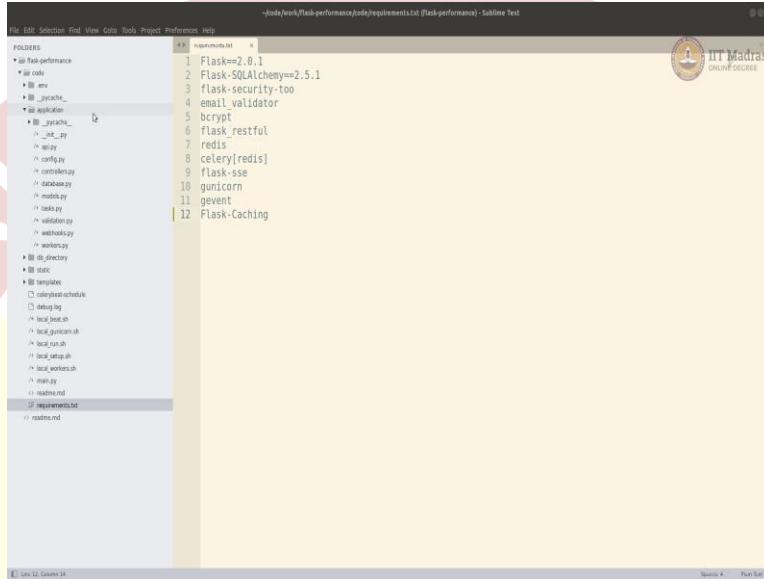


# IIT Madras

## ONLINE DEGREE

**Modern Application Development-II**  
**Software Consultant - Thejesh G N**  
**Bachelor of Science in Programming and Data Science**  
**Indian Institute of Technology, Madras**  
**Performance Measurement and Caching**

(Refer Slide Time: 00:16)



The screenshot shows a Sublime Text editor window with a file named "requirements.txt" open. The file contains the following dependencies:

```
Flask==2.0.1
Flask-SQLAlchemy==2.5.1
Flask-security-too
email_validator
bcrypt
flask_restful
redis
celery[redis]
flask-sse
gunicorn
gevent
Flask-Caching
```

Welcome to the Modern Application Development Two Screencast. In this screencast, we will see how to measure and improve performance. For this, you will need a Linux environment, you will also need Redis. So, please install and run Redis. Now, performance measurement and improvement is a vast subject.

There are many areas of application in which a bottleneck might appear. But we will see some of the common scenarios. And we will also see how to mitigate them or improve the performance in such scenarios. Now, before we start, let us look at the application that we have. We have an application with various files inside the application folder.

(Refer Slide Time: 01:04)

-code/work/flask-performance/code/main.py (Flask-Performance) - Sublime Text

File Edit Select Find View Goto Tools Project Preferences Help

FOLDERS

- **flask-performance**
- **\_\_init\_\_.py**
- **app** code
  - **\_\_init\_\_.py**
  - **code**
  - **db** directory
    - **\_\_init\_\_.py**
    - **alembic**
    - **migrations**
    - **models.py**
    - **scripts.py**
    - **tasks.py**
    - **utils.py**
    - **wrappers.py**
    - **workers.py**
  - **config.py**
  - **controllers.py**
  - **database.py**
  - **middlewares.py**
  - **models.py**
  - **tasks.py**
  - **utils.py**
  - **wrappers.py**
  - **workers.py**
- **db** directory
- **static**
- **templates**
  - **colorprint.sch**
- **celerybeat.sch**
- **debug.log**
- **local.debug**
- **local.development**
- **local.prod**
- **local.setup**
- **local.workshop**

- **main.py**
- **readme.md**
- **requirements.txt**
- **readme.rnd**

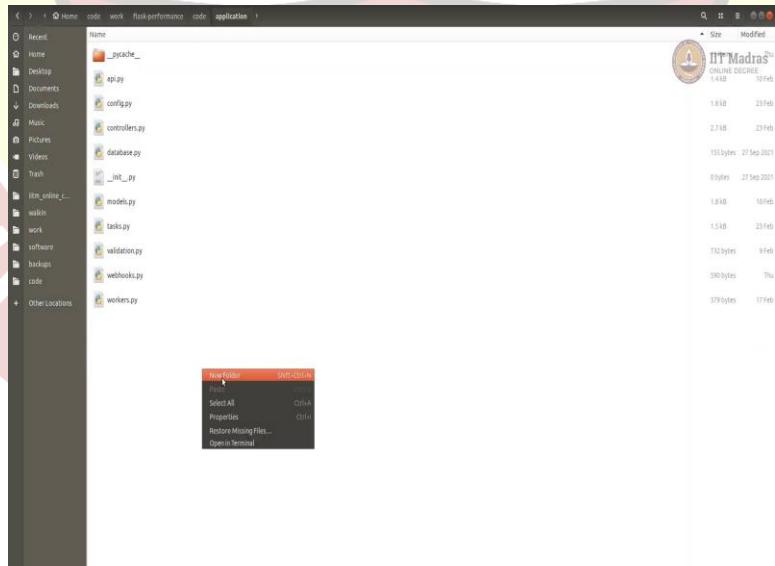
git: 1. L. Colmena | Python

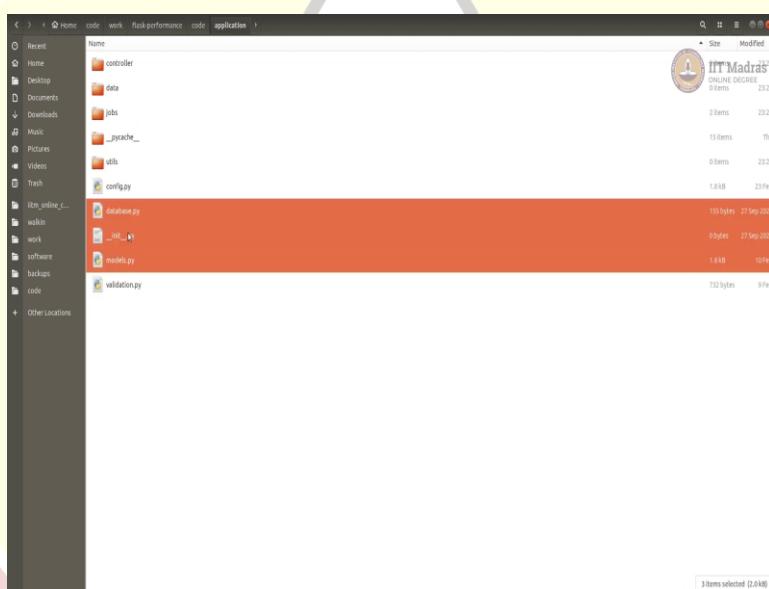
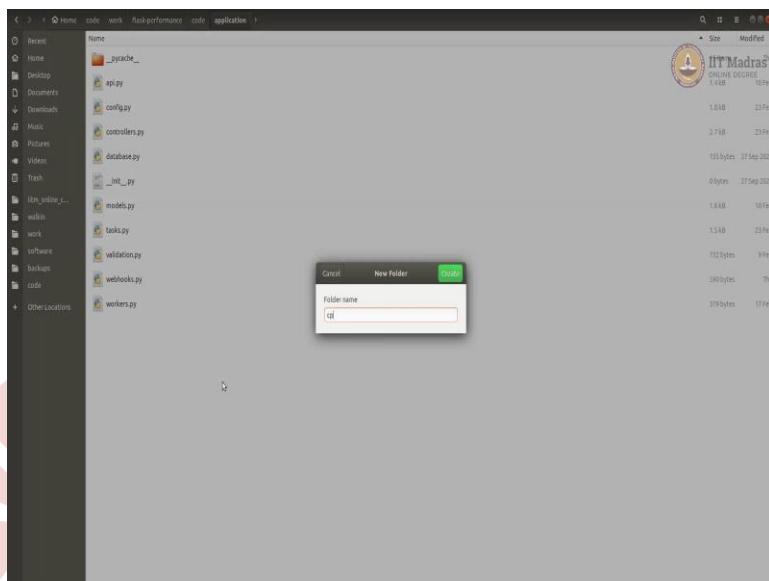
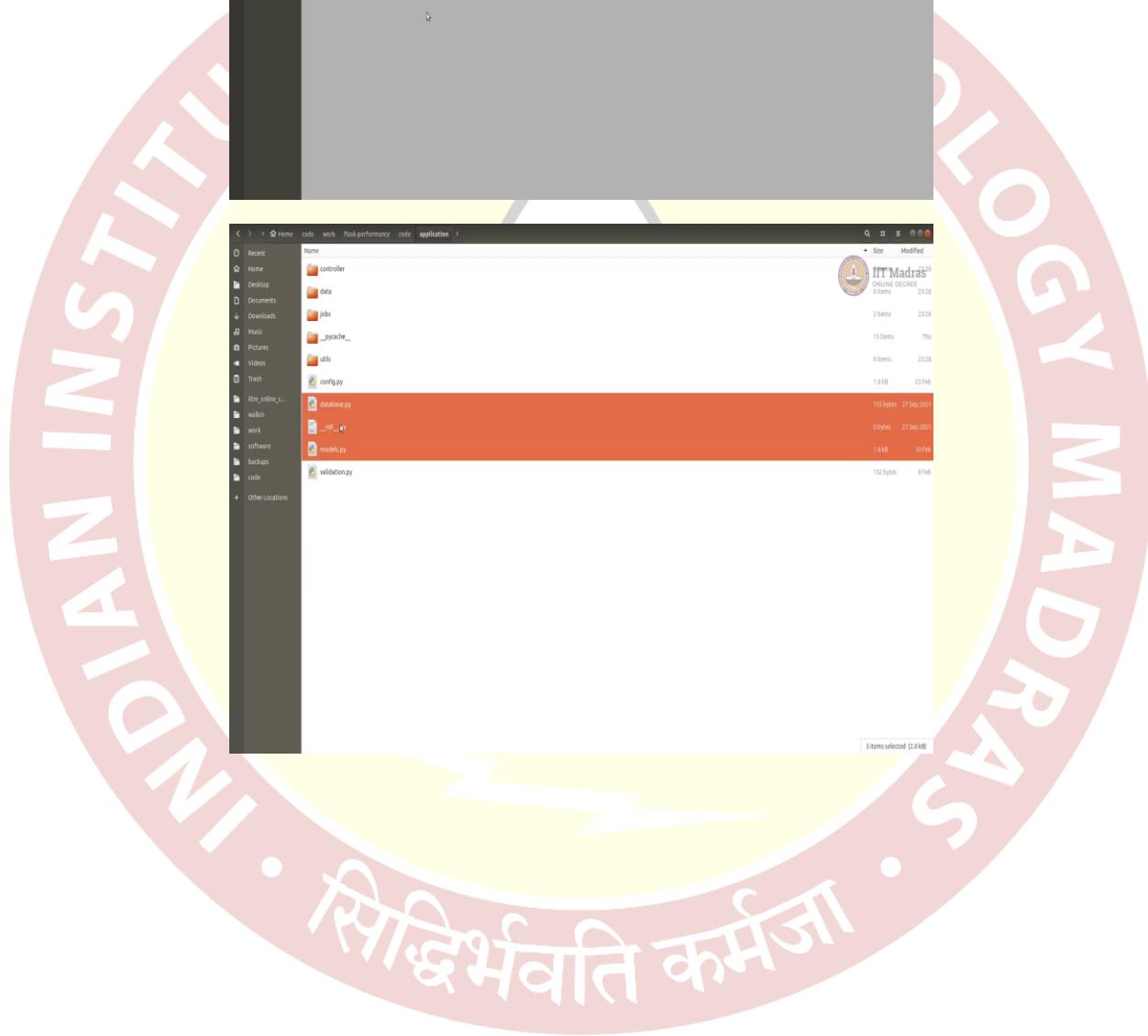
IIT Madras  
ONLINE COLLEGE

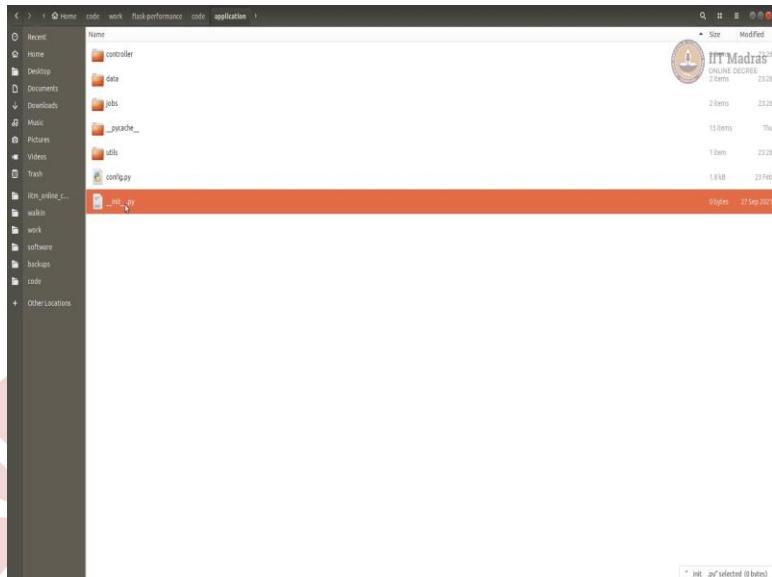
```
1 import os
2 from flask import Flask, render_template
3 from flask_restful import Resource, Api
4 from application import config
5 from application.config import LocalDevelopmentConfig, StageConfig
6 from application.database import db
7 from application import workers
8 from sqlalchemy.orm import scoped_session, sessionmaker
9 from flask_security import Security, SQLAlchemySessionUserDatastore, SQLAlchemyUserDat
10 from application.models import User, Role
11 from flask_login import LoginManager
12 from flask_security import utils
13 from flask_sse import sse
14
15 # import logging
16 # logging.basicConfig(filename='debug.log', level=logging.DEBUG, format=f'%(asctime)s %(leveln
17
18
19 app = None
20 api = None
21 celery = None
22
23 def create_app():
24     app = Flask(__name__, template_folder="templates")
25     print(os.getenv('ENV', "development"))
26     if os.getenv('ENV', "development") == "production":
27         app.logger.info("Currently no production config is setup.")
28         raise Exception("Currently no production config is setup.")
29     elif os.getenv('ENV', "development") == "stage":
30         app.logger.info("Starting stage.")
31         print("Starting stage")
32         app.config.from_object(StageConfig)
33         print("pushed config")
34     else:
35         app.logger.info("Starting Local Development.")
36         print("Starting Local Development")
```

And our main app is in the main. Now before we continue, I want to organize it a little bit better into different modules. So that, going through it becomes easier and managing it becomes easier. Now you can skip it forward if you do not watch it. But it is quicker, just go ahead and do it in front of you.

(Refer Slide Time: 01:27)

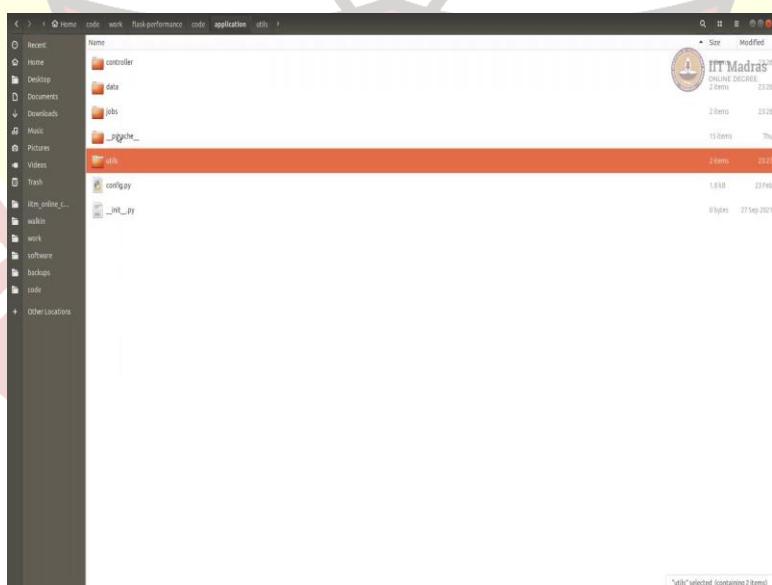






When in the application folder, I am going to create 4 new folders is called controller, and then a folder called data. A folder called a jobs and the folder called utils. Now I am going to move API controller and web books into a controller and workers tasks into Jobs folder, and database and models into data folder and validation into utils folder. Now since they are all modules or packages, I need to add a underscore init file.

(Refer Slide Time: 02:28)



-/code/work/flask-performance/code/application/\_init\_.py (flask-performance) - Sublime Text

File Edit Selection Find View Goto Tools Project Preferences Help

FOOLDERS

- iit-flask-performance
- ii code
- ii env
- ii \_pycache\_
- ii application
- ii \_pycache\_
- ii controller
- ii data
- ii jobs
- ii utils
- ii \_\_init\_\_.py
- ii config.py
- ii db\_directory
- ii static
- ii templates
- cronjob-schedule
- debug.log
- ✓ local\_beat.sh
- ✓ local\_gunicorn.sh
- ✓ local\_rabbit.sh
- ✓ local\_setup.sh
- ✓ local\_workers.sh
- ✓ main.py
- ✓ README.md
- ✓ requirements.txt
- ✓ README.md

1

IIT Madras  
ONLINE DEGREE

Line 1 Colons 1

Span: 4 Python

So, I am going to just do that, copy this, just paste here, paste here, paste here, then paste here, kind of almost done. So, we just need to change the package path, this is just to show that all the controllers without its API, or generic web controllers, or web controllers are all in the control of folder, or module. And all the database related things are in data, all the jobs related things are in the jobs, and all other utils if you have any more, you can put in a utils, makes it a little more cleaner.

