INDIAN INSTITUTE OF TECHNOLOGY MADRAS

सिद्धिर्भवति कर्मजा

IIT Madras

ONLINE DEGREE

Modern Application Development - 1

Thejesh G N
Software Consultant
Institute of Information Technology & Management B Sc Degree
Indian Institute of Technology, Madras

Introduction to Flask-RESTful - 1

(Refer Slide Time: 0:12)

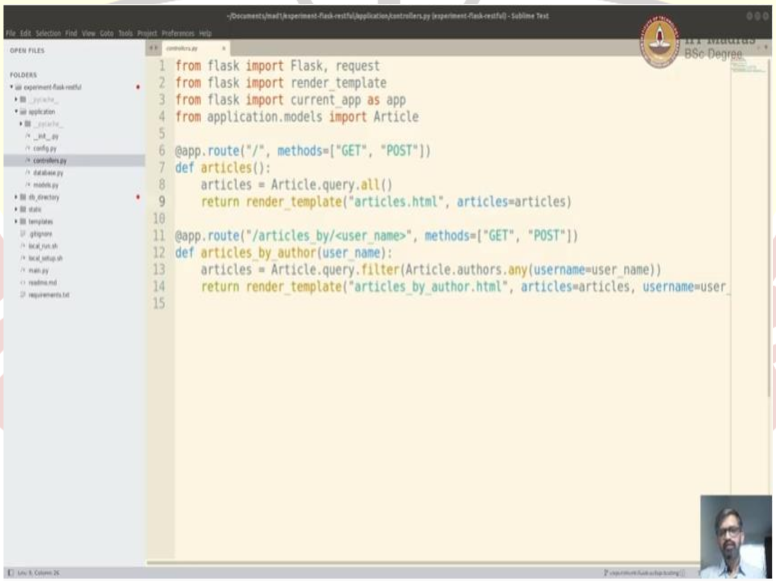
Modern Application Development - I

Introduction to Flask-RESTful - I

Thejesh G N

Software Consultant

IITM BSc Degree, IIT Madras

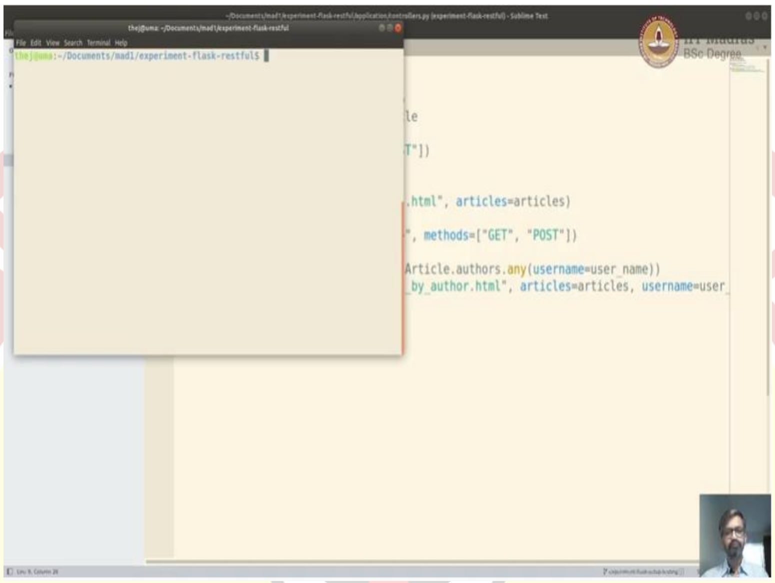


```
from flask import Flask, request
from flask import render_template
from flask import current_app as app
from application.models import Article

@app.route("/", methods=["GET", "POST"])
def articles():
    articles = Article.query.all()
    return render_template("articles.html", articles=articles)

@app.route("/articles_by/<user_name>", methods=["GET", "POST"])
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, username=user_
```

Welcome to the Modern Application Development screencast. In this short screencast, we will learn how to build a simple restful API using SQLAlchemy in flask. Before we start open the following applications on your desktop, so that you can work along with me, a browser, Chrome or Firefox, text editor or an IDE.

In this case, I am using sublime text, any IDE would be ok, I am just using the IDE for syntax highlighting and some basic help, not really using it in depth one, especially because IDE's differ, and I want to use the bare minimum of functionality from an IDE. So that any IDE you would be using, you should be able to follow.

(Refer Slide Time: 1:09)

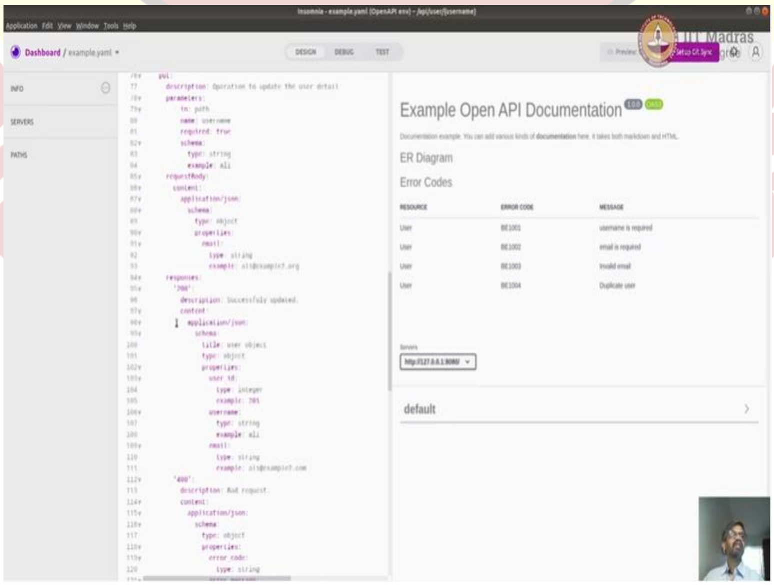

~/Documents/mad1/experiment-flask-restful/application/controllers.py (experiment-flask-restful) - Sublime Text
thej@uma: ~/Documents/mad1/experiment-flask-restful
File Edit View Search Terminal Help

```
thej@uma:~/Documents/mad1/experiment-flask-restful$
```

```
le

T"])

.html", articles=articles)

", methods=["GET", "POST"])

Article.authors.any(username=user_name))
_by_author.html", articles=articles, username=user_
```

Terminal, I am just going to use the terminal that comes built into you, know Your Ubuntu desktop and one of the API browsers.