(Refer Slide Time: 03:19)

-/code/work/flask-performance/code/application/controller/controllers.py (flask-performance) - Sublime Text

File Edit Selection Find View Goto Tools Project Preferences Help

FOOLDERS

- iit-flask-performance
- ii code
- ii env
- ii \_pycache\_
- ii application
- ii \_pycache\_
- ii controller
- ii data
- ii jobs
- ii utils
- ii \_\_init\_\_.py
- ii config.py
- ii db\_directory
- ii static
- ii templates
- cronjob-schedule
- debug.log
- ✓ local\_beat.sh
- ✓ local\_gunicorn.sh
- ✓ local\_rabbit.sh
- ✓ local\_setup.sh
- ✓ local\_workers.sh
- ✓ main.py
- ✓ README.md
- ✓ requirements.txt
- ✓ README.md

```

1 from flask import Flask, request
2 from flask import render_template
3 from main import app as app
4 from application.datamodels import Article
5 from flask_security import login_required, roles_accepted, roles_required
6 from application import tasks
7 from flask_sse import sse
8
9 print("in controller app", app)
10 @app.route("/hello", methods=["GET"])
11 def hello():
12     return "World"
13
14
15 @app.route("/", methods=["GET", "POST"])
16 def articles():
17     app.logger.info("Inside get all articles using info")
18     articles = Article.query.all()
19     app.logger.debug("Inside get all articles using debug")
20     return render_template("articles.html", articles=articles)
21
22 @app.route("/articles_by<user_name>", methods=["GET", "POST"])
23 @login_required
24 def articles_by_author(user_name):
25     articles = Article.query.filter(Article.authors.any(username=user_name))
26     return render_template("articles_by_author.html", articles=articles, user_name=user_name)
27
28
29 @app.route("/article_like<article_id>", methods=["GET", "POST"])
30 def like(article_id):
31     print(article_id)
32     job_id = tasks.calculate_aggregate_likes.delay(article_id)
33     print(" Job started with job_id = {}".format(job_id))
34     return "OK", 200
35
36
37 @app.route("/feedback", methods=["GET", "POST"])

```

IIT Madras  
ONLINE DEGREE

Line 1 Colons 22

Span: 4 Python



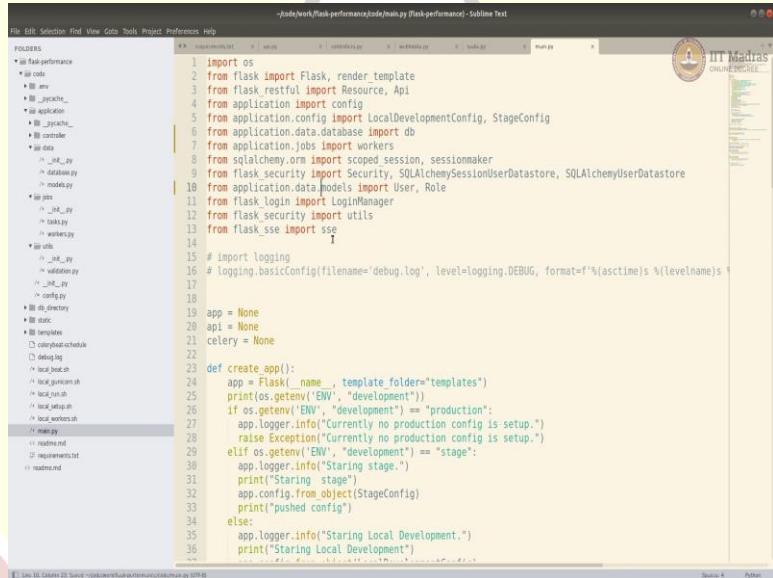
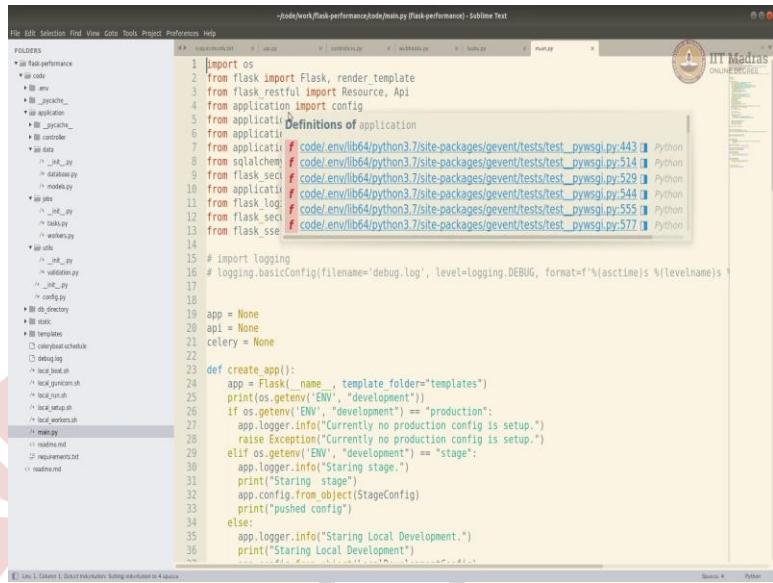
```
->code/work/task-performance/code/application/controller/webhooks.py - [Task-performance] - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
PROJECTS
* task-performance
  * __code
    * __env
    * __pycache__
    * __application
      * __celery
        * __init__.py
        * celery.py
        * controllers.py
        * webhooks.py
      * __data
      * __jobs
      * __logs
      * __init__.py
      * config.py
      * __static
      * __templates
        * celerybeat-schedule
        * debug.log
        * __init__.py
        * local_gunicorn.sh
        * local_psutil.sh
        * local_uptime.sh
        * local_workers.sh
        * main.py
        * README.md
        * requirements.txt
        * README.md

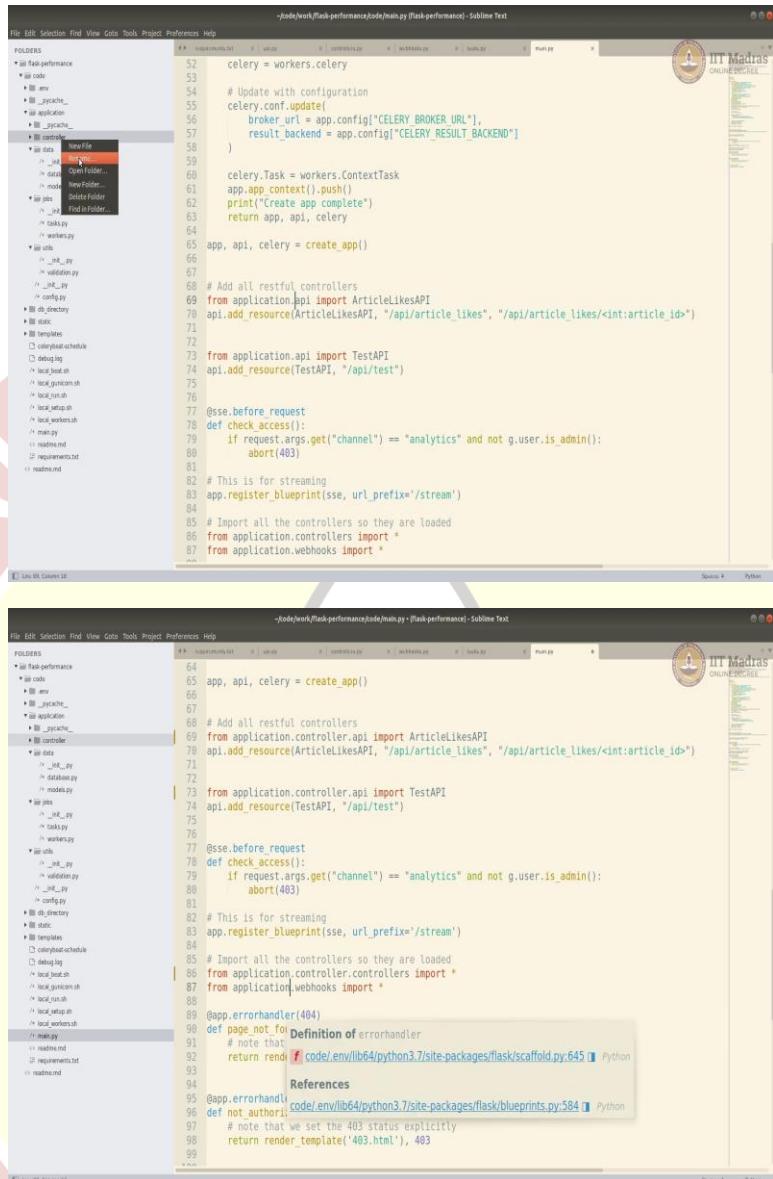
1 from flask import Flask, request
2 from flask import render_template
3 from main import app
4 from application.models import Article
5 from flask_security import login_required, roles_accepted, roles_required
6 from application.job import tasks
7 from flask_sse import running_jobs
8 print("in cont")
9 send_job_offset
10 on_job_process_down
11 @app.route('/ws/', methods=['POST'])
12 def webhook_get(f):
13     _cleanup_after_job_finish
14     get_head
15     parse_th_no_details
16     content = request.json
17     print(content)
18     validate
19     print("-----")
20     print(request.headers)
21     call_async_job
22
23
24 return "OK", 200
25
```

```
->code/work/task-performance/code/application/jobs/workers.py - [Task-performance] - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
PROJECTS
* task-performance
  * __code
    * __env
    * __pycache__
    * __application
      * __celery
        * __init__.py
        * celery.py
        * controllers.py
        * __data
        * __jobs
        * __logs
        * __init__.py
        * tasks.py
        * workers.py
        * __utils
          * __init__.py
          * validation.py
        * __init__.py
        * config.py
        * __static
        * __templates
        * celerybeat-schedule
        * debug.log
        * local_beat.sh
        * local_gunicorn.sh
        * local_psutil.sh
        * local_uptime.sh
        * local_workers.sh
        * main.py
        * README.md
        * requirements.txt
        * README.md

1 from celery import Celery
2 from flask import current_app as app
3
4 celery = Celery("Application Jobs")
5
6
7 # Create a subclass of the task
8 # that wraps the task execution in an application context.
9 # So its available
10
11 class ContextTask(celery.Task):
12     def __call__(self, *args, **kwargs):
13         with app.app_context():
14             return self.run(*args, **kwargs)
15
```







I will, just quickly change the import so that they work this application dot comes from utils. And this comes from data. This comes from data as well, anything else, nothing else here. And here main dot app here, this comes from data and application dot jobs, import task. We are books same here application dot jobs. Your data, rest seems okay. Can do similarly, refactoring in other things to just to make it clean and fast?

Cleaner rather than faster because I think if it is cleaner, it becomes easier to debug and improve it upon in the future. Here it is jobs, utils seems to be okay. We just have one more file left which is the main dot py. This is correct. This is correct. This will be data. Here are some of the imports. That is it, I think we have reorganized, it is pretty much clean.

(Refer Slide Time: 05:43)

```
the@juna:~/code/work/flask-performance$ cd code/flask-performance
the@juna:~/code/work/flask-performance$ cd codes
the@juna:~/code/work/flask-performance/codes$ ls
*application local_beat.sh local_workers.sh requirements.txt
celerybeat-schedule local_unicorn.sh main.py static
db_directory local_run.sh __pycache__ templates
debug.log local_setup.sh README.md
the@juna:~/code/work/flask-performance/codes$ ls -a
.* __pycache__ local_beat.sh main.py static
*application local_unicorn.sh __pycache__ templates
celerybeat-schedule local_run.sh README.md
db_directory local_setup.sh README.md
the@juna:~/code/work/flask-performance/codes$ sh local_run.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

the@juna:~/code/work/flask-performance/codes$ source venv/bin/activate
(venv) the@juna:~/code/work/flask-performance/codes$ cd ..
(venv) the@juna:~/code/work/flask-performance$ cd ..
(venv) the@juna:~/code/work$ cd flask-performance
(venv) the@juna:~/code/work$ cd main.py
(venv) the@juna:~/code/work$ nano main.py

# Local Development
# pushed config
# DB Init
# DB Init complete
# Create app complete
# Developing Local Development
# pushed config
# DB Init
# DB Init complete
# Create app complete
# in controller app <Flask 'main'>
# in controller app <Flask 'main'>
#   * Serving Flask app 'main' (lazy loading)
#   * Environment: production
#     WARNING: This is a development server. Do not use it in a production environment!
#     Use a production WSGI server instead.
#     * Debug mode: on
#     * Running on all addresses.
#     WARNING: This is a development server. Do not use it in a production deployment!
# nt.
# * Restarting with stat
# [pid=1] [detached]
# celerybeat-schedule
# debug.log
# local_beat.sh
# local_unicorn.sh
# local_run.sh
# local_setup.sh
# requirements.txt
# README.md
# main.py
# pushed config
# DB Init
# DB Init complete
# Create app complete
# Developing Local Development
# pushed config
# DB Init
# DB Init complete
# Create app complete
# in controller app <Flask 'main'>
# in controller app <Flask 'main'>
#   * Serving Flask app 'main' (lazy loading)
#   * Environment: production
#     WARNING: This is a development server. Do not use it in a production environment!
#     Use a production WSGI server instead.
#     * Debug mode: on
#     * Running on all addresses.
#     WARNING: This is a development server. Do not use it in a production deployment!
# nt.
# * Restarting with stat
# [pid=1] [detached]
```



Application for the next batch is now open | Up to 100% scholarships for female students available [APPLY NOW](#)

**PROGRAMMING & DATA SCIENCE**

IIT Madras, India's top technical institute, welcomes you to the world's first BSc Degree program in Programming and Data Science.

For the first time, you can work towards an undergraduate degree / diploma from an IIT regardless of your age or location, and with a wide range of academic backgrounds.

**DEGREE PROGRAM**

**BSc in Programming and Data Science**  
with option to exit earlier with Foundation Certificate or Diploma

Program covers all topics thoroughly starting from the foundations. Anyone who has completed class 12 can apply.

[APPLY NOW](#)

**EXCLUSIVE DIPLOMA PROGRAM**

**Diploma in Programming / Diploma in Data Science**

For applicants with knowledge of the foundations and requirement to complete class 12. Anyone who has completed UG-degree or at least two years of a UG-degree can apply.

[GO TO DIPLOMA WEBSITE](#)

All Articles - localhost:8080

localhost:8080 - Google Search

localhost:8080 - localhost:8080/ela

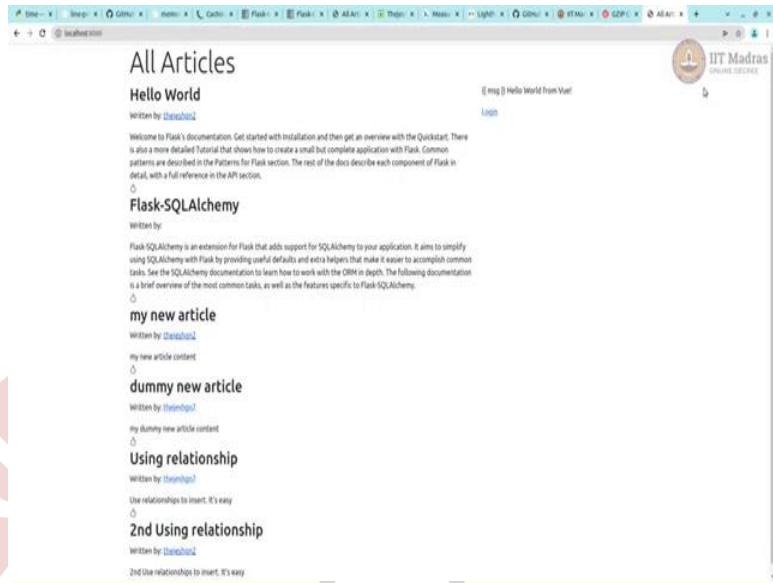
All Articles - localhost:8080/articles\_by\_thepeshg2

localhost:8080/login/restart-n2z/articles\_jyq2Jfhepgj2

Search Google or type a URL

All Articles Thesaurus Projects Data IIT Madras Wikipedia, Inc. Nodus Code... Amazon.in localhost GitHub Add shortcut

Customize Chrome



So, let us start it have already set up the environment. You can see here, consider dot enb, if you want to consider type it again. So, I will just run it just to check whether all the changes are okay, it seems to be okay, let us just open our page and see if it loads. So the restructuring is fine.

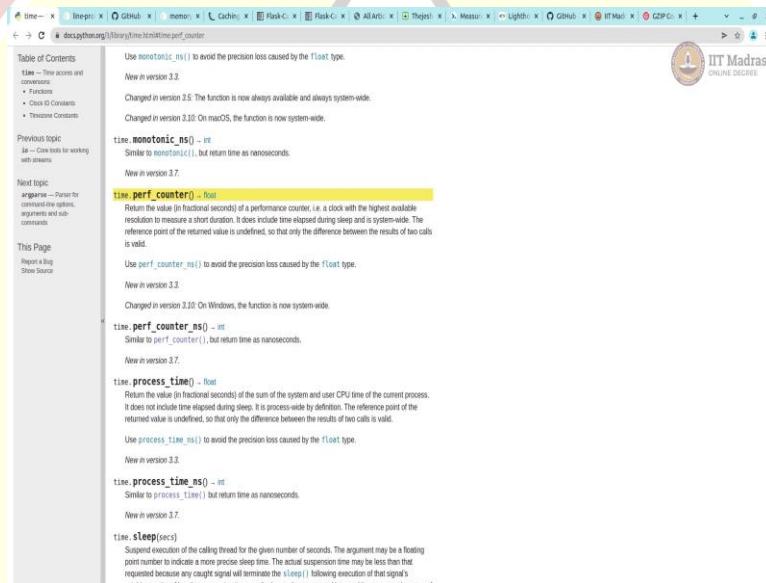
(Refer Slide Time: 06:18)

Now, we want to see how we can improve this application from two points of view. One is from the front end point of view and another one is the back end point of view. Back in, there are two parts, mostly the processing that we are going to do and the queries that we are going to run. For example, let us take a controller, here is a controller, there are three things that are happening here, one we are getting, when the user accesses slash, it is going to come to article.

And it is going to do some processing if there are loops or anything like that, it is going to do a query, then it is going to render the template and then return to the user. Now, if the articles are not going to change very often, we do not need to run this for every user all the time. Or, for example, we do not need to run at least this, if you want to change some rendering, for example, if they are rendering as username, that you do not want to do, then we do not we can want to render this, but we do not want to repeat this.

So, you can actually, reduce the process at various levels. So, you first we will start with measuring, and then we will see how to reduce it. Let us start measuring.

(Refer Slide Time: 07:44)



The image shows two screenshots of a Sublime Text editor window. Both screenshots display the same Python code for a Flask application controller. The code includes imports for flask, flask\_see, time, and application.models. It defines several routes: a simple hello world route, a route to get all articles, a route to get articles by user, a route to like an article, and a feedback route. The code uses the perf\_counter\_ns function from time to measure the execution time of the article retrieval logic.

```

1 from flask import Flask, request
2 from flask import render_template
3 from main import app
4 from application.data.models import Article
5 from flask_security import login_required, roles_accepted, roles_required
6 from application.jobs import tasks
7 from flask_see import see
8 from time import perf_counter
9 from time import perf_counter_ns
10
11 print("in controller app", app)
12
13 @app.route("/hello", methods=["GET"])
14 def hello():
15     return "World"
16
17 @app.route("/", methods=["GET", "POST"])
18 def articles():
19     app.logger.info("Inside get all articles using info")
20     articles = Article.query.all()
21     app.logger.debug("Inside get all articles using debug")
22     return render_template("articles.html", articles=articles)
23
24 @app.route("/articles_by<user_name>", methods=["GET", "POST"])
25 @login_required
26 def articles_by_author(user_name):
27     articles = Article.query.filter(Article.authors.any(username=user_name))
28     return render_template("articles_by_author.html", articles=articles, user_name=user_name)
29
30
31 @app.route("/article_like<article_id>", methods=["GET", "POST"])
32 def like(article_id):
33     print(article_id)
34     job_id = tasks.calculate_aggregate_likes.delay(article_id)
35     print("Job started with job id = {}".format(job_id))
36     return "OK", 200
37

```

```

7 from flask_see import see
8 from time import perf_counter_ns
9
10 print("in controller app", app)
11
12 @app.route("/hello", methods=["GET"])
13 def hello():
14     return "World"
15
16 @app.route("/", methods=["GET", "POST"])
17 def articles():
18     app.logger.info("Inside get all articles using info")
19     start = perf_counter_ns()
20     articles = Article.query.all()
21     stop = perf_counter_ns()
22     print("Time taken", stop - start)
23     app.logger.debug("Inside get all articles using debug")
24     return render_template("articles.html", articles=articles)
25
26 @app.route("/articles_by<user_name>", methods=["GET", "POST"])
27 @login_required
28 def articles_by_author(user_name):
29     articles = Article.query.filter(Article.authors.any(username=user_name))
30     return render_template("articles_by_author.html", articles=articles, user_name=user_name)
31
32
33 @app.route("/article_like<article_id>", methods=["GET", "POST"])
34 def like(article_id):
35     print(article_id)
36     job_id = tasks.calculate_aggregate_likes.delay(article_id)
37     print("Job started with job id = {}".format(job_id))
38     return "OK", 200
39
40
41 @app.route("/feedback", methods=["GET", "POST"])
42 def feedback():
43     if request.method == "GET":
44

```

For measuring, we are going to use a simplest function called perf counter ns, it is a built in python function that we can use. It starts a timer, and then you can start or stop the timer whenever you want and measure the difference. Let us start doing that. Now I am just going to add here, I am just going to use the nanosecond one and not the second one.

So, only this is enough. So, let us say we want to start, we want to improve this, let us assume that this is the one which is taking a lot of time, we want to improve this. So, let us add a start pointer, and then we will add a stop pointer, stop and then we will print it. Print Time taken stop minus start to the end minus start, let us say how much time it takes.

Here we do not have many articles. So, it might take a quite less time. But, I assume if you are running a media organization where you are at 10s of 1000s of articles, then this query would be taking a lot of time because it is trying to query all over the whole table. Even if you have, some constraint, it is going to run over the table and then reduce and do all of those.

So, generally, listing is one of the areas that we concentrate when you are improving the performance. So, we will see how much time it takes.

(Refer Slide Time: 09:45)

The screenshot shows a terminal window and a browser window side-by-side.

**Terminal Window:**

```
the@the:~/code/week/Flask-performance$ 
+ Restarting with stat
development
Starting Local Development
pushed config
DB Init
DB Init complete
Create app complete
development
starting Local Development
pushed config
DB Init
DB Init complete
Create app complete
in controller app <flask 'main'>
in controller app <flask 'main'>
+ debugger is active!
+ debugger PIN: 362-250-897
Time taken: 106801437
127.0.0.1 - [02/Mar/2022 23:34:55] "GET / HTTP/1.1" 200 -
127.0.0.1 - [02/Mar/2022 23:34:55] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 200 -
127.0.0.1 - [02/Mar/2022 23:34:55] "GET /static/bootstrap/js/bootstrap.min.js HTTP/1.1" 200 -
127.0.0.1 - [02/Mar/2022 23:34:55] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - [02/Mar/2022 23:34:55] "GET /static/vue.min.js HTTP/1.1" 200 -
127.0.0.1 - [02/Mar/2022 23:34:55] "GET /static/vue.min.js HTTP/1.1" 300 -
127.0.0.1 - [02/Mar/2022 23:34:55] "GET /static/app.js HTTP/1.1" 200 -
Time taken: 2134475
127.0.0.1 - [02/Mar/2022 23:35:03] "GET / HTTP/1.1" 200 -
127.0.0.1 - [02/Mar/2022 23:35:03] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - [02/Mar/2022 23:35:03] "GET /static/bootstrap/js/bootstrap.min.js HTTP/1.1" 304 -
127.0.0.1 - [02/Mar/2022 23:35:03] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - [02/Mar/2022 23:35:03] "GET /static/vue.min.js HTTP/1.1" 304 -
127.0.0.1 - [02/Mar/2022 23:35:03] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 304 -
127.0.0.1 - [02/Mar/2022 23:35:03] "GET /static/app.js HTTP/1.1" 304 -
127.0.0.1 - [02/Mar/2022 23:35:08] "GET /static/vue.min.js HTTP/1.1" 304 -
127.0.0.1 - [02/Mar/2022 23:35:08] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - [02/Mar/2022 23:35:08] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 304 -
127.0.0.1 - [02/Mar/2022 23:35:08] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - [02/Mar/2022 23:35:08] "GET /static/bootstrap/js/bootstrap.min.js HTTP/1.1" 304 -
127.0.0.1 - [02/Mar/2022 23:35:08] "GET /static/app.js HTTP/1.1" 304 -
127.0.0.1 - [02/Mar/2022 23:35:08] "GET /static/vue.min.js HTTP/1.1" 304 -
```

**Browser Window:**

The browser displays the Python documentation for the `time` module. The specific section shown is `time.monotonic_ns()`. The page includes the IIT Madras logo and navigation links like 'This Page', 'Report a Bug', and 'Show Source'.

**Content of time.monotonic\_ns() Documentation:**

- Table of Contents**: Includes links to `time`, `time.access`, `time.constants`, `time.futex`, `time.ID_Conditions`, `time.ID_Conds`, `time.sleep`, `time.time`, `time.ticks`, and `time.ticks_per_second`.
- Changes in Version 3.3**: `time.monotonic_ns()` now avoids precision loss caused by the `float` type.
- Changes in Version 3.0**: The function is now always available and system-wide.
- Changes in Version 3.0**: On macOS, the function is now system-wide.
- time.perf\_counter\_ns()**: Returns the value (in nanoseconds) of a performance counter, i.e. a clock with the highest available resolution to measure a short duration. It does include time elapsed during sleep and is system-wide. The reference point of the returned value is undefined, so that only the difference between the results of two calls is valid.
- time.perf\_counter()**: Returns the value (in fractional seconds) of a performance counter, i.e. a clock with the highest available resolution to measure a short duration. It does include time elapsed during sleep and is system-wide. The reference point of the returned value is undefined, so that only the difference between the results of two calls is valid.
- time.sleep()**: Suspends execution of the calling thread for the given number of seconds. The argument may be a floating-point number to indicate a more precise sleep time. The actual suspension time may be less than that requested because any caught signal will terminate the `sleep()` following execution of that signal's



The screenshot shows a web browser displaying the Flask-Caching documentation at <https://flask-caching.readthedocs.io/en/latest/>. The page has a sidebar with a table of contents and a main content area with sections like 'Version support', 'Installation', and 'Set Up'. It includes code snippets for configuration and usage.

```

    #!/usr/bin/env python
    from flask import Flask
    from flask_caching import Cache

    config = {
        'DEBUG': True,                      # some Flask specific configs
        'CACHE_TYPE': 'SimpleCache',        # Flask-Caching related configs
        'CACHE_DEFAULT_TIMEOUT': 300
    }
    app = Flask(__name__)
    # tell Flask to use the above defined config
    app.config.from_mapping(config)
    cache = Cache(app)

    cache = Cache(config={'CACHE_TYPE': 'SimpleCache'})
    app = Flask(__name__)
    cache.init_app(app)

```

Let us stop this starting in, and let us refresh this and not close the page. So, here it is, so it is taking, some nanoseconds. And if I refresh, it is roughly changing. But it is taking somewhat similar time, it changes sometimes based on, whether the code is already compiled, it is in the memory of the based on that. But it is taking, what we call significant amount of time.

Because we know that it is entering here, running this query, and all of this. What if, what if the articles do not change often? Let us say all our editors upload the article in the morning, and we do not change articles during the day, and only the evening, they will upload again. So, we could literally cache this for a whole day without querying the database, every time the user accesses.

So, if in case, if this whole thing came from memory, it would have been really fast instead of hitting a database. Now, here, we are using a SQLite. So, it is on the same machine. And it is relatively fast, because I am using an SSD and also I have data in that, very less data compared to any media organization. But assume a case where you are hitting a Postgres database on a different machine.

And there is a network called network query, getting the data and doing the stuff, then actually optimizing this becomes very important. Let us see how to optimize this if you want to optimize this. Basically, I do not want to get this every time, I want to get it only when required, or cache it for time that I do not want to query database all the time. To cache this, we are going to use a library called flask caching it is one of the recommended ones by the flask project itself, you can go check out. Check the project, can install it by pip installing Flask-caching.

(Refer Slide Time: 12:08)

The image shows two screenshots of a code editor window from Sublime Text. The top screenshot displays the file `controllers.py` with Python code for a Flask application. The bottom screenshot displays the file `requirements.txt` with a list of Python package dependencies.

**controllers.py (Top Screenshot):**

```
from flask_sse import sse
from time import perf_counter_ns
print("in controller app", app)
@app.route("/hello", methods=["GET"])
def hello():
    return "World"
@app.route("/", methods=["GET", "POST"])
def articles():
    app.logger.info("Inside get all articles using info")
    start = perf_counter_ns()
    articles = Article.query.all()
    stop = perf_counter_ns()
    print("Time taken", stop - start)
    app.logger.debug("Inside get all articles using debug")
    return render_template("articles.html", articles=articles)
@app.route("/articles_by<user_name>", methods=["GET", "POST"])
@login_required
def articles_by_author(user_name):
    articles = Article.query.filter(Article.authors.any(username=user_name))
    return render_template("articles_by_author.html", articles=articles, user_name=user_name)
@app.route("/article_like<article_id>", methods=["GET", "POST"])
def like(article_id):
    print(article_id)
    job_id = tasks.calculate_aggregate_likes.delay(article_id)
    print("Job started with job_id = {}".format(job_id))
    return "OK", 200
@app.route("/feedback", methods=["GET", "POST"])
def feedback():
    pass
```

**requirements.txt (Bottom Screenshot):**

```
Flask==2.0.1
Flask-SQLAlchemy==2.5.1
flask-security-too
email_validator
bcrypt
flask_restful
redis
celery[redis]
flask-sse
gunicorn
gevent
Flask-Caching
```

The screenshot shows two windows. The top window is a Sublime Text editor displaying a Python configuration file named 'config.py'. The code defines several classes: 'Config', 'LocalDevelopmentConfig', and 'StageConfig'. It includes settings for database paths ('SQLITE\_DB\_DIR'), security ('SECRET\_KEY', 'SECURITY\_PASSWORD\_HASH', 'SECURITY\_PASSWORD\_SALT'), and Celery brokers ('CELERY\_BROKER\_URL', 'CELERY\_RESULT\_BACKEND'). The bottom window is a web browser showing a configuration page for 'Flask-Caching'. The page lists various cache types like 'NullCache', 'SimpleCache', 'FilesystemCache', 'RedisCache', 'RedisClusterCache', 'MemcachedCache', 'SASMemcachedCache', and 'SprinklesASAMemcachedCache'. It also includes configuration options for 'CACHE\_TYPE', 'CACHE\_NO\_NULL\_WARNING', 'CACHE\_ARGS', 'CACHE\_OPTIONS', 'CACHE\_DEFAULT\_TIMEOUT', 'CACHE\_IGNORE\_ERRORS', 'CACHE\_THRESHOLD', and 'CACHE\_KEY\_PREFIX'.

13:30

I have already added to requirements dot txt here. And I have already installed it, so I do not need to do any further installation, go ahead and install on your local machine. Once you install, we need to do some setup, flask cache in, supports multiple types of caching, you can make it null, which means no caching. And there is simple caching, Redis caching in and various kinds of caching.

We could use local file system caching in the sense you read from the database, write it to local file and read it from there and display to the user every time you want to. Or you could use Redis

because very since Redis is a fast in-memory key value database, it could be much much faster than actually calling the database.

Similarly, you can use memcached. In this case, we are going to use a Redis Cache, because we already have the Redis running and we are going to use the same Redis of for many things it is some multipurpose tool. So, we are going to set up cache type here. And since you are going to use Redis, you are going to set it up Redis cache in the config.