(Refer Slide Time: 1:19)


Insomnia - example.yaml (OpenAPI env) - /api/user/{username}
Application Edit View Window Tools Help
Dashboard / example.yaml
DESIGN
DEBUG
TEST
INFO
SERVERS
PATHS
Preview
Setup Git Sync

```
    put:
      description: Operation to update the user detail
      parameters:
        - in: path
          name: username
          required: true
          schema:
            type: string
            example: ali
      requestBody:
        content:
          application/json:
            schema:
              type: object
              properties:
                email:
                  type: string
                  example: ali@example.org
      responses:
        '200':
          description: Successfuly updated.
          content:
            application/json:
              schema:
                title: user object
                type: object
                properties:
                  user_id:
                    type: integer
                    example: 101
                  username:
                    type: string
                    example: ali
                  email:
                    type: string
                    example: ali@example.com
        '400':
          description: Bad request.
          content:
            application/json:
              schema:
                type: object
                properties:
                  error_code:
                    type: string
```

Example Open API Documentation 1.0.0 OAS3

Documentation example. You can add various kinds of **documentation** here. It takes both markdown and HTML.

ER Diagram

Error Codes

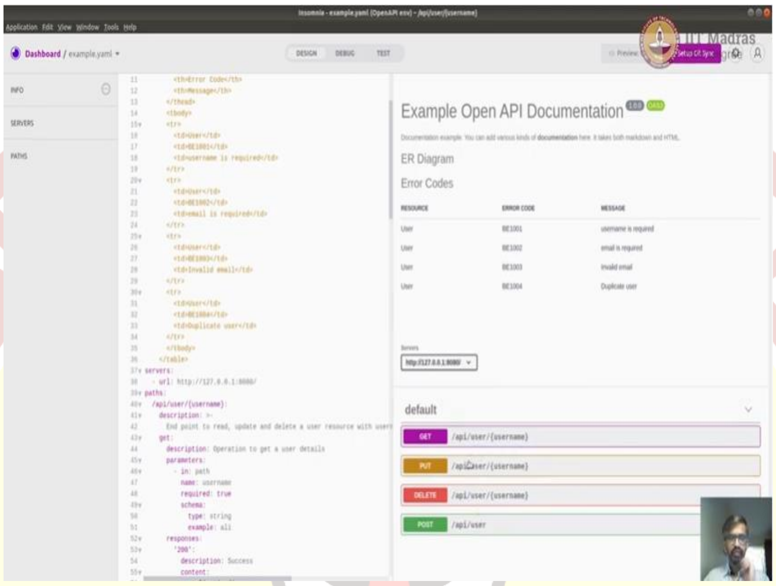
RESOURCE	ERROR CODE	MESSAGE
User	BE1001	username is required
User	BE1002	email is required
User	BE1003	Invalid email
User	BE1004	Duplicate user

Servers
http://127.0.0.1:8080/

default

In this case, I am going to use insomnia that we have seen before now before we start building an API. We need to know what API we are going to build. That starts with the documentation like last time, we saw how a documentation is built. Usually, when you are a developer and a designer wants you to implement an API, he gives you documentation and one such format is open API and you would have already gone through the details of it in our previous screencast.

(Refer Slide Time: 2:06)



Insomnia - example.yaml (OpenAPI env) - /api/user/{username}
Application Edit View Window Tools Help
Dashboard / example.yaml
DESIGN
DEBUG
TEST
INFO
SERVERS
PATHS
Preview
Setup Git Sync

```
          <th>Error Code</th>
          <th>Message</th>
        </thead>
        <tbody>
        <tr>
          <td>User</td>
          <td>BE1001</td>
          <td>username is required</td>
        </tr>
        <tr>
          <td>User</td>
          <td>BE1002</td>
          <td>email is required</td>
        </tr>
        <tr>
          <td>User</td>
          <td>BE1003</td>
          <td>Invalid email</td>
        </tr>
        <tr>
          <td>User</td>
          <td>BE1004</td>
          <td>Duplicate user</td>
        </tr>
        </tbody>
      </table>
servers:
  - url: http://127.0.0.1:8080/
paths:
  /api/user/{username}:
    description: >-
      End point to read, update and delete a user resource with usern
    get:
      description: Operation to get a user details
      parameters:
        - in: path
          name: username
          required: true
          schema:
            type: string
            example: ali
      responses:
        '200':
          description: Success
          content:
```

Example Open API Documentation 1.0.0 OAS3

Documentation example. You can add various kinds of **documentation** here. It takes both markdown and HTML.

ER Diagram

Error Codes

RESOURCE	ERROR CODE	MESSAGE
User	BE1001	username is required
User	BE1002	email is required
User	BE1003	Invalid email
User	BE1004	Duplicate user

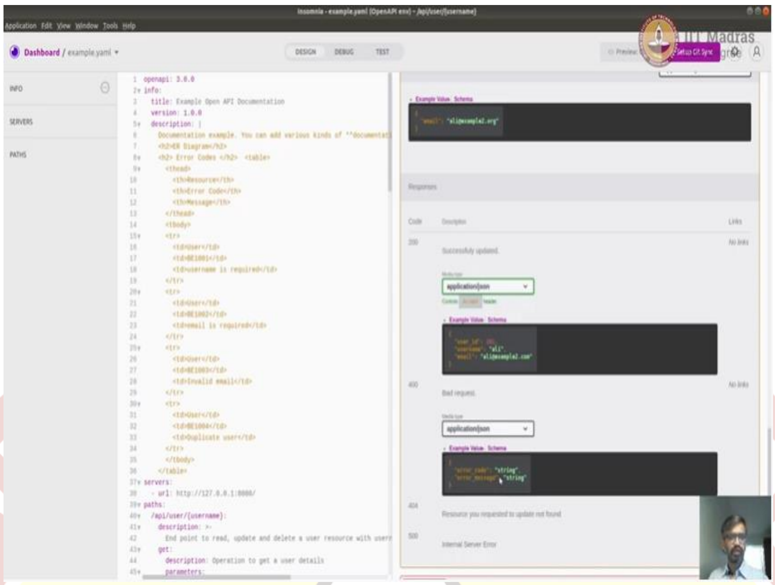
Servers
http://127.0.0.1:8080/

default

GET /api/user/{username}
PUT /api/user/{username}
DELETE /api/user/{username}
POST /api/user

Now here is that same document that we built in the last screencast, which defines the API for user model or user entity. It has just four APIs, all crud related ones. So just read, update, put and delete and then some error codes. What should it throw when, when there is an error and what should be the output formats, like in case of, for example, valid output: What how it should look like, or, if in case of invalid or if there is an error, how the output should look like.