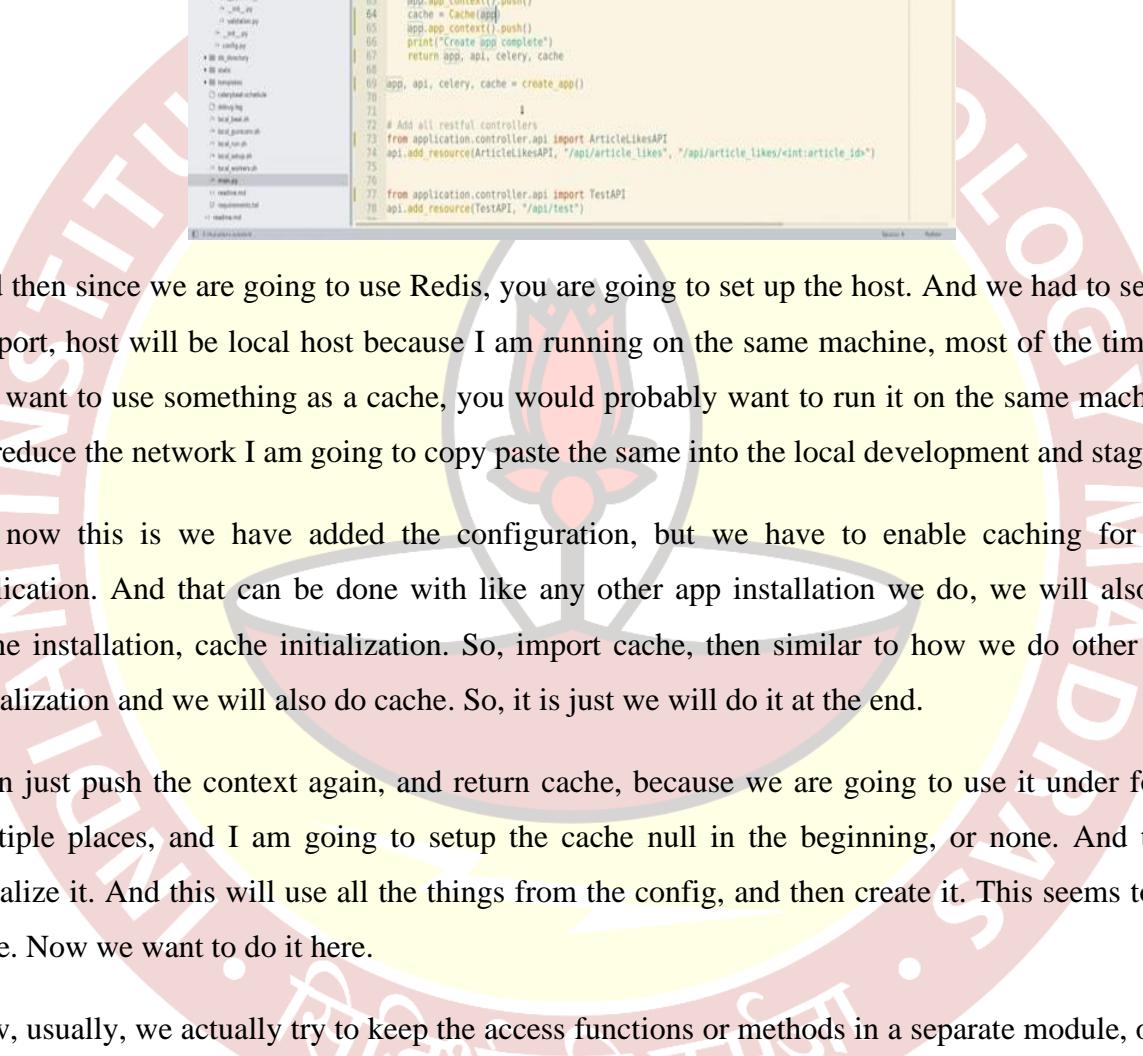
(Refer Slide Time: 13:31)

The screenshot shows two windows from the Sublime Text editor. The top window displays the file `application/config.py` with the following content:

```
-/code/work/task-performance/code/application/config.py (task-performance) - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
...
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    REDIS_URL = "redis://localhost:6379"
14    CACHE_TYPE = "RedisCache"
15    CACHE_REDIS_HOST = "localhost"
16    CACHE_REDIS_PORT = 6379
17
18    class LocalDevelopmentConfig(Config):
19        SQLITE_DB_DIR = os.path.join(basedir, "../db_directory")
20        SQLALCHEMY_DATABASE_URI = "sqlite:///{}+os.path.join(SOLITE_DB_DIR, "testdb.sqlite3")"
21        DEBUG = True
22        SECRET_KEY = "ash ah secret"
23        SECURITY_PASSWORD_HASH = "bcrypt"
24        SECURITY_PASSWORD_SALT = "really super secret" # Read from ENV in your case
25        SECURITY_REGISTERABLE = True
26        SECURITY_CONFIRMABLE = False
27        SECURITY_SEND_REGISTER_EMAIL = False
28        SECURITY_UNAUTHORIZED_VIEW = None
29        WTF_CSRF_ENABLED = False
30        CELERY_BROKER_URL = "redis://localhost:6379/1"
31        CELERY_RESULT_BACKEND = "redis://localhost:6379/2"
32        REDIS_URL = "redis://localhost:6379"
33
34    class StageConfig(Config):
35        SQLITE_DB_DIR = os.path.join(basedir, "../db_directory")
36        SQLALCHEMY_DATABASE_URI = "sqlite:///{}+os.path.join(SOLITE_DB_DIR, "testdb.sqlite3")"
37
```

The bottom window shows a table of Redis configuration parameters and their descriptions:

Parameter	Description
<code>memcached or memcached</code>	SASLMemcachedCache (pylibmc required; old name is <code>saslmemcached</code> ) SpreadSASLMemcachedCache (pylibmc required; old name is <code>spreadmemcached</code> )
<code>CACHE_NO_NULL_WARNING</code>	Silence the warning message when using cache type of "null".
<code>CACHE_ARGS</code>	Optional list to supply and pass during the cache class instantiation.
<code>CACHE_OPTIONS</code>	Optional dictionary to pass during the cache class instantiation.
<code>CACHE_DEFAULT_TIMEOUT</code>	The default timeout that is used if no timeout is specified. Unit of time is seconds.
<code>CACHE_DELETE_ERRORS</code>	If set to any errors that occurred during the deletion process will be ignored. However, if it is set to <code>False</code> it will log on the first error. This option is only relevant for the backends <code>filesystem</code> and <code>simple</code> . Details in <code>FATCache</code> .
<code>CACHE_THRESHOLD</code>	The maximum number of items the cache will store before it starts deleting items. Only for <code>SimpleCache</code> and <code>FilesystemCache</code> .
<code>CACHE_KEY_PREFIX</code>	A prefix that is added before all keys. This makes it possible to use the same cached service for different apps. Used only for <code>RedisCache</code> and <code>MemcachedCache</code> .
<code>CACHE_SOURCE_CHECK</code>	The default condition applied to function decorators which controls if the source code of the function should be included when forming the hash which is used as the cache key. This ensures that if the source code of the function changes, the item will be deleted when the new function is called even if the arguments are the same. Default is <code>False</code> .
<code>CACHE_UNSET_NAME</code>	The name of the newsgroup caching instance to connect to, for example: <code>newsgagelocation:3333</code> , defaults to an empty string, which means gNSG will cache in the local instance. If the cache is in the same instance as the working app, you only have to provide the name of the cache.
<code>CACHE_MEMCACHED_SERVERS</code>	A list or a tuple of server addresses. Used only for <code>MemcachedCache</code> .
<code>CACHE_MEMCACHED_USERNAME</code>	Username for SASL authentication with memcached. Used only for <code>SASLMemcachedCache</code> .
<code>CACHE_MEMCACHED_PASSWORD</code>	Password for SASL authentication with memcached. Used only for <code>SASLMemcachedCache</code> .
<code>CACHE_REDIS_HOST</code>	A Redis server host. Used only for <code>RedisCache</code> .
<code>CACHE_REDIS_PORT</code>	A Redis server port. Default is 6379. Used only for <code>RedisCache</code> .
<code>CACHE_REDIS_PASSWORD</code>	A Redis password for server. Used only for <code>RedisCache</code> and <code>RedisSentinelCache</code> .
<code>CACHE_REDIS_DB</code>	A Redis db (renumbered number index). Default is 0. Used only for <code>RedisCache</code> and <code>RedisSentinelCache</code> .



```

File Edit Selection Find View Goto Tools Project Preferences Help
File Edit Selection Find View Goto Tools Project Preferences Help
43: db.init_app(app)
44: print("DB Init complete")
45: app.app_context().push()
46: app.logger.info("App setup complete")
47: # Setup Flask-Security
48: user_datastore = SQLAlchemyUserDatasource(db.session, User, Role)
49: security = Security(app, user_datastore)
50: api = Api(app)
51: app.app_context().push()
52:
53: # Create celery
54: celery = workers.celery
55:
56: # Update with configuration
57: celery.conf.update(
58:     broker_url = app.config["CELERY_BROKER_URL"],
59:     result_backend = app.config["CELERY_RESULT_BACKEND"]
60: )
61:
62: celery.Task = workers.ContextTask
63: app.app_context().push()
64: cache = Cache(app)
65: app.app_context().push()
66: print("Create app complete")
67: return app, api, celery, cache
68:
69: app, api, celery, cache = create_app()
70:
71: #
72: # Add all restful controllers
73: from application.controller.api import ArticleLikesAPI
74: api.add_resource(ArticleLikesAPI, "/api/article_likes", "/api/article_likes/<int:article_id>")
75:
76: from application.controller.api import TestAPI
77: api.add_resource(TestAPI, "/api/test")

```

And then since we are going to use Redis, you are going to set up the host. And we had to set up the port, host will be local host because I am running on the same machine, most of the time, if you want to use something as a cache, you would probably want to run it on the same machine. To reduce the network I am going to copy paste the same into the local development and staging.

So, now this is we have added the configuration, but we have to enable caching for our application. And that can be done with like any other app installation we do, we will also do cache installation, cache initialization. So, import cache, then similar to how we do other app initialization and we will also do cache. So, it is just we will do it at the end.

Then just push the context again, and return cache, because we are going to use it under for it multiple places, and I am going to setup the cache null in the beginning, or none. And then initialize it. And this will use all the things from the config, and then create it. This seems to be done. Now we want to do it here.

Now, usually, we actually try to keep the access functions or methods in a separate module, or in a separate file. And so that it can be reused across different calls. For example, I had articles get if I had a function called get articles that can be used here, that can be used in an API that can be used in web books, all of that will use the same function and it can be reused.

And it is very useful, because then you can apply your caching, your authorization or any other things that you want to do at one place. And it applies, regardless from where it is accessed. So,

what we will do is we will create a file called data access dot pie, and I will add some functions to it. So, that it can be used by our controllers.

(Refer Slide Time: 16:31)

First thing, I will add article and article likes, and then I will add a function called get articles. And it is going to do one simple thing return the articles. Similarly, I will add another function get articles by user name. That also we use in one of our controllers, and I return the articles filtered by the username. Now, let us take this get all articles and use it in the controller.

Instead of this I can just get our articles, before that just do from data, data access spelling correct, access correct. Now, we will call this. Similarly instead of this could use and get articles by username and pass the username. So, now this is all done. So, you are start is the end is there data accesses there. Now, we just want to cache this is that can be done very easily here.

So, we want to cache the output of this, whenever this gets called, we want the cached or to be returned. And that can be done very easily by using a cache. So, this says, cache it for at least 50 microseconds or milliseconds. And the cache prefixes get all articles. So, now one of the things that you might have to do is to change your models, because I think since it is in time the lazy dynamic will not work for the article author scating.

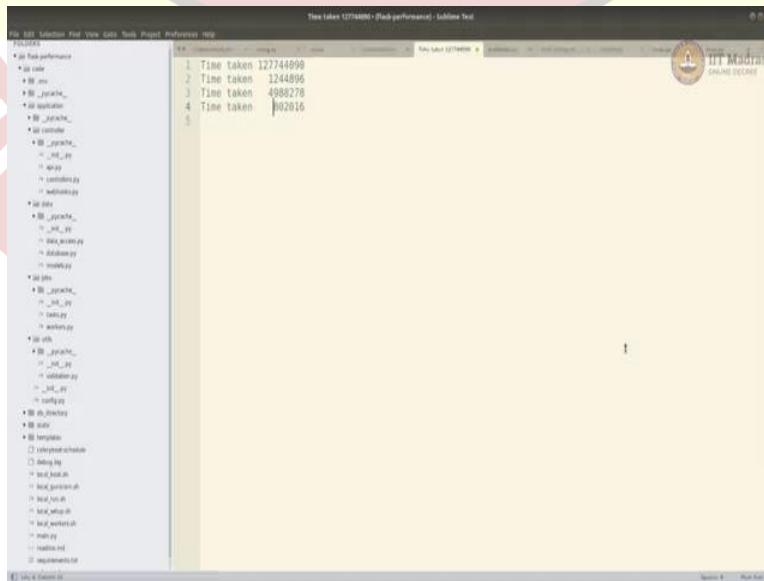
So, you might have to change it to get it using sub-query so that it gets everything and caches it. So, that can be done just by changing lazy is equal to sub-query this. So, now we are ready.

(Refer Slide Time: 19:47)

```

File Edit View Search Terminal Help
Starting Local Development
pushed config
DB init
DB init complete
Create app complete
In controller app <flask 'main'>
In controller app <flask 'main'>
  * Debugger is active
    -> Debugger Port: 0.0.0.0:250-897
Time taken: 127744998
[02/Mar/2022 23:45:01] "GET / HTTP/1.1" 200 -
[02/Mar/2022 23:45:01] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 304 -
[02/Mar/2022 23:45:01] "GET /static/style.css HTTP/1.1" 200 -
[02/Mar/2022 23:45:01] "GET /static/js/custom.js HTTP/1.1" 304 -
[02/Mar/2022 23:45:01] "GET /static/vue/vue.min.js HTTP/1.1" 304 -
[02/Mar/2022 23:45:01] "GET /static/bootstrap/js/bootstrap.min.js HTTP/1.1" 304 -
[02/Mar/2022 23:45:01] "GET /static/js/app.js HTTP/1.1" 304 -
Time taken: 1244896
[02/Mar/2022 23:45:04] "GET / HTTP/1.1" 200 -
[02/Mar/2022 23:45:04] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 304 -
[02/Mar/2022 23:45:04] "GET /static/js/app.js HTTP/1.1" 304 -
[02/Mar/2022 23:45:04] "GET /static/style.css HTTP/1.1" 304 -
[02/Mar/2022 23:45:04] "GET /static/js/custom.js HTTP/1.1" 304 -
[02/Mar/2022 23:45:04] "GET /static/vue/vue.min.js HTTP/1.1" 304 -
[02/Mar/2022 23:45:04] "GET /static/bootstrap/js/bootstrap.min.js HTTP/1.1" 304 -
[02/Mar/2022 23:45:04] "GET /static/js/app.js HTTP/1.1" 304 -
Time taken: 4988278
[02/Mar/2022 23:45:08] "GET / HTTP/1.1" 200 -
[02/Mar/2022 23:45:08] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 304 -
[02/Mar/2022 23:45:08] "GET /static/style.css HTTP/1.1" 304 -
[02/Mar/2022 23:45:08] "GET /static/bootstrap/js/bootstrap.min.js HTTP/1.1" 304 -
[02/Mar/2022 23:45:08] "GET /static/vue/vue.min.js HTTP/1.1" 304 -
[02/Mar/2022 23:45:08] "GET /static/bootstrap/js/bootstrap.min.js HTTP/1.1" 304 -
[02/Mar/2022 23:45:08] "GET /static/js/app.js HTTP/1.1" 304 -
[02/Mar/2022 23:45:08] "GET /static/js/app.js HTTP/1.1" 304 -
Time taken: 892016
[02/Mar/2022 23:45:12] "GET / HTTP/1.1" 200 -
[02/Mar/2022 23:45:12] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 304 -
[02/Mar/2022 23:45:12] "GET /static/bootstrap/js/bootstrap.min.js HTTP/1.1" 304 -
[02/Mar/2022 23:45:12] "GET /static/style.css HTTP/1.1" 200 -
[02/Mar/2022 23:45:12] "GET /static/vue/vue.min.js HTTP/1.1" 304 -
[02/Mar/2022 23:45:12] "GET /static/bootstrap/js/bootstrap.min.js HTTP/1.1" 304 -
[02/Mar/2022 23:45:12] "GET /static/js/app.js HTTP/1.1" 304 -
[02/Mar/2022 23:45:12] "GET /static/js/app.js HTTP/1.1" 304 -
Time taken: 127744998

```



```

app.route('/')
@cache(timeout=50)
def index():
    return 'Cached for 50s'

```

### Caching Other Functions

Using the same @cached decorator you are able to cache the result of other non-view related functions. The only stipulation is that you replace the `key_prefix`, otherwise it will use the request path. Cache keys control what should be fetched from the cache. If, for example, a key does not exist in the cache, a new key-value entry will be created in the cache. Otherwise the value (i.e. the cached result) of the key will be returned:

```

@cache(timeout=50, key_prefix='all_comments')
def get_all_comments():
    comments = do_something_dbs()
    return [x.author for x in comments]

```

### Memoization

[See `memoize`](#)

In memoization, the function's arguments are also included into the cache key.

#### Note:

With functions that do not receive arguments, `cached()` and `memoize()` are effectively the same.

Memoization is also designed for methods, since it will take into account the identity of the 'self' or 'cls' argument as part of the cache key.

The theory behind memoization is that if you have a function you need to call several times in one request, it would only be calculated the first time that function is called with those arguments. For example, an sqlalchemy object that determines if a user has a role. You might need to call this function many times during a single request. To keep from hitting the database every time this information is needed you might do something like the following:

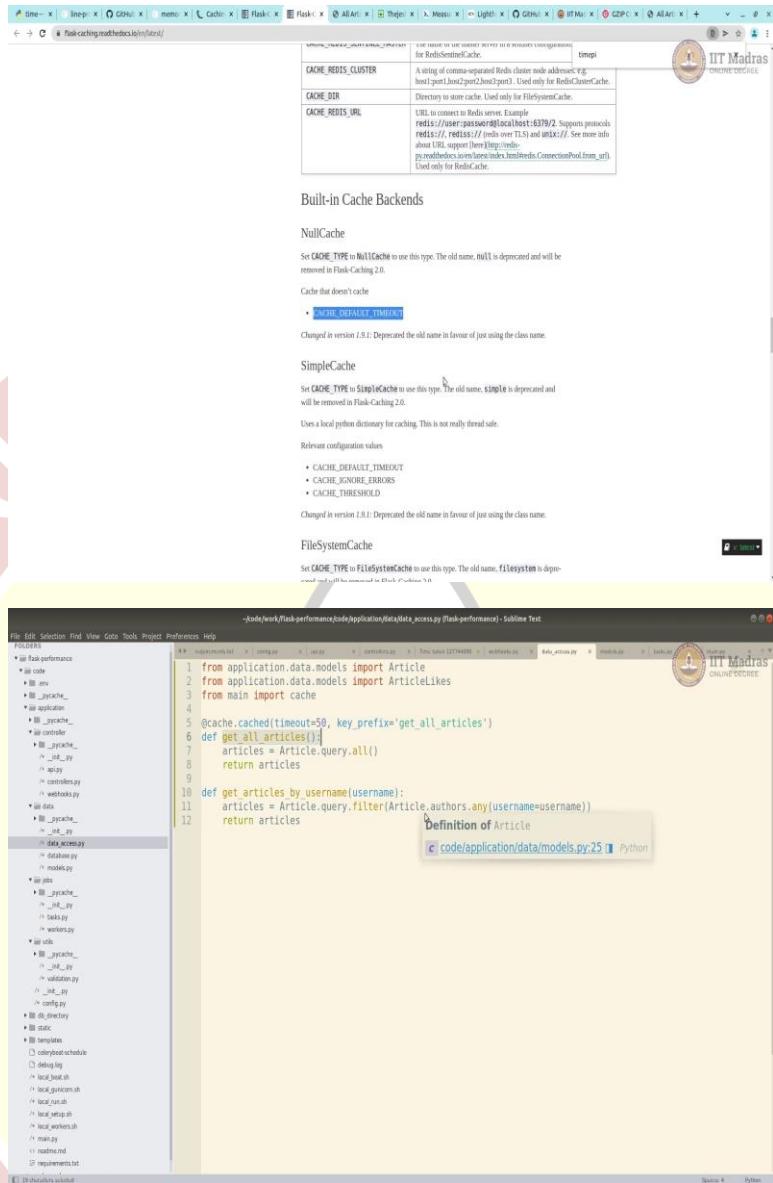
```

# File: /Users/ashish/PycharmProjects/flask-performance/application/config.py (Flask-Performance) - Sublime Text
File Edit Selection Find Goto Tools Project Preferences Help
Project: flask-performance
File: config.py
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    REDIS_URL = "redis://localhost:6379"
14    CACHE_TYPE = "RedisCache"
15    CACHE_REDIS_HOST = "localhost"
16    CACHE_REDIS_PORT = 6379
17
18
19 class LocalDevelopmentConfig(Config):
20     SQLITE_DB_DIR = os.path.join(basedir, "../db_directory")
21     SQLITE_DATABASE_URI = "sqlite:///{}+os.path.join(SQLITE_DB_DIR, "testdb.sqlite3")".format(SQLITE_DB_DIR)
22     DEBUG = True
23     SECRET_KEY = "ash ah seet"
24     SECURITY_PASSWORD_HASH = "bcrypt"
25     SECURITY_PASSWORD_SALT = "really super secret" # Read from ENV in your case
26     SECURITY_REGISTERABLE = True
27     SECURITY_CONFIRMABLE = False
28     SECURITY_SEND_REGISTER_EMAIL = False
29     SECURITY_UNAUTHORIZED_VIEW = None
30     WTF_CSRF_ENABLED = False
31     CELERY_BROKER_URL = "redis://localhost:6379/1"
32     CELERY_RESULT_BACKEND = "redis://localhost:6379/2"
33     REDIS_URL = "redis://localhost:6379"
34     CACHE_TYPE = "RedisCache"
35     CACHE_REDIS_HOST = "localhost"
36     CACHE_REDIS_PORT = 6379
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57

```

**INDIAN INSTITUTE OF TECHNOLOGY MADRAS**

**सिद्धिर्भवति कर्मजा**



Let us check whether this works, controller. Let us check if this works start this, took that much, that much, that much. So, let us check the difference, this is the first time axis. This is the subsequent accesses. So, you can see how small they are compared to the first time axis, at least 10 to more than 10,100 times smaller. It is a factor of that so it is significantly faster.

Now, if you had a much bigger, much bigger data, or much bigger database, it, this could have been this first number would have been much bigger. This first number would have been much bigger, this would be somewhat some similar, because it is still reading from memory. What we are using here is the flask caching cached here, this one, and key prefixes.

All comments, and usually you keep it unique. So, that if you are, you do not want two functions to return the same thing if you have the same, prefix, or if they have the same name. So, ideally, you would want the key prefix to be separate. So, that, the keys are unique, and are cached properly. The other one to remember is the timeout.

And if it automatically timed out, we could also define a standard timeout in our config. We are not defined here. But we can default timeout, we can define a default timeout, we are default timeout. Cache default timeout, we could use this could keep it to 300 milliseconds or something. And then you can overwrite based on the requirement of that specific function. You could also do this for this. But here is a small change.

This has to have for user name. So, this does not have any input to the output always remains the same, regardless of what is the user. What is the user name, or what is a, who is the user or regardless of whether he is accessing from the first page last page, whatever page, unless you have a page initial as part of this, but we do not have it.

But in this, it differs because this is articles by user name. This is usually getting called when we access this link here, and this could be different users. And we should cache it differently for different users. Otherwise, sometimes, if we do not do it properly, for all users, it will show this same content, which is wrong. That is why we are going to use this different caching way to do it. It is called memorize.

(Refer Slide Time: 23:47)

The screenshot shows a browser window with several tabs open, including 'Time', 'Keep', 'GHu', 'memo', 'Cache', 'Flask', 'All Ans', 'Then', 'Ans', 'Light', 'GHu', 'IT Madras', 'GZP', 'All Ans', and a test tab. The main content area displays Python code examples:

```
using our same sqlalchemy you are able to cache the results in your view methods
The only stipulation is that you replace the key prefix, otherwise it will use the
request path cache key. Keys control what should be fetched from the cache. If, for example, a
key does not exist in the cache, a new key-value entry will be created in the cache. Otherwise the
value (i.e. the cached result) of the key will be returned:

```

```
@cache.cached(timeout=50, key_prefix='all_comments')
def get_all_comments():
    comments = db.session.query()
    return [x.author for x in comments]

```

```
cached_comments = get_all_comments()
```

Memoization

See `memoize()`

Note:  
With functions that do not receive arguments, `cached()` and `memoize()` are effectively the same.

Memoize is also designed for methods, since it will take into account the identity of the 'self' or 'cls' argument as part of the cache key.

The theory behind memoization is that if you have a function you need to call several times in one request, it would only be calculated the first time that function is called with those arguments. For example, an sqlalchemy object that determines if a user has a role. You might need to call this function many times during a single request. To keep from hitting the database every time this information is needed you might do something like the following:

```
class Person(db.Model):
    @cache.memoize(50)
    def has_membership(self, role_id):
        return Group.query.filter_by(user=self, role_id=role_id).count() == 1
```

Warning:  
Using mutable objects (classes, etc) as part of the cache key can become sticky. It is suggested to not pass in an object instance into a memoized function. However, the memoize does perform a repr() on the passed in arguments so that if the object has a

File: articles.py Last performance issue application/articles/articles.py [Task performance] - Sublime Text

```
from application.data.models import Article
from application.data.models import ArticleLikes
from main import cache

@cache.cached(timeout=50, key_prefix='get_all_articles')
def get_all_articles():
    articles = Article.query.all()
    return articles

@cache.memoize(50)
def get_articles_by_username(username):
    articles = Article.query.filter(Article.authors.any(username=username))
    return articles
```



```
# NameError: name 'username' is not defined
NameError
NameError: name 'username' is not defined

Breakpoint (most recent call last)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 100, in __call__
    return self.wsgi_app(environ, start_response)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 203, in wsgi_app
    response = self.handle_exception(e)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 197, in wsgi_app
    return self.preprocess_exception(e)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 192, in preprocess_exception
    return self.error_router(e, request=request)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 187, in error_router
    return original_handler(e)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 171, in full_dispatch_request
    response = self.full_dispatch_request()
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 166, in full_dispatch_request
    rv = self.handle_user_exception(e)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 160, in handle_user_exception
    return self.dispatch_error(e)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 144, in dispatch_error
    return self.error_handler_spec[http://werkzeug.pocoo.org/1/error/#error](view, e)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 131, in articles_by_author
    articles = data_access.get_articles_by_username(username)
NameError: name 'username' is not defined.

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.
To switch between the interactive traceback and the plain one, you can click on the "Breakpoint" header. From the test traceback you can also create a paste of it. For code execution mouse-over the frame you want to debug and click on the console icon on the right side.
You can execute arbitrary Python code in the stack frames and there are some extra helpers available for introspection:
• dump(): dump all that's known about the frame
• dumpobj(): dump all that's known about the object

Brought to you by DON'T PANIC, your friendly thinking powered instant debugger.
```

```
# pickle.PicklingError
pickle.PicklingError: Can't pickle <class 'sqlalchemy.orm.session.SignallingSession'>: attribute lookup signallingSession on sqlalchemy.orm.session failed
pickle.PicklingError
pickle.PicklingError: Can't pickle <class 'sqlalchemy.orm.session.SignallingSession'>: attribute lookup signallingSession on sqlalchemy.orm.session failed

Breakpoint (most recent call last)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 100, in __call__
    return self.wsgi_app(environ, start_response)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 203, in wsgi_app
    response = self.handle_exception(e)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 197, in wsgi_app
    return self.preprocess_exception(e)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 192, in preprocess_exception
    return self.error_router(e, request=request)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 187, in error_router
    return original_handler(e)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 171, in full_dispatch_request
    response = self.full_dispatch_request()
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 166, in full_dispatch_request
    rv = self.handle_user_exception(e)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 160, in handle_user_exception
    return self.dispatch_error(e)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 144, in dispatch_error
    return self.error_handler_spec[http://werkzeug.pocoo.org/1/error/#error](view, e)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 131, in articles_by_author
    articles = data_access.get_articles_by_username(username)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 126, in __init__
    _DecoratedFunction.__init__(self, func, *args, **kwargs)
File "/home/krishnakumar/PycharmProjects/FromPackagetoHelloPy/venv/lib/python3.7/site-packages/werkzeug/_internal.py", line 125, in __init__
    raise pickle.PicklingError("Can't pickle %s: attribute lookup %s on %s failed"
pickle.PicklingError: Can't pickle <class 'sqlalchemy.orm.session.SignallingSession'>: attribute lookup SignallingSession on sqlalchemy.orm.session failed.

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.
To switch between the interactive traceback and the plain one, you can click on the "Breakpoint" header. From the test traceback you can also create a paste of it. For code execution mouse-over the frame you want to debug and click on the console icon on the right side.
You can execute arbitrary Python code in the stack frames and there are some extra helpers available for introspection:
• dump(): dump all that's known about the frame
• dumpobj(): dump all that's known about the object
```

So, let us go back to the documentation, see what memorize means. Here, so in memorization, the function arguments are also included in the cache key. So, along with their name, it is the function parameters like whatever the actual value of username is also used as a key. So, that if, indirectly it is like for example, the key is like, roughly, it would be like this.

Roughly the key would be like this, and hence that would be unique for each of the query. And this, we do not need to do and the memorize automatically handles it. And 50 is a timeout period. So, let us try that also. Let us go back and add the similar time out for that. Timers for the same, going to remove login required for now?

Now let us say we click on this user name is not defined made is this. This is user underscore name, stop this, I think it is similar. So, the way caching does is it actually takes the output and pickle sit and stores in the Redis. So, sometimes, if the object cannot be pickled and put in, then there will be an issue. So, let us see, what is the issue here?

(Refer Slide Time: 26:39)

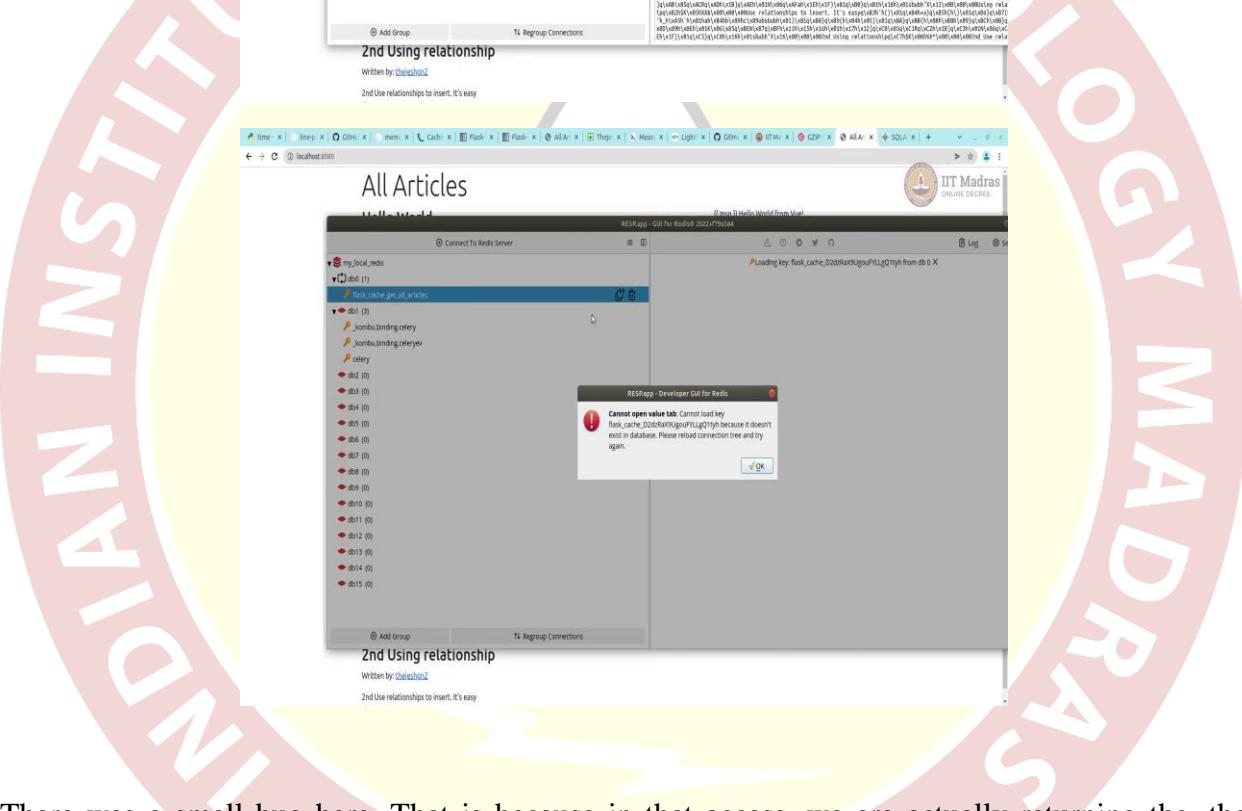
The screenshot displays a development environment with a code editor and a web browser.

**Code Editor:** Shows a Python file named `articles.py` with the following content:

```
1 from application.data.models import Article
2 from application.data.models import ArticleLikes
3 from main import cache
4
5 @cache.cached(timeout=50, key_prefix='get_all_articles')
6 def get_all_articles():
7     articles = Article.query.all()
8     return articles
9
10 @cache.memoize(50)
11 def get_articles_by_username(username):
12     articles = Article.query.filter(Article.authors.any(username=username))
13     print(str(articles))
14     return articles.all()
```

**Browser:** Shows a Flask application at `http://127.0.0.1:5000/articles/byThejeshgn2`. The page title is "Author Articles - thejeshgn2". It displays a list of articles:

- Hello World
- Written by thejeshgn2
- Welcome to Flask's documentation. Get started with Installation and then get an overview with the Quickstart. There is also a more detailed Tutorial that shows how to create a small but complete application with Flask. Common patterns are described in the Patterns for Flask section. The rest of the docs describe each component of Flask in detail, with a full reference in the API section.
- my new article**
- Written by thejeshgn2
- my new article content
- dummy new article**
- Written by thejeshgn2
- my dummy new article content
- Using relationship**
- Written by thejeshgn2
- Use relationships to insert. It's easy
- Znd Using relationship**
- Written by thejeshgn2
- Znd Use relationships to insert. It's easy



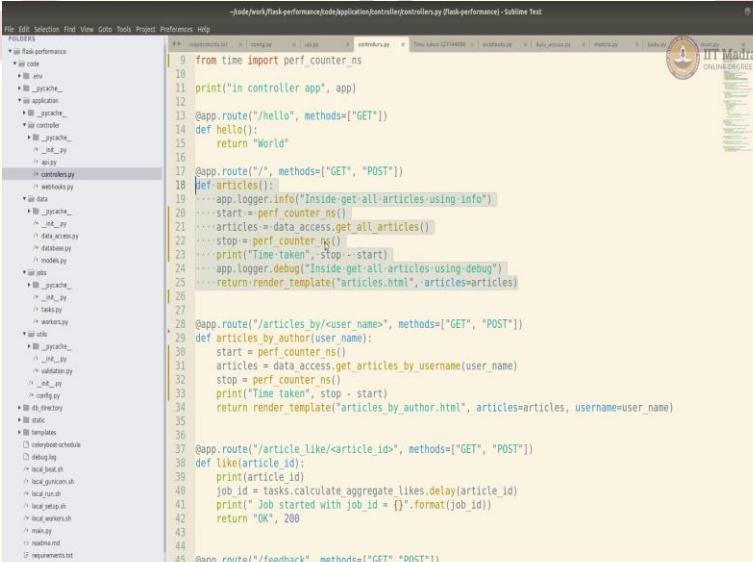
There was a small bug here. That is because in that access, we are actually returning the session and not the actual or its response of the query. So, I am just going to do dot all. So, it gets all the objects from the database, runs the queries and get dot all and then actually returns it. So, that result can be cached and not the reference. So, this will be actually the query. Now let us see if this works. Let us restart the server.