(Refer Slide Time: 2:30)



```
openapi: 3.0.0
info:
  title: Example Open API Documentation
  version: 1.0.0
  description: |
    Documentation example. You can add various kinds of **documentati
    <h2>ER Diagram</h2>
    <h2> Error Codes </h2>  <table>
      <thead>
        <th>Resource</th>
        <th>Error Code</th>
        <th>Message</th>
      </thead>
      <tbody>
      <tr>
        <td>User</td>
        <td>BE1001</td>
        <td>username is required</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1002</td>
        <td>email is required</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1003</td>
        <td>Invalid email</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1004</td>
        <td>Duplicate user</td>
      </tr>
      </tbody>
    </table>
servers:
  - url: http://127.0.0.1:8080/
paths:
  /api/user/{username}:
    description: >-
      End point to read, update and delete a user resource with userr
    get:
      description: Operation to get a user details
      parameters:
```

```
{
  "email": "ali@example2.org"
}
```

Responses

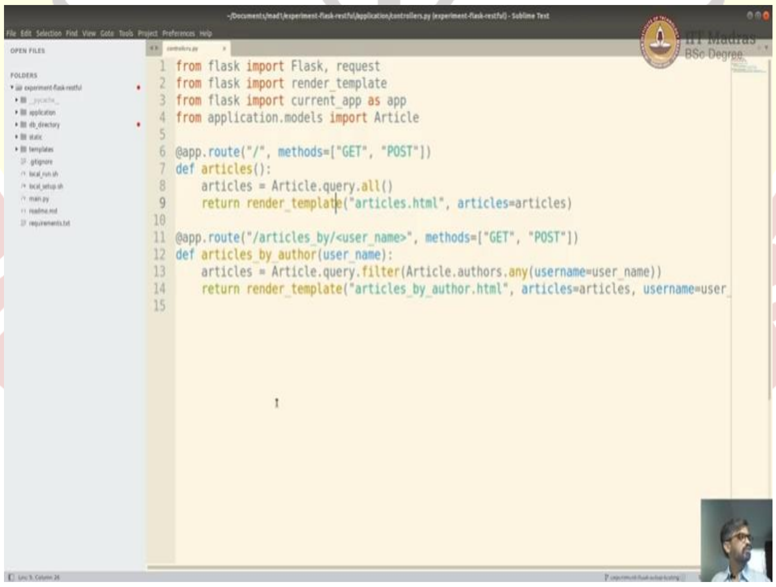
Code	Description	Links
200	Successfully updated.	No links
400	Bad request.	No links
404	Resource you requested to update not found	
500	Internal Server Error	

```
{
  "user_id": 80,
  "username": "ali",
  "email": "ali@example2.com"
}
```

```
{
  "error_code": "string",
  "error_message": "string"
}
```

Examples like that, it is important for you to go through this in detail before you start actual coding. Otherwise, you would end up doing or implementing something, that is completely different from what has been designed or documented, so start with always going through the documentation first and planning your implementation.

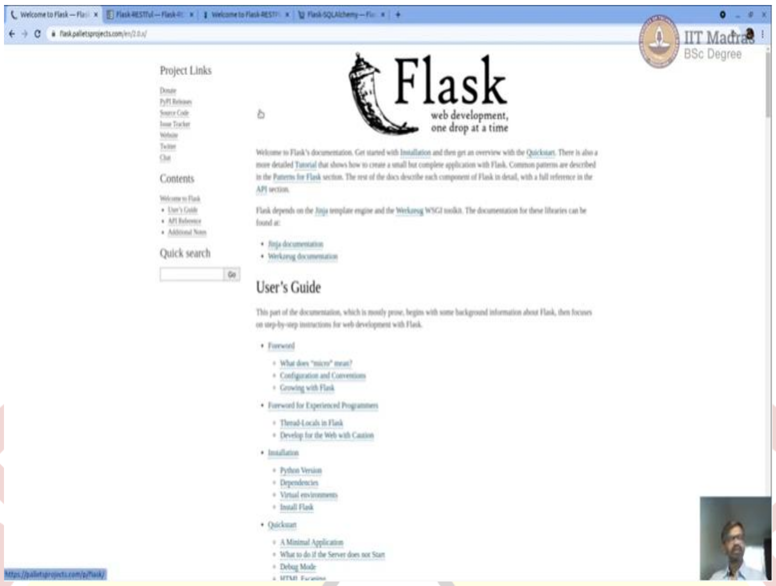
(Refer Slide Time: 3:05)



```python
from flask import Flask, request
from flask import render_template
from flask import current_app as app
from application.models import Article

@app.route("/", methods=["GET", "POST"])
def articles():
    articles = Article.query.all()
    return render_template("articles.html", articles=articles)

@app.route("/articles_by/<user_name>", methods=["GET", "POST"])
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, username=user
```

Now, we will start with the same project that we built previously, which has a structure of stranded your flask application, but we are going to just add an API part to this. Now, before we start implementing the APIs, we do not want to really implement every part of it. We want to use some existing framework, so it helps us to build the API.

(Refer Slide Time: 3:30)



Project Links

Donate
PyPI Releases
Source Code
Issue Tracker
Website
Twitter
Chat

Contents

Welcome to Flask
- User's Guide
- API Reference
- Additional Notes

Quick search

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.

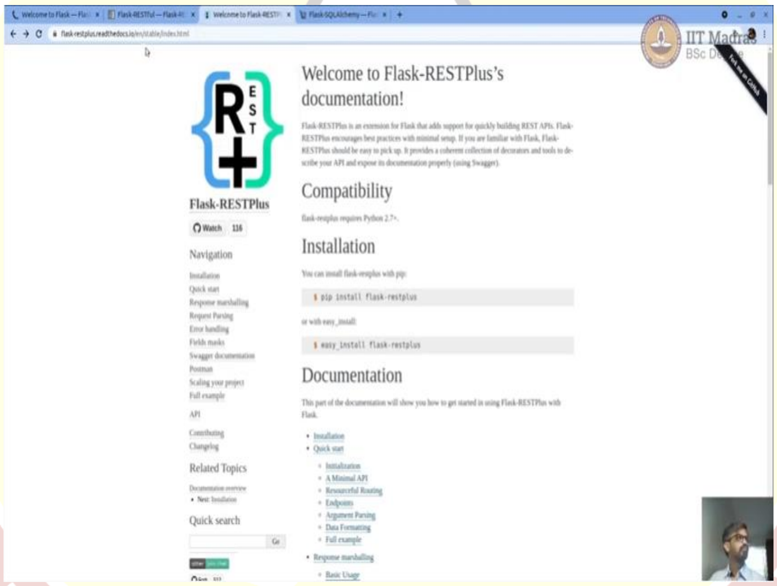
Flask depends on the Jinja template engine and the Werkzeug WSGI toolkit. The documentation for these libraries can be found at:

- Jinja documentation
- Werkzeug documentation

User's Guide

This part of the documentation, which is mostly prose, begins with some background information about Flask, then focuses on step-by-step instructions for web development with Flask.

- Foreword
  - What does "micro" mean?
  - Configuration and Conventions
  - Growing with Flask
- Foreword for Experienced Programmers
  - Thread-Locals in Flask
  - Develop for the Web with Caution
- Installation
  - Python Version
  - Dependencies
  - Virtual environments
  - Install Flask
- Quickstart
  - A Minimal Application
  - What to do if the Server does not Start
  - Debug Mode



Flask-RESTPlus

Navigation

Installation
Quick start
Response marshalling
Request Parsing
Error handling
Fields masks
Swagger documentation
Postman
Scaling your project
Full example

API

Contributing
Changelog

Related Topics

Documentation overview
- Next: Installation

Quick search

Welcome to Flask-RESTPlus's documentation!