And there it works. And you can actually see it in the. See it in the Redis to by default reduce flash cache uses the database 0, if you do not give any, you can give the specific one, you can

refresh this to see. For example, this is the current one and it is been cached with a TTL, I think, which is reducing, if you keep reloading it, will keep changing reducing.

Once it expense it is actually query second from the database and get it. Let us go back to the homepage. And see, you can refresh this, you can see that, flash cache and get all articles is there. And it is quickly it is getting expired, the previous one got expired, and then this one is still valid for another 44 seconds. And even that is gone now. So, that is how you actually get the caching working on the backend in the database part.

(Refer Slide Time: 28:38)



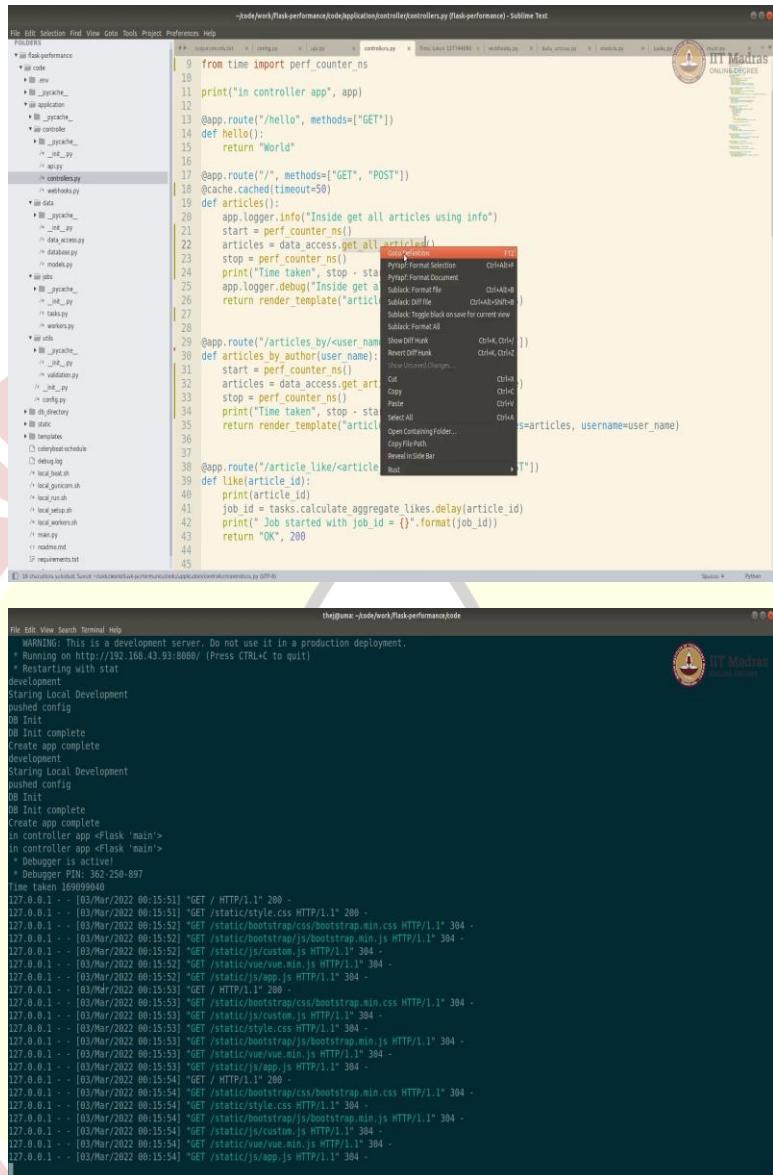
The screenshot shows the PyCharm IDE interface with the following details:

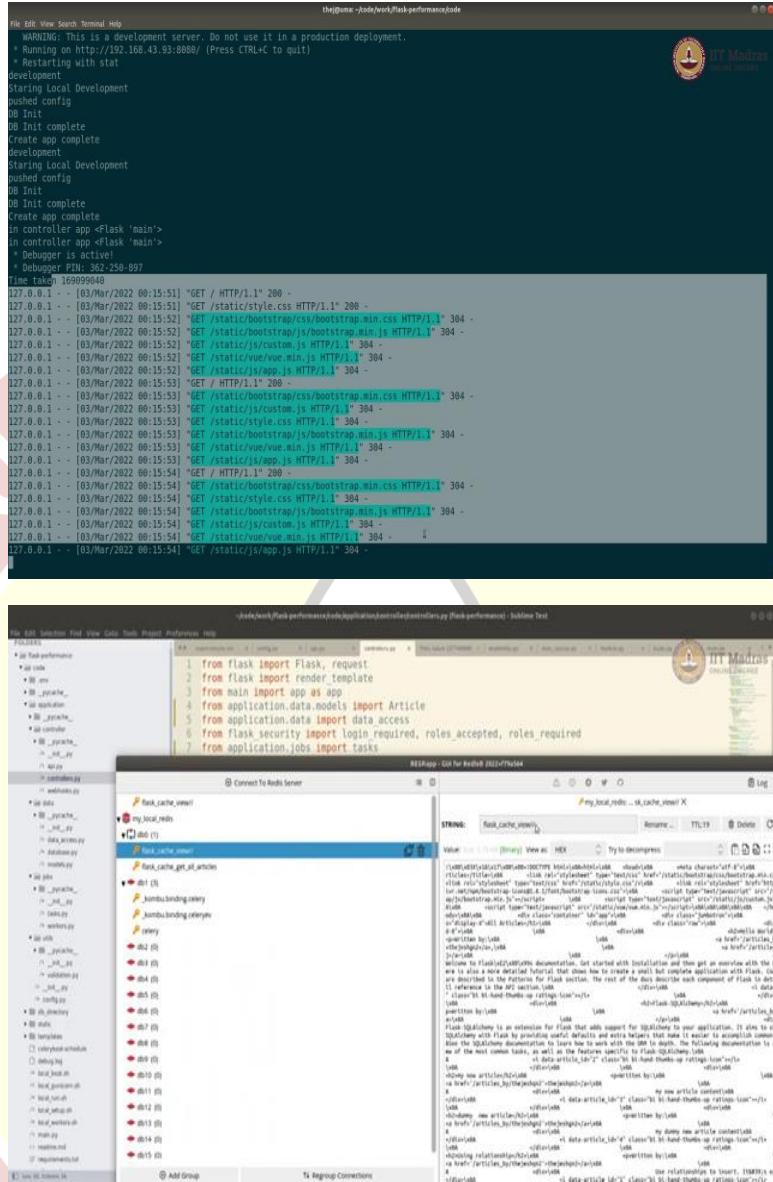
- File Path:** file:///work/Flask-performance/code/applications/controller/controllers.py (Task Performance) - Sublime Test
- Tool Bar:** File Edit Selection Find View Goto Tools Project Preferences Help
- Code Editor:** The main editor displays the `controllers.py` file with annotations for performance metrics. Annotations include:
  - Time taken: 127744000 ns at line 9: `from time import perf_counter_ns`
  - Time taken: 127744000 ns at line 19: `print("in controller app", app)`
  - Time taken: 127744000 ns at line 29: `def articles():`
  - Time taken: 127744000 ns at line 39: `def like(article_id):`
  - Time taken: 127744000 ns at line 49: `def feedback():`
- Sidebar:** Shows the project structure with expanded sections for `__init__.py`, `articles.py`, `like.py`, and `feedback.py`. The `__init__.py` section is highlighted.
- Bottom Status Bar:** Shows the message "8 lines, 200 characters selected".
- PyCharm Logo:** In the top right corner.
- IT Madras Online Classes:** A watermark logo in the top right corner.

Now, you could also say that, this whole thing does not change that often. If the database, values are not changing. And if it is, it same for logged in users logged out users and all of it and you do not want to change in your this part. You want to respond with the same values regardless then you can cache it at the whole controller level. You can cache it at this level.

This will be really useful in an API where you want, the response is going to be same. And you can cache the whole response so that you will avoid database calling, marshaling, demarshaling all of that effort and just respond quickly. It can work authentication. No, it does not like it does not mean that just pick just for an example, I removed, authentication for one of the thing, because I did not want to log in. Otherwise, the caching also works with authentication.

(Refer Slide Time: 29:40)





Now let us see here. Let us add a caching at the whole articles level. Let us check whether how to add it. So, it would be just that, so let us see if it caches for let us say, this is cache for 50 and let us say this is cache for much smaller time. You could be 20. And it imported in the controllers. Start this; now let me just go back to the browser. And if we do this, and this, this, you can see that the first time we see that time taken has been printed.

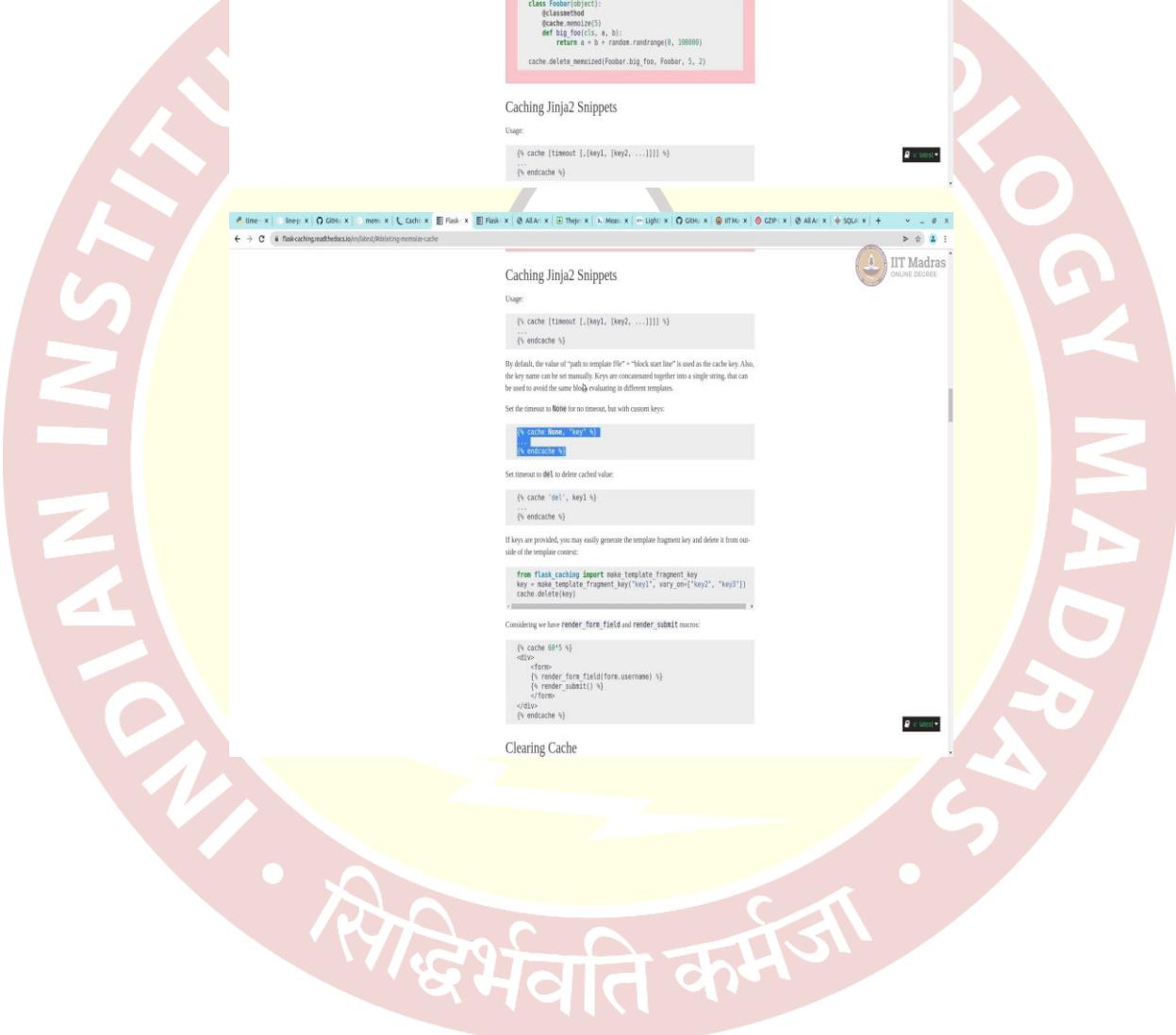
The rest of the times no time taken has been printed that because it does not even enter this. It responds directly from here, we can even see that the cache, you can see, let us refresh this and enable auto refresh as well see you can see the whole view has been slash view. That it is

showing slash the whole view, all HTML has been cached. So, that the whole time the takes to run this, render this, all of that gets saved, because we are just loading it from the cache.

(Refer Slide Time: 31:36)

The screenshot shows a developer's workspace with several open applications:

- Top Left:** A file explorer window showing the project structure of a Flask application. The `controller` directory is expanded, revealing files like `controller.py`, `articles.py`, and `articles.html`. Other files visible include `app.py`, `data\_access.py`, `models.py`, `jobs.py`, `utils.py`, `workers.py`, and configuration files like `config.py` and `requirements.txt`.
- Top Right:** A terminal window with the title "jupyter notebook" running on port 8888. It displays Python code for a Flask application, including imports for Flask, Request, RenderTemplate, and Article, along with route definitions for "/hello" and "/".
- Middle Left:** A terminal window titled "time -> C" showing the command "flask caching methods".
- Middle Center:** A browser tab titled "Caching Jinja2 Snippets" containing a snippet of Python code for a class-based view named `Fedor` that handles caching logic using `@cache.memoize` and `cache.delete`.
- Middle Right:** Another browser tab titled "Flask - flask - flask.pocoo.org" showing a snippet of Jinja2 template code with cache-related directives like `{{ cache }}` and `{{ cache|del }}`.
- Bottom:** A terminal window titled "Considering we have render\_form field and render\_submit macros:" which is partially cut off at the bottom of the screen.



New in version 0.2.

You might need to delete the cache on a per-function basis. Using the above example, let say you change the user's permissions and assign them to a role, but now you need to re-calculate if they have certain membership or not. You can do this with the `delete_memoized()` function:

```
cache.delete_memoized(user.has_membership)
```

**Note:**

If only the function name is given as parameter, all the memoized versions of it will be invalidated. However, you can delete specific cache by providing the same parameter values as when caching. In following example only the user-role cache is deleted:

```
user.has_membership('demo', 'admin')
user.has_membership('demo', 'user')
cache.delete_memoized(user.has_membership, 'demo', 'user')
```

**Warning:**

If a classmethod is memoized, you must provide the class as the first \*args argument:

```
class FooBar(object):
    @classmethod
    @cache.memoized()
    def big_bar(a, b):
        return a + b + random.randrange(0, 100000)
cache.delete_memoized(FooBar.big_bar, FooBar, 5, 2)
```

**Caching Jinja2 Snippets**

Usage:

```
{% cache [timeout [, [key1, [key2, ...]]]] %}
...
{% endcache %}
```

By default, the value of "path to template file" + "black star line" is used as the cache key. Also, the key name can be set manually. Keys are concatenated together into a single string, that can be used to avoid the same block evaluating in different templates.