Flask-RESTPlus is an extension for Flask that adds support for quickly building REST APIs. Flask-RESTPlus encourages best practices with minimal setup. If you are familiar with Flask, Flask-RESTPlus should be easy to pick up. It provides a coherent collection of decorators and tools to describe your API and expose its documentation properly (using Swagger).

Compatibility

flask-restplus requires Python 2.7+.

Installation

You can install flask-restplus with pip:

```
$ pip install flask-restplus
```

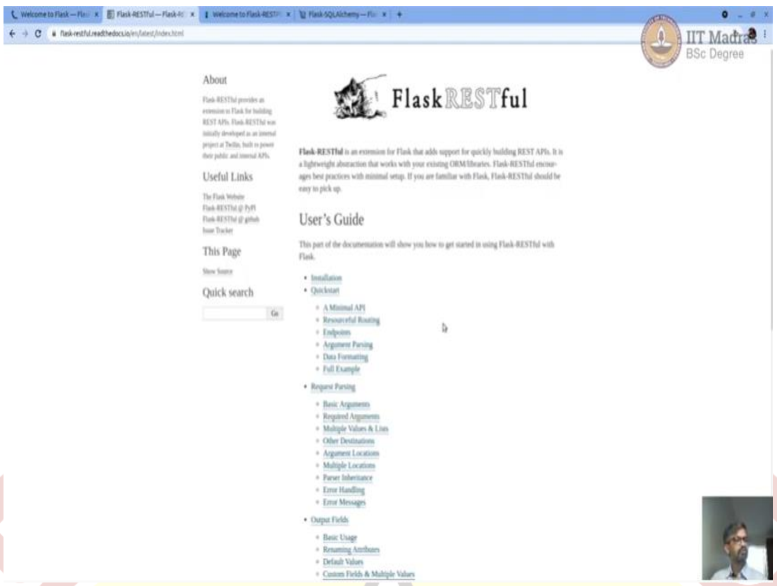
or with easy_install:

```
$ easy_install flask-restplus
```

Documentation

This part of the documentation will show you how to get started in using Flask-RESTPlus with Flask.

- Installation
- Quick start
  - Initialization
  - A Minimal API
  - Resourceful Routing
  - Endpoints
  - Argument Parsing
  - Data Formatting
  - Full example
- Response marshalling
  - Basic Usage


flask-restful.readthedocs.io/en/latest/index.html

About

Flask-RESTful provides an extension to Flask for building REST APIs. Flask-RESTful was initially developed as an internal project at Twilio, built to power their public and internal APIs.

Useful Links

The Flask Website
Flask-RESTful @ PyPI
Flask-RESTful @ github
Issue Tracker

This Page

Show Source

Quick search

Flask RESTful

Flask-RESTful is an extension for Flask that adds support for quickly building REST APIs. It is a lightweight abstraction that works with your existing ORM/libraries. Flask-RESTful encourages best practices with minimal setup. If you are familiar with Flask, Flask-RESTful should be easy to pick up.

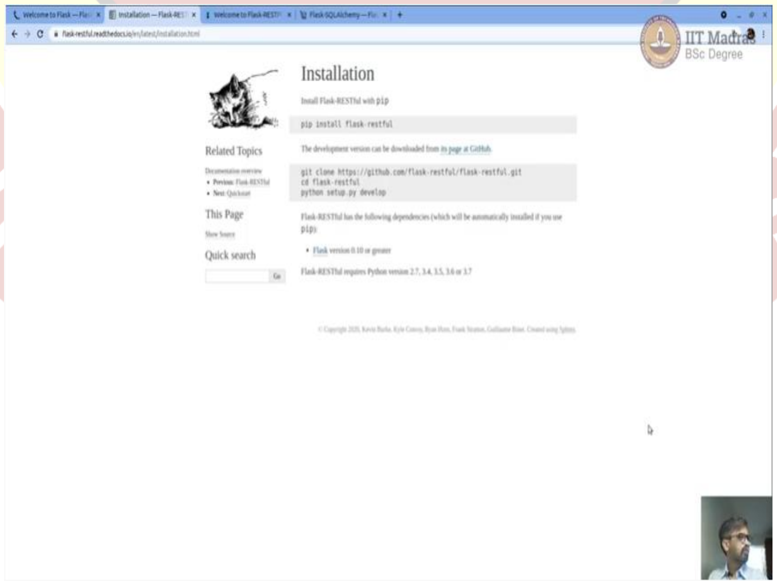
User's Guide

This part of the documentation will show you how to get started in using Flask-RESTful with Flask.

- Installation
- Quickstart
  - A Minimal API
  - Resourceful Routing
  - Endpoints
  - Argument Parsing
  - Data Formatting
  - Full Example
- Request Parsing
  - Basic Arguments
  - Required Arguments
  - Multiple Values & Lists
  - Other Destinations
  - Argument Locations
  - Multiple Locations
  - Parser Inheritance
  - Error Handling
  - Error Messages
- Output Fields
  - Basic Usage
  - Renaming Attributes
  - Default Values
  - Custom Fields & Multiple Values

There are quite a few frameworks that are available in the Flask world to implement APIs. One of them is Rest plus, which is popular, Restful, which is another one which is popular and also simple. In this case, you are going to use a restful, it should not really matter which one you are going to use, because the input and output you are going to be like standardized and what API's framework that you are going to use should not really matter. But in this case, we will go through one of the default ones or one of the standard ones. So it is easy for all of us to follow.

(Refer Slide Time: 4:16)


flask-restful.readthedocs.io/en/latest/installation.html

Installation

Install Flask-RESTful with pip

```
pip install flask-restful
```

The development version can be downloaded from its page at GitHub.

```
git clone https://github.com/flask-restful/flask-restful.git
cd flask-restful
python setup.py develop
```

Flask-RESTful has the following dependencies (which will be automatically installed if you use pip):

- Flask version 0.10 or greater

Flask-RESTful requires Python version 2.7, 3.4, 3.5, 3.6 or 3.7

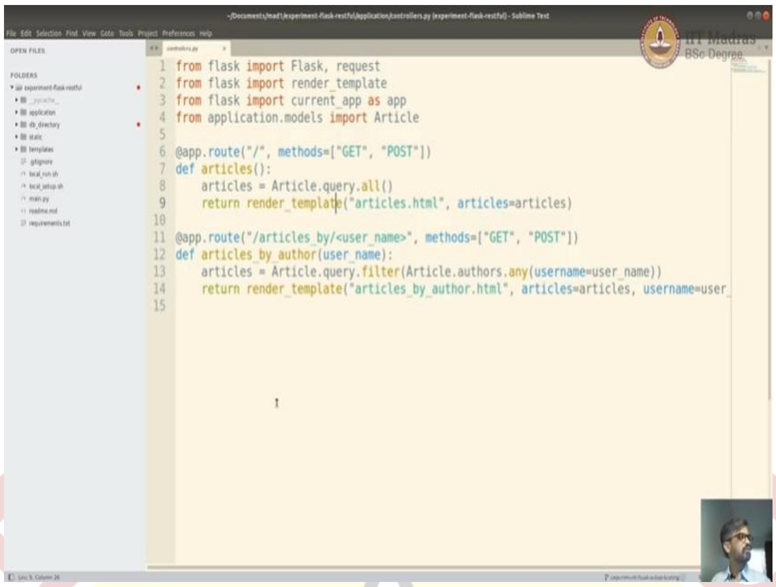
Related Topics

Documentation overview
- Previous: Flask-RESTful
- Next: Quickstart

This Page

Show Source

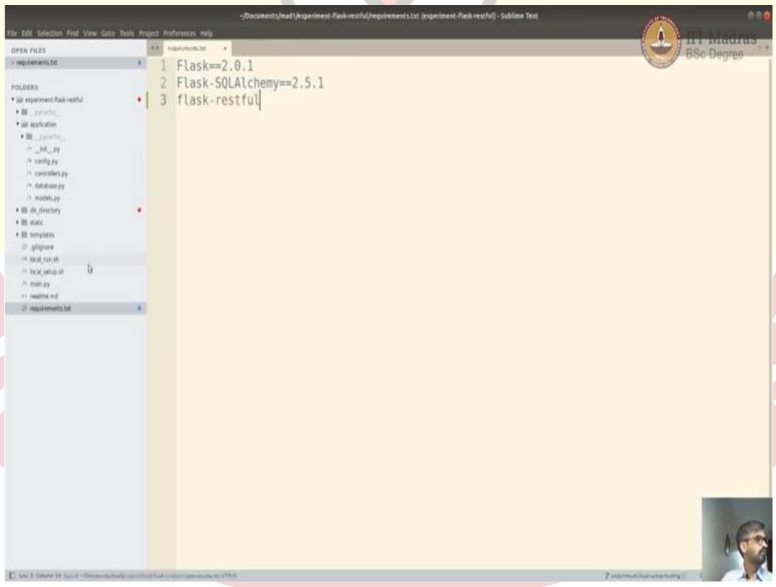
Quick search



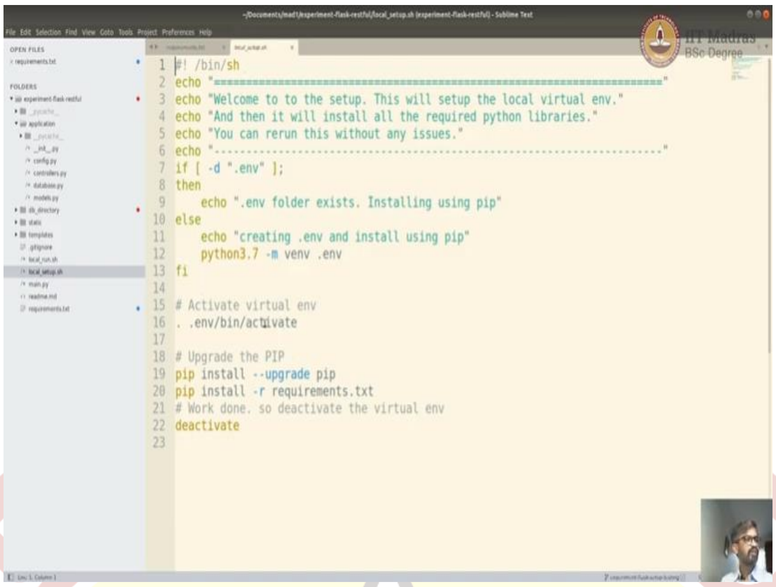
```python
from flask import Flask, request
from flask import render_template
from flask import current_app as app
from application.models import Article

@app.route("/", methods=["GET", "POST"])
def articles():
    articles = Article.query.all()
    return render_template("articles.html", articles=articles)

@app.route("/articles_by/<user_name>", methods=["GET", "POST"])
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, username=user_
```

Let us start with the flask restful, it starts with installation, let us install it and since we already have environment setup, here I have now actually removed the environment. So I am going to start up from the beginning. Before we do that, I am just going to upgrade my requirements.txt with another package called flask restful.

(Refer Slide Time: 4:37)



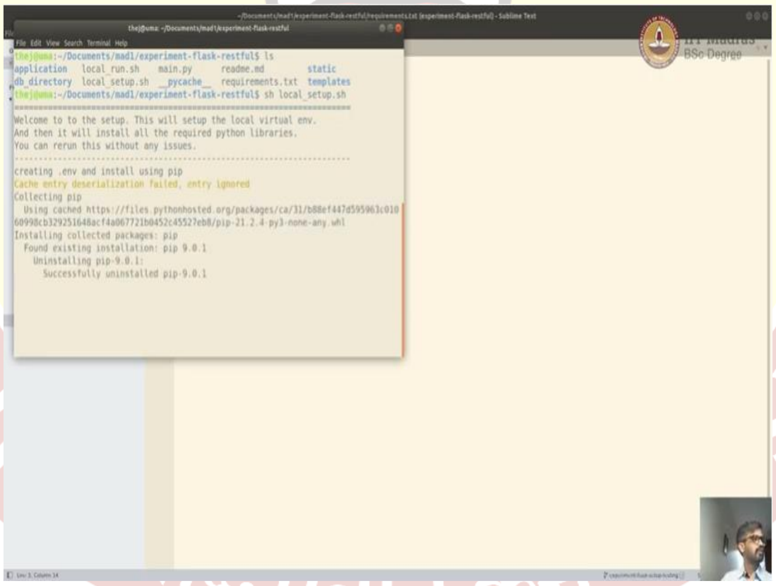
```
Flask==2.0.1
Flask-SQLAlchemy==2.5.1
flask-restful
```



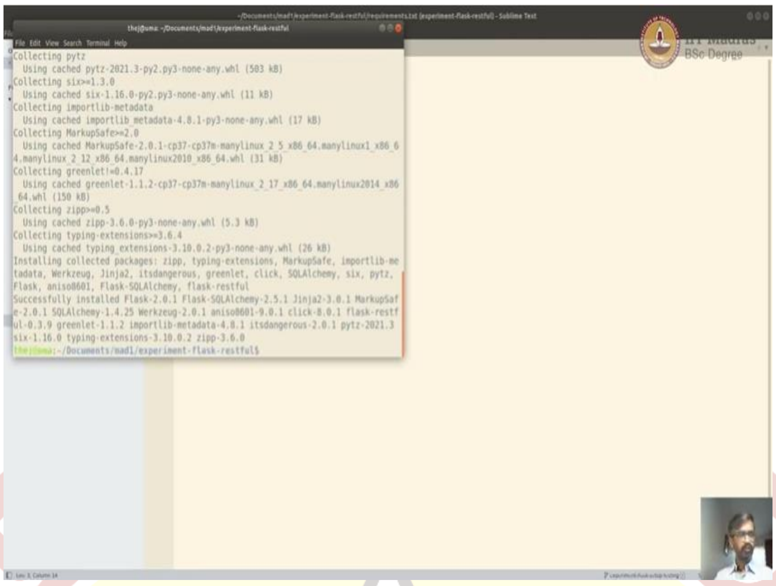
```sh
#! /bin/sh
echo "======================================================================"
echo "Welcome to to the setup. This will setup the local virtual env."
echo "And then it will install all the required python libraries."
echo "You can rerun this without any issues."
echo "----------------------------------------------------------------------"
if [ -d ".env" ];
then
    echo ".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
```

So, I am just going to add here then I am going to call my local setup, it is just going to create environment, activate it and install things in the requirements.txt, since we have added flask restful it is going to install that also.