Set timeout to None for no timeout, but with custom key:

```
{% cache None 'key' %}
...
{% endcache %}
```

Set timeout to del to delete cached value:

```
{% cache 'del', key %}
...
{% endcache %}
```

If keys are provided, you may easily generate the template fragment key and delete it from outside of the template context:

```
from flask_caching import make_template_fragment_key
key = make_template_fragment_key('key1', vary_on=['key2', 'key3'])
cache.delete(key)
```

Considering we have `render_form_field` and `render_submit` macros:

```
{% cache 60*5 %}
<div>
<form>
  {% render_form_field(form.username) %}
  {% render_submit() %}
</form>
</div>
{% endcache %}
```

**Clearing Cache**



The screenshot shows the PyCharm IDE interface with the following details:

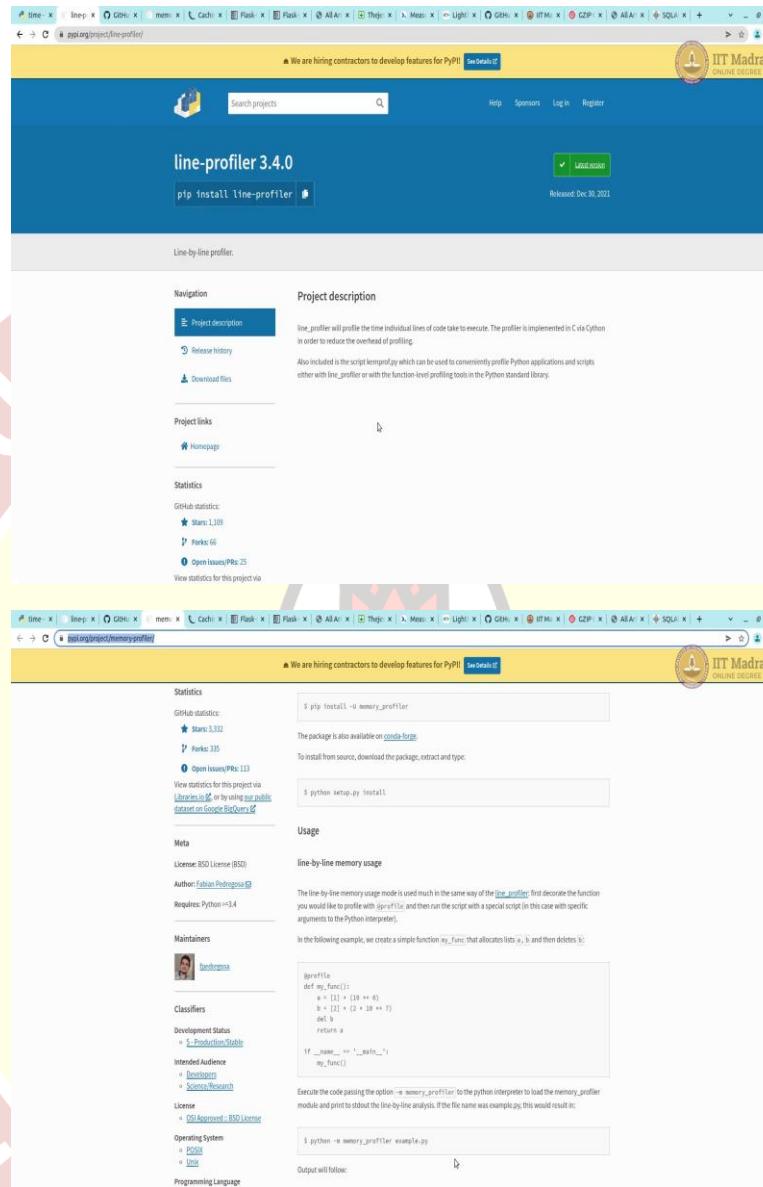
- File Structure:** The left sidebar displays the project structure. The file `data_access.py` is highlighted in the `application` directory.
- Code Editor:** The main editor window contains the Python code for `data_access.py`. The code defines a function `get_articles_by_username` that uses a database query to filter articles by a specific user's ID.
- Code Analysis Results:** A vertical bar on the right side of the editor highlights specific lines of code:
  - Line 1: `from application.data.models import Article`
  - Line 2: `from application.data.models import ArticleLikes`
  - Line 3: `from main import cache`
  - Line 5: `@cache.memoize(50)`
  - Line 11: `def get_articles_by_username(username):`
  - Line 12:  `articles = Article.query.filter(Article.authors.any(username=username))`
  - Line 13:  `print(str(articles))`
  - Line 14:  `return articles.all()`
- Toolbars and Status Bar:** The top bar includes standard options like File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, Help, and a status message "-code-work://task-performance/code/application/data/data\_access.py [Flash-performance] - Sublime Text". The bottom status bar shows "19 characters selected".
- PyCharm Logo:** A watermark logo for "IIT Madras ONLINE COURSE" is visible in the top right corner.

Now, you can also time out the caches when you want, if you want to immediately clear, for some reason, for example, you have a long term cache for getting reading the things. But when you post something, you want to expire, the cache. That also you can do by expiring, the cache or deleting the cache, basically, you could it same function as deleting this from here and that would expire, the cache. Or you could have a function called expire or delete, let us just check, you can delete the memorized cache, or you can delete by key as well. Both of them will work.

Now, you can also use caching within Jinja2 template if you want to cache just like how in controller, the cache only small part of it. If this was not used, we are caching only this part. Similarly, within the whole article dot HTML, if you want to cache only small parts of it, you could cache that as well, you can see that, in the tutorial, I am not going to show how to do that, you can see it in the tutorial how to do it.

Now this is one step. And the most often used step to improve the performance. You also saw that I am printing the queries here, when I ran, and you could use explain on the database, to get explanation on the query, how it runs, and then you can add indexes and stuff. I am not going to take you through that because I think you would have learned that as part of your DBMS. Similarly, you could also do a memory.

(Refer Slide Time: 33:27)



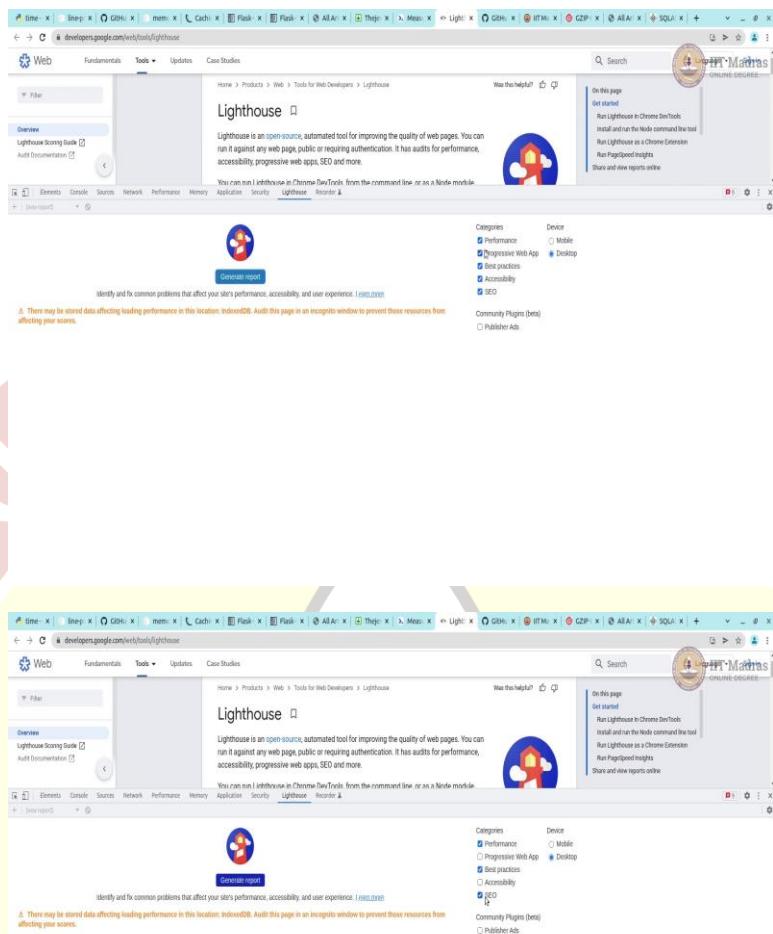
Or line profiling, line profiling or memory profiling at the Python level to check, how much memory it is getting used, are there any objects that is not been garbage collected, etcetera, etcetera. To improve the performance of the Python program, this is similar to any other, Python program it does not have to be just for flask.

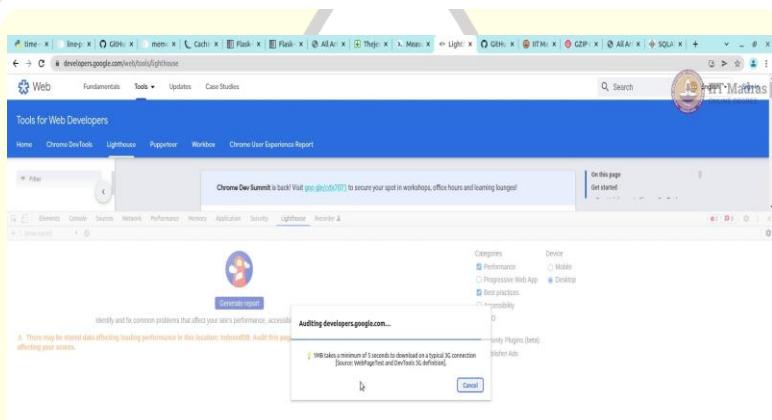
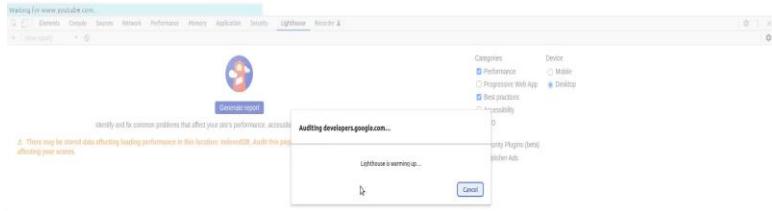
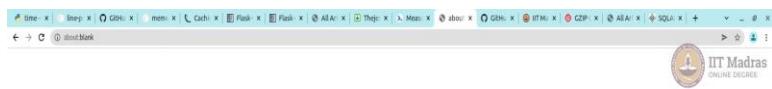
If you can do this with your regular Python program, you can do in a similar way for flask app 2. There, I am going to just stop the backend optimization, and I am going to show how to measure the front end performance using something called Lighthouse.

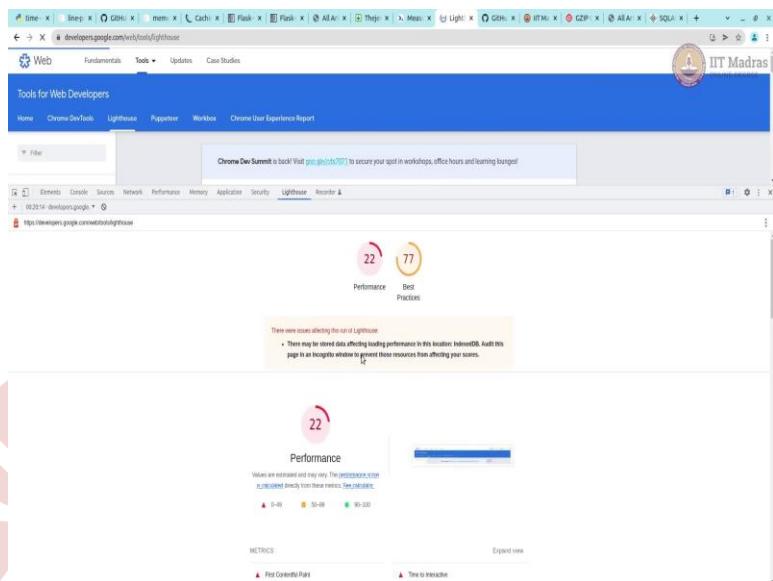
(Refer Slide Time: 34:14)

The image displays two screenshots of browser extensions used for website performance analysis:

- Top Screenshot:** A screenshot of the **githsSpeed** extension interface. It shows the URL <https://onlinedegree.iltm.ac.in/> and a large green box indicating "ZIP Is Enabled". Below this, detailed compression statistics are provided:
  - Original Size: 143.7 kB
  - GZIP Size(compressed): 33.9 kB
  - Compression %: 76.39% is compressedFurther down, "More Information" includes:
  - HTTP Status: 200 OK
  - Request Time: 2432 ms
  - Compression Time: 0.0089 s
  - Content Type: text/html
  - Server: Google-Frontend
- Bottom Screenshot:** A screenshot of the **Google DevTools Lighthouse** extension interface. It shows the URL <https://onlinedegree.iltm.ac.in/> and a summary message: "How Does This Test Work? This test will check whether your website (and ultimately the server your website is hosted on) has gzip or brotli enabled by connecting to your domain and requesting the necessary information. Next to it will also give you some additional information, like the file size of the original version of the web page you test and how much you (may) benefit from using gzip compression." Below this, there is a "What Does Gzip Compression Do?" section.



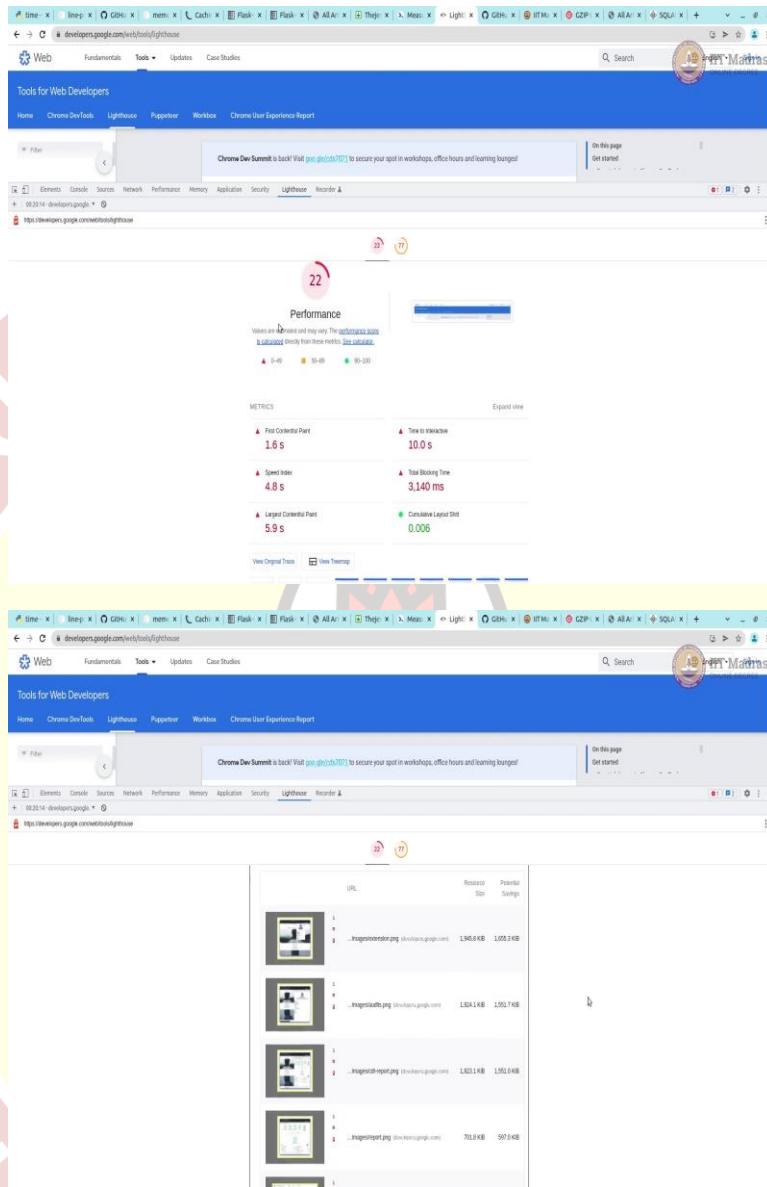


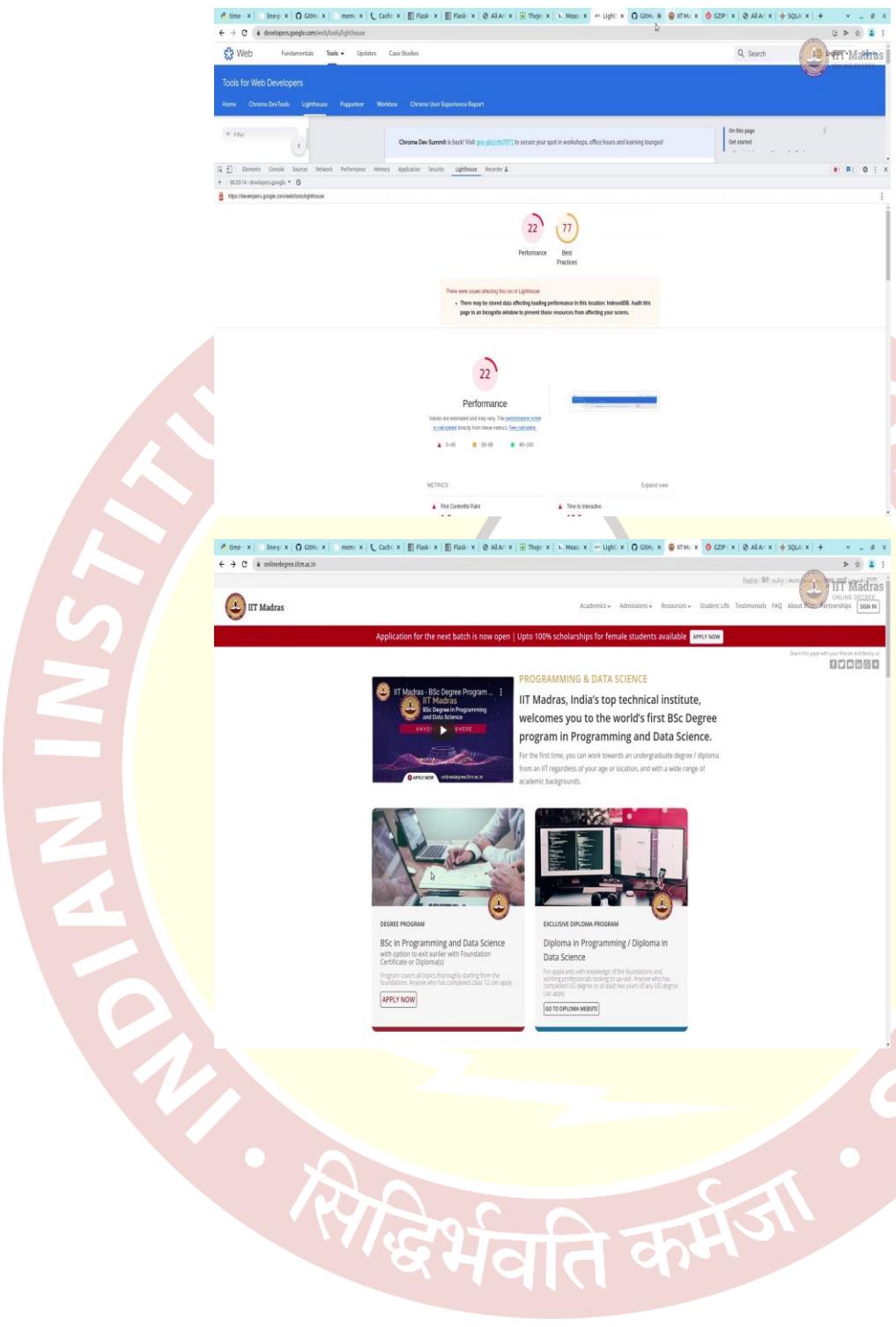


Now lighthouse is a project. Maybe I am not let us is an open source project by Google or for measuring performance. It is also been integrated into the Chrome. So, if you are in right click and click on Inspect, you can go to the lighthouse and it will show Chrome. And then you can choose the categories, I will probably choose.