(Refer Slide Time: 5:11)



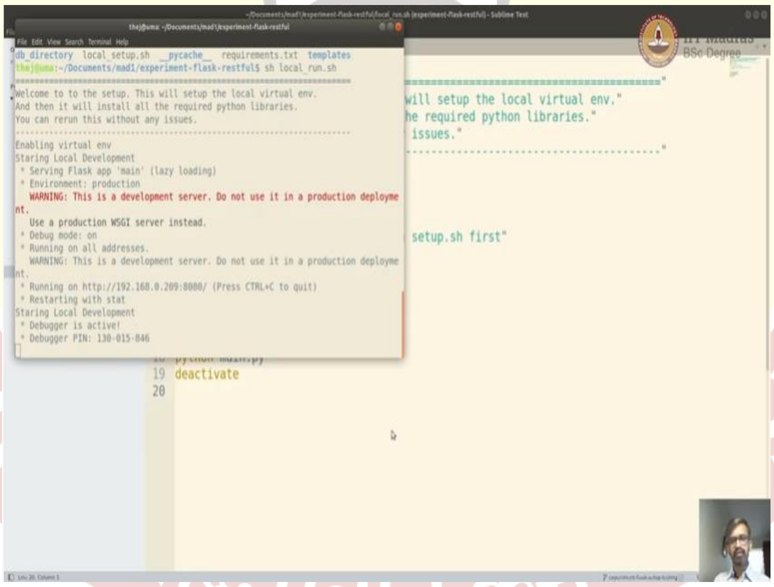
```
thej@uma:~/Documents/mad1/experiment-flask-restful$ ls
application   local_run.sh    main.py      readme.md          static
db_directory  local_setup.sh  __pycache__  requirements.txt   templates
thej@uma:~/Documents/mad1/experiment-flask-restful$ sh local_setup.sh
======================================================================
Welcome to to the setup. This will setup the local virtual env.
And then it will install all the required python libraries.
You can rerun this without any issues.
----------------------------------------------------------------------
creating .env and install using pip
Cache entry deserialization failed, entry ignored
Collecting pip
  Using cached https://files.pythonhosted.org/packages/ca/31/b88ef447d595963c010
60998cb329251648acf4a067721b0452c45527eb8/pip-21.2.4-py3-none-any.whl
Installing collected packages: pip
  Found existing installation: pip 9.0.1
    Uninstalling pip-9.0.1:
      Successfully uninstalled pip-9.0.1
```



```
Collecting pytz
  Using cached pytz-2021.3-py2.py3-none-any.whl (503 kB)
Collecting six>=1.3.0
  Using cached six-1.16.0-py2.py3-none-any.whl (11 kB)
Collecting importlib-metadata
  Using cached importlib_metadata-4.8.1-py3-none-any.whl (17 kB)
Collecting MarkupSafe>=2.0
  Using cached MarkupSafe-2.0.1-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_6
4.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (31 kB)
Collecting greenlet!=0.4.17
  Using cached greenlet-1.1.2-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl (150 kB)
Collecting zipp>=0.5
  Using cached zipp-3.6.0-py3-none-any.whl (5.3 kB)
Collecting typing-extensions>=3.6.4
  Using cached typing_extensions-3.10.0.2-py3-none-any.whl (26 kB)
Installing collected packages: zipp, typing-extensions, MarkupSafe, importlib-me
tadata, Werkzeug, Jinja2, itsdangerous, greenlet, click, SQLAlchemy, six, pytz,
Flask, aniso8601, Flask-SQLAlchemy, flask-restful
Successfully installed Flask-2.0.1 Flask-SQLAlchemy-2.5.1 Jinja2-3.0.1 MarkupSaf
e-2.0.1 SQLAlchemy-1.4.25 Werkzeug-2.0.1 aniso8601-9.0.1 click-8.0.1 flask-restf
ul-0.3.9 greenlet-1.1.2 importlib-metadata-4.8.1 itsdangerous-2.0.1 pytz-2021.3
six-1.16.0 typing-extensions-3.10.0.2 zipp-3.6.0
thej@uma:~/Documents/mad1/experiment-flask-restful$
```

Let us, let us start with that yeah. So I am going to do ls, I am going to do sh local_setup.sh, so it is going to create a .env folder for virtual environment and then going to install everything that is required. Done, yes, it is installed.

(Refer Slide Time: 5:47)

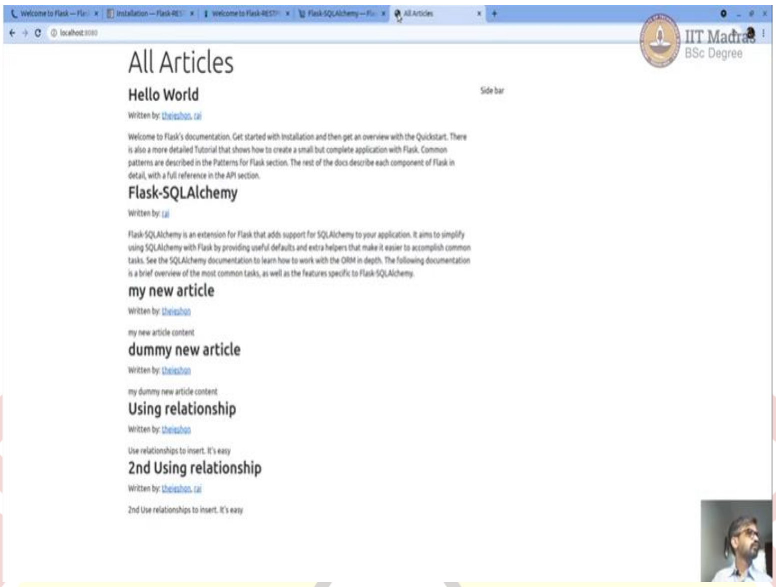


```
db_directory  local_setup.sh  __pycache__  requirements.txt  templates
thej@uma:~/Documents/mad1/experiment-flask-restful$ sh local_run.sh
======================================================================
Welcome to to the setup. This will setup the local virtual env.
And then it will install all the required python libraries.
You can rerun this without any issues.
----------------------------------------------------------------------
Enabling virtual env
Staring Local Development
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   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 deployme
nt.
 * Running on http://192.168.0.209:8080/ (Press CTRL+C to quit)
 * Restarting with stat
Staring Local Development
 * Debugger is active!
 * Debugger PIN: 130-015-846
```

```
deactivate
```

So now, if we start using local setup or local run, rather it will going to set up environment and as development and start the main.py, which is our main application. So, let us start and just to check whether everything is running, sh local_run.sh it is running on port 8080.

(Refer Slide Time: 5:57)


All Articles

Hello World

Written by: theieshon, rai

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.

Flask-SQLAlchemy

Written by: rai

Flask-SQLAlchemy is an extension for Flask that adds support for SQLAlchemy to your application. It aims to simplify using SQLAlchemy with Flask by providing useful defaults and extra helpers that make it easier to accomplish common tasks. See the SQLAlchemy documentation to learn how to work with the ORM in depth. The following documentation is a brief overview of the most common tasks, as well as the features specific to Flask-SQLAlchemy.

my new article

Written by: theieshon

my new article content

dummy new article

Written by: theieshon

my dummy new article content

Using relationship

Written by: theieshon

Use relationships to insert. It's easy

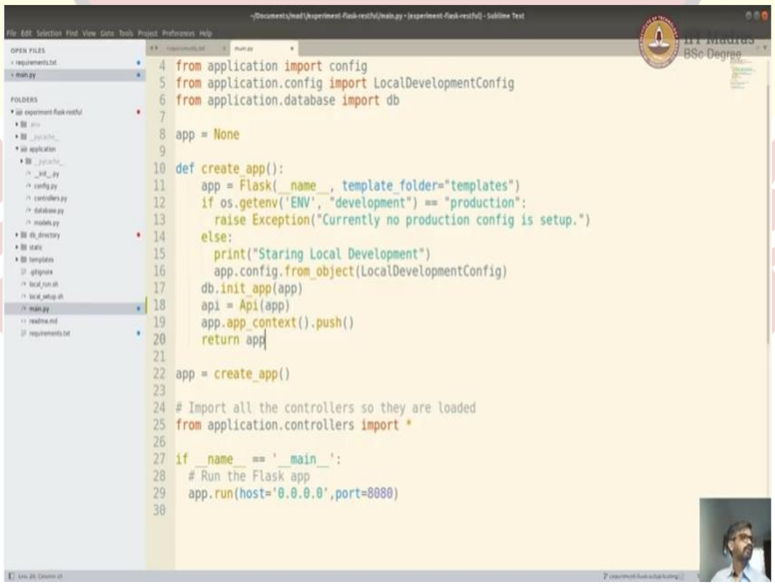
2nd Using relationship

Written by: theieshon, rai

2nd Use relationships to insert. It's easy

Let us just try it cool seems to be running. This is what we have done until now. So now we are going to add a rest APIs to this, rest APIs is form of controllers, so we are just going to add another module here to do all our controlling of rest related controllers, but before we do that we are going to initialize the flask restful framework as part of the App and then, configure it and then start with the rest of the things.

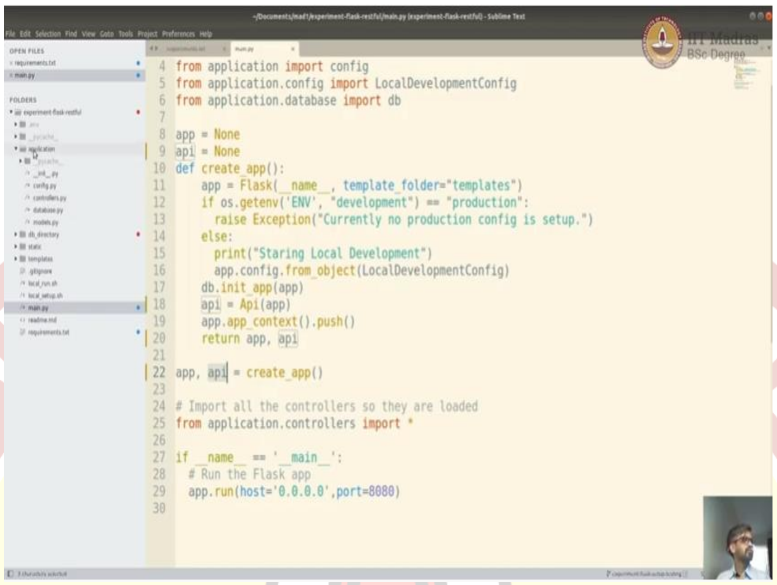
(Refer Slide Time: 7:07)


```
from application import config
from application.config import LocalDevelopmentConfig
from application.database import db

app = None

def create_app():
    app = Flask(__name__, template_folder="templates")
    if os.getenv('ENV', "development") == "production":
      raise Exception("Currently no production config is setup.")
    else:
      print("Staring Local Development")
      app.config.from_object(LocalDevelopmentConfig)
    db.init_app(app)
    api = Api(app)
    app.app_context().push()
    return app

app = create_app()

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

if __name__ == '__main__':
  # Run the Flask app
  app.run(host='0.0.0.0',port=8080)
```

~/Documents/mad1/experiment-flask-restful/main.py - (experiment-flask-restful) - Sublime Text
IIT Madras
BSc Degree

So, to configure it, let us import the flask restful and then init the app. To import, I am just going to copy paste, I am just going to import resource and API actually from the flask restful and then I am going to init the app with flask restful. That is done by just doing this, API equal to API of app.

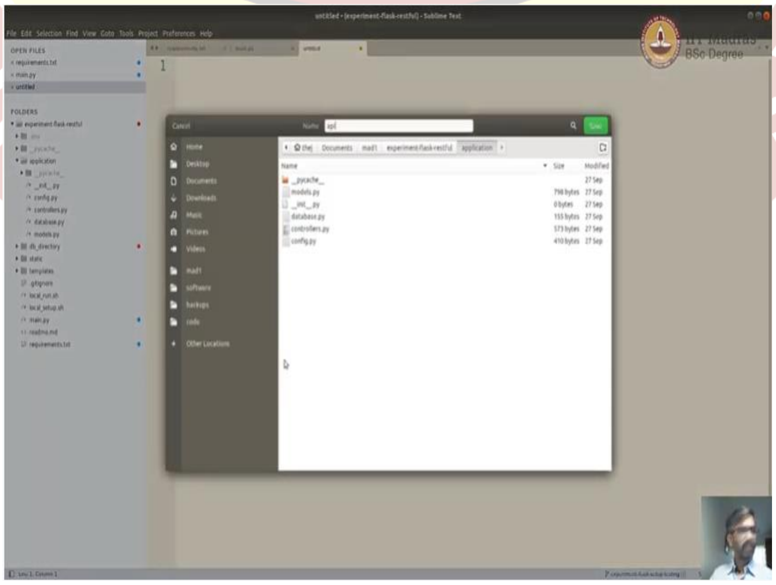
(Refer Slide Time: 7:46)



```
from application import config
from application.config import LocalDevelopmentConfig
from application.database import db

app = None
api = None
def create_app():
    app = Flask(__name__, template_folder="templates")
    if os.getenv('ENV', "development") == "production":
      raise Exception("Currently no production config is setup.")
    else:
      print("Staring Local Development")
      app.config.from_object(LocalDevelopmentConfig)
    db.init_app(app)
    api = Api(app)
    app.app_context().push()
    return app, api

app, api = create_app()

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

if __name__ == '__main__':
  # Run the Flask app
  app.run(host='0.0.0.0',port=8080)
```