You can do Progressive Web App checks, Performance checks, Best practices, Accessibility, and SEO checks. I am just going to check because there is this is not a, I am going to just check these three. Click on this and generate report will take couple of minutes to run, gives you a report or some instructions on how to improve any performance related issue or the best practice related issue. It is done. So, here you can see the performance is 22 and best practice is 77.

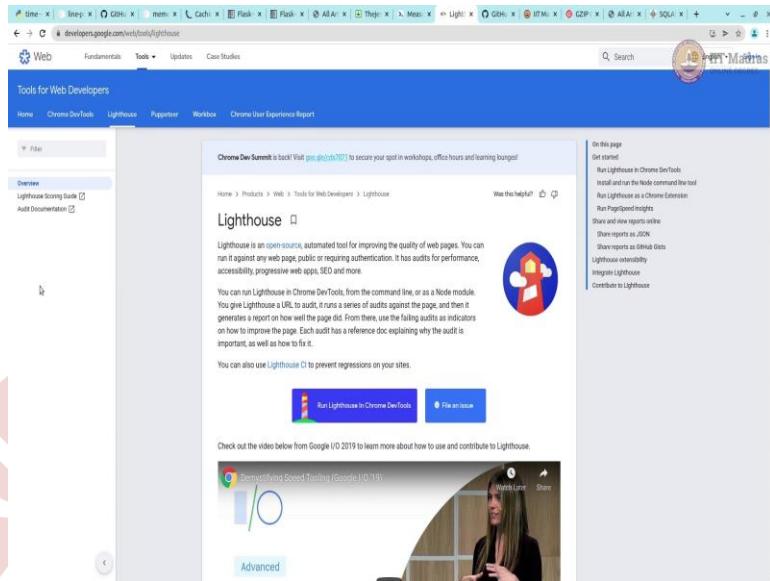
(Refer Slide Time: 35:35)











It also shows, if it is in red, it is actually quite bad, yellow is and green is great, shows some of the values how much each one, took and what it means like the image size is big, you could reduce the image size to improve the loading time, for example, that is one of the inputs that you can use. Format of the images, for example, I think most of them are using png here.

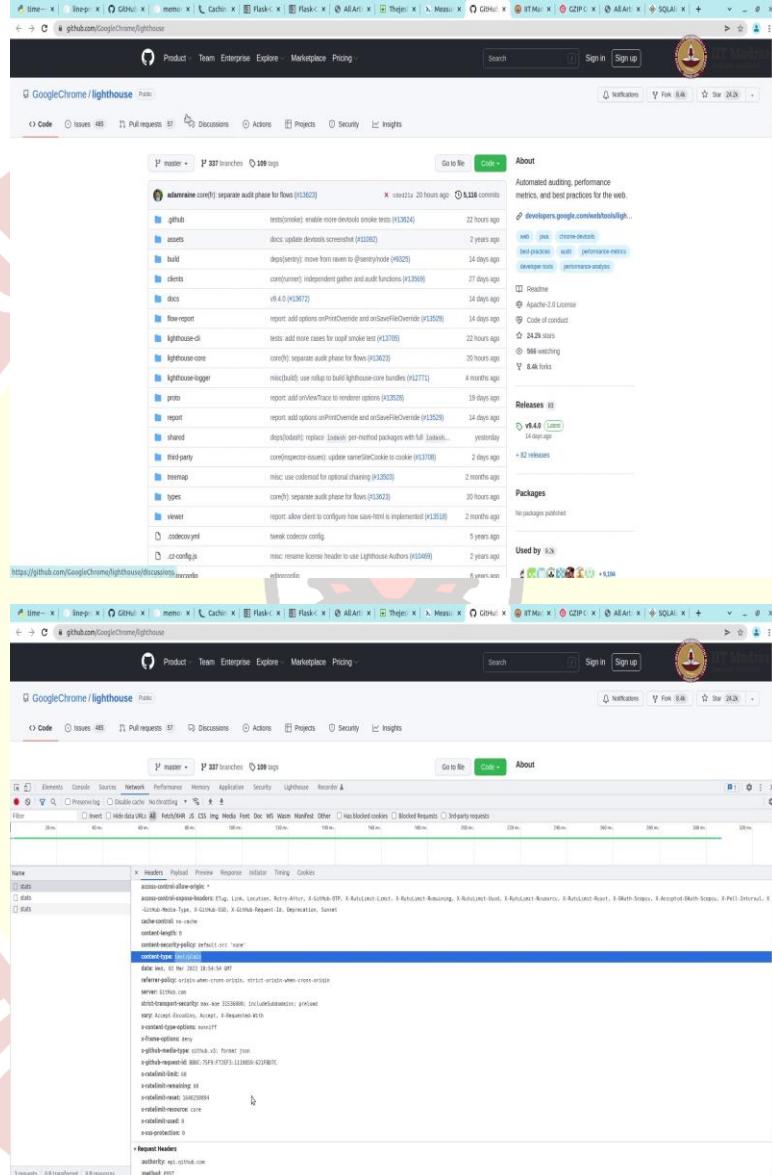
You could use other formats like, WebP, especially when you are delivering to Chrome to improve the speed. Many other I think you can go through them and see to work on some of them, you do not need to work on all of them to work, you can work on some of the time, which actually brings a larger change.

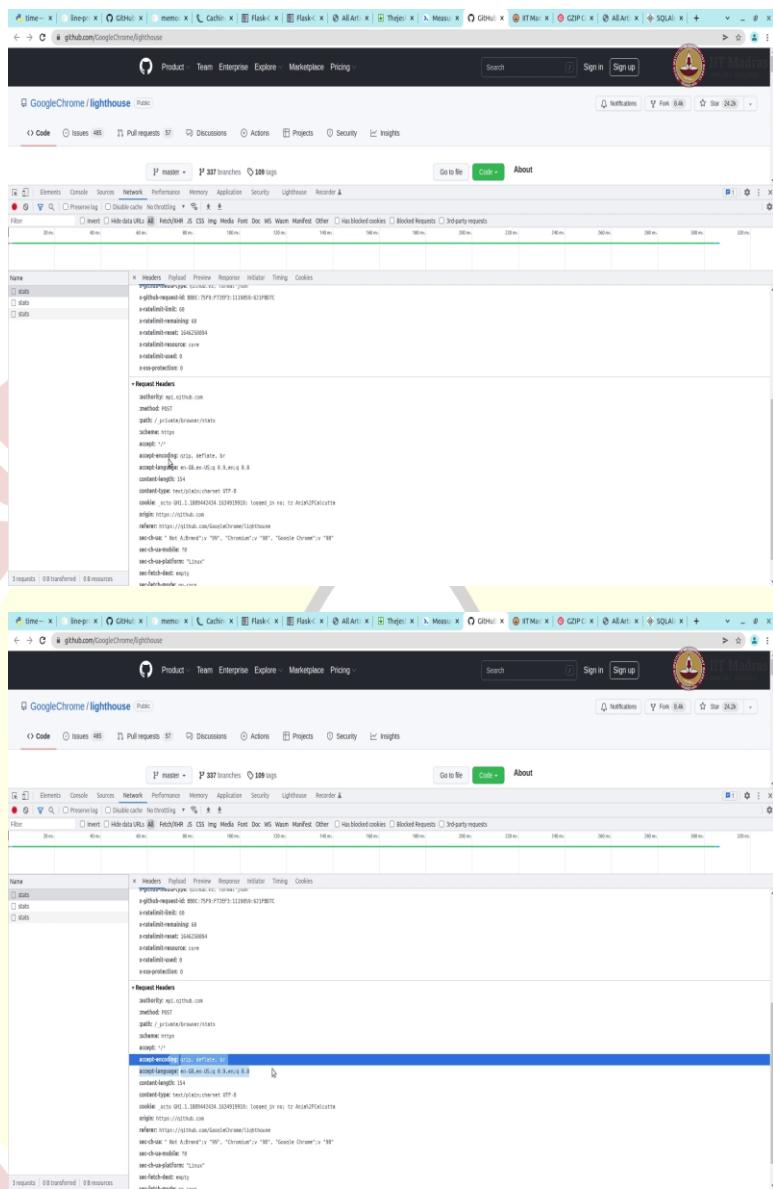
And you can improve this, for example, in our IIT degree site go to the lighthouse, I am just going to say the same thing, click on Generate Report, you can say that performance is 56. And the best practice is 77. Sometimes it also depends on the speed of the Internet. I mean, if your speed is or the browser is slow, it could reduce.

You can run it multiple times to take an average of how the performance is, it does shows like, you could I am using third party, for example, YouTube Embedded and I could reduce it to make it faster, things like that. Some of them, you cannot do anything, but some of the others things you can work on them to improve, the tips are actually good, you should go through them and improve. If you are interested in accessibility, you should run the accessibility test as well.

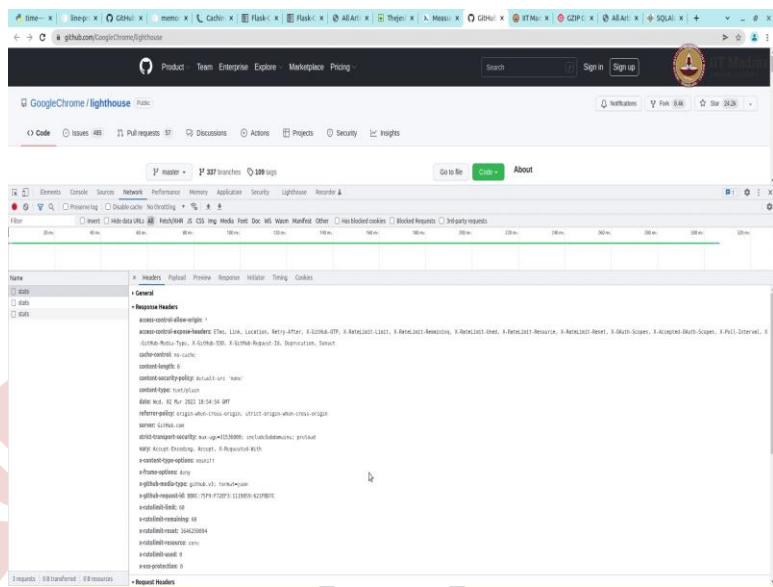
The lighthouse site itself is actually quite useful, you can check the scoring guide and tips how to improve the, scoring and that is very useful.

(Refer Slide Time: 38:01)





The image shows a screenshot of a browser's developer tools Network tab. A specific POST request is highlighted. The request URL is `/private-browser/stats`. The request headers include `accept-encoding: gzip, deflate, br;`, `accept-language: en-US,en;q=0.8,hi;q=0.8;`, and `content-length: 114`. The response status is `200 OK` with `content-type: text/plain;charset=UTF-8`. The response body contains JSON data. The browser tabs at the top show various extensions and tools like GitHub, Cache, and Lighthouse. The browser title bar indicates it's a Google Chrome session. The background features a watermark with the text "INDIAN INSTITUTE OF LOGY MADRAS" and "सिद्धिर्भवति कर्मजा".



The other thing I would want you to look at this whether the gzipped caching is enabled for any site for most of the requests, you can look at the network. For example, if we look at the network, it shows multiple response headers, and also shows whether the gzipped is enabled or disabled. Gzipped is a way of compressing the content in text plain or text, HTML, compressed on the server side and sent gzipped basically.

And send to the client and the client like browser like Chrome can decrypt it and show to the user; that way the bandwidth that is used on the transport is very less and it will be much faster. So, it can, you can, when you send a request, you can say that or your browser actually sees that it can accept gzipped. And most of the time servers will respond with a gzipped content.

(Refer Slide Time: 39:15)

```
Mr. ERIK VIVEK SINGH Terminal Help
development
Starting Local Development
pushed config
BB Init
BB Init complete
Create app complete
development
Starting Local Development
pushed config
BB Init
BB Init complete
Create app complete
on controller app <Flask 'main'>
  @app.route('/')
    def index():
        return "Hello, World!"
```

\* Debugger is active!

\* Debugger PIN: 362-250-897

Time taken: 16999994ns

127.0.0.1 - [03/Mar/2022 00:15:51] "GET / HTTP/1.1" 200 -
127.0.0.1 - [03/Mar/2022 00:15:51] "GET /static/style.css HTTP/1.1" 200 -
127.0.0.1 - [03/Mar/2022 00:15:52] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:52] "GET /static/bootstrap/js/bootstrap.min.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:52] "GET /static/js/custom.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:52] "GET /static/vue/vue.min.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:52] "GET /static/js/app.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:53] "GET / HTTP/1.1" 200 -
127.0.0.1 - [03/Mar/2022 00:15:53] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:53] "GET /static/js/custom.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:53] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:53] "GET /static/bootstrap/js/bootstrap.min.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:53] "GET /static/vue/vue.min.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:54] "GET /static/js/app.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:54] "GET / HTTP/1.1" 200 -
127.0.0.1 - [03/Mar/2022 00:15:54] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:54] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:55] "GET /static/bootstrap/js/bootstrap.min.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:55] "GET /static/js/custom.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:55] "GET /static/vue/vue.min.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:55] "GET /static/js/app.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:55] "GET / HTTP/1.1" 200 -
Detected change in '/home/thej/code/work/flask-performance/code/application/controller/controllers.py', reloading
\* Restarting with stat
development
Starting Local Development
pushed config

```
File Edit View Search Terminal Help
> Debugger: PWN: 362-250-897
Time taken: 160899040
127.0.0.1 - [03/Mar/2022 00:15:51] "GET / HTTP/1.1" 200 -
127.0.0.1 - [03/Mar/2022 00:15:51] "GET /static/style.css HTTP/1.1" 200 -
127.0.0.1 - [03/Mar/2022 00:15:52] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:52] "GET /static/bootstrap/js/bootstrap.min.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:52] "GET /static/js/custom.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:52] "GET /static/vue/vue.min.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:52] "GET /static/js/app.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:53] "GET / HTTP/1.1" 200 -
127.0.0.1 - [03/Mar/2022 00:15:53] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:53] "GET /static/js/custom.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:53] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:53] "GET /static/bootstrap/js/bootstrap.min.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:53] "GET /static/js/custom.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:53] "GET /static/vue/vue.min.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:54] "GET /static/js/app.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:54] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:54] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:54] "GET /static/bootstrap/js/bootstrap.min.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:54] "GET /static/js/custom.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:54] "GET /static/vue/vue.min.js HTTP/1.1" 304 -
127.0.0.1 - [03/Mar/2022 00:15:54] "GET /static/js/app.js HTTP/1.1" 304 -
? Detected change in /home/thejame/code/work/Flask-performance/code/application/controller/controllers.py, reloading
? Restarting with stat
development
Starting Local Development
pushed config
?8 Init
?8 Init complete
Create app complete
development
Starting Local Development
pushed config
?8 Init
?8 Init complete
Create app complete
in controller app <flask 'main'>
in controller app <flask 'main'>
? Debugger is active!
? Debugger PWN: 362-250-897
```



The screenshot shows two browser windows side-by-side. Both windows have tabs for 'time', 'Index', 'GitHub', 'menu', 'Cache', and 'Flask'. The left window displays the 'Caching Jinja2 Snippets' page from flask-caching.readthedocs.io, showing code snippets for using the cache directive in Jinja2 templates. The right window displays the 'Flask-Caching' page from the same URL, which provides an overview of the extension, its version support (Python 3.5+), installation instructions (using pip), and configuration examples. The IIT Madras logo is visible in the top right corner of both pages.

This may not work with our development server but it will surely work with the proxy like nginx that we shown in the last deployment process and then you can enable the gzipped. Those are the few things that you can do to improve the performance. And again, this is like much vast subject and we can keep looking at more and more ways to improve the performance.

The first thing is to identify where the bottleneck and I would like you to add Performance Counters everywhere that is possible and use cache content as much as possible. Because it is, one, it is easy to implement and easy to use. And two, it brings a lot of value in the sense that it makes your application really fast without much effort. Thank you for watching, that is all.