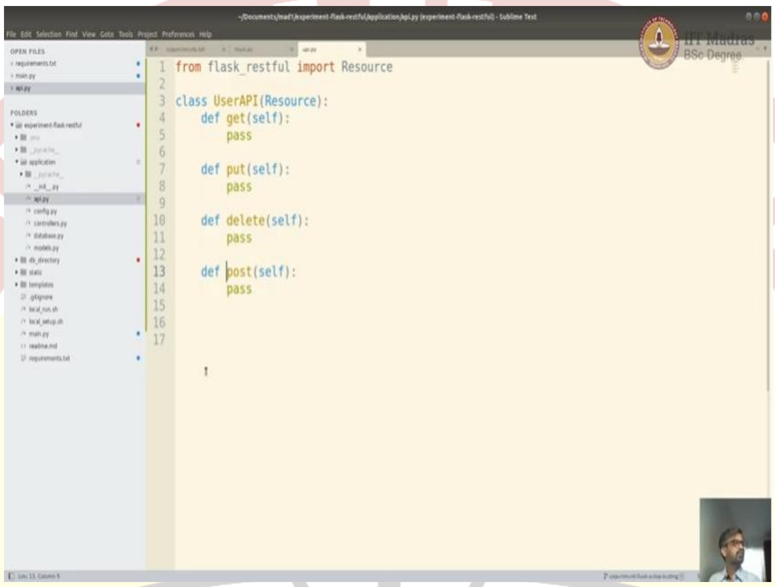
And then I just want to be it to be accessible, so I will just do initialize it to none and return both app and api. So here, so now we have imported the framework, Restful API, API and then initialized it with our app and then we just set the api object with that value and that is the minimum setup of the thing we have done that now we are going to add a module to write our restful controllers.

(Refer Slide Time: 7:55)



I am just going to call it api.py, nothing, complicated, we are just going to call it api.py. You could further divide into if you have, if your models or business logic is huge, you can even, create a folder called api within that you can call multiple like, for example, user_api, etc, etc. Here I think it is simple enough, so we are going to call it api.py in that we are going to write our API Controllers.

(Refer Slide Time: 10:19)

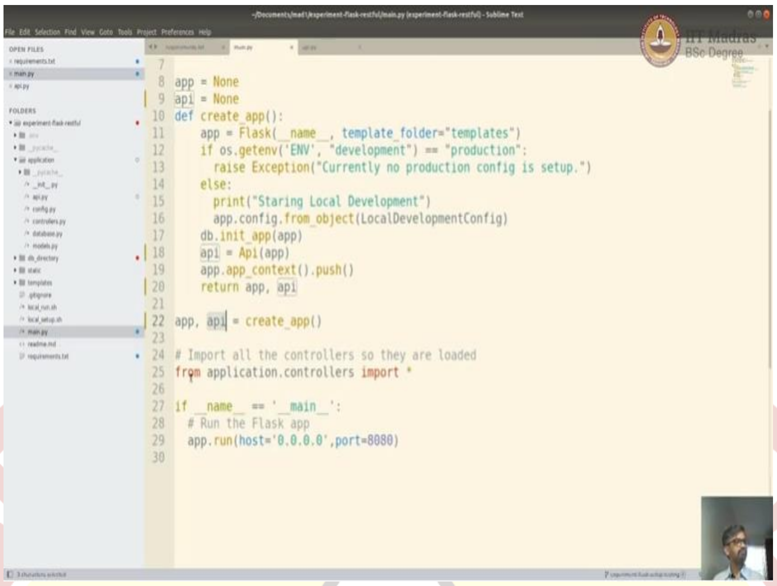


```
from flask_restful import Resource

class UserAPI(Resource):
    def get(self):
        pass

    def put(self):
        pass

    def delete(self):
        pass

    def post(self):
        pass
```

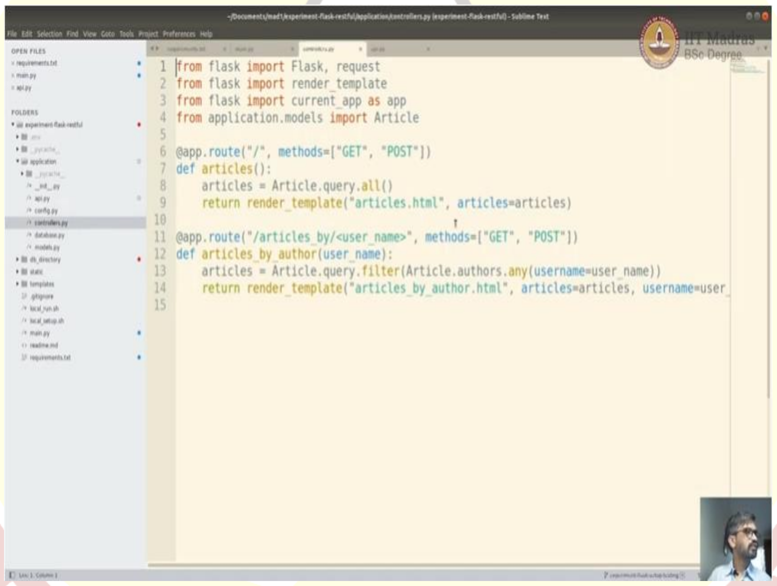
So, since we are going to write it for user class, I am just going to call that UserAPI. All the API classes in restful are going to extend a Resource class. So, let us start with that, so it will be this will be our class which will handle all the user related API's and it will have it will extend Resource class. Now this Resource class is from restful. So I am just going to extend that, import that rather here, so I am just importing the Resource and extending it now we need to do write get if we go back to here. We need to do get, put, delete and post. We need to write all four.

So, let us put the basic structure for it, so it is going to be a def function called get which will handle all the get requests. I am not going to implement anything; I am just going to call pass. Similarly, put, delete, post. So all basic structure of all the, request handlers specific to each method like when the method is get and the method is put and the method is delete and when the method is post. Currently, they are not doing anything they are just passing, nothing specific.

(Refer Slide Time: 10:33)



```
app = None
api = None
def create_app():
    app = Flask(__name__, template_folder="templates")
    if os.getenv('ENV', "development") == "production":
      raise Exception("Currently no production config is setup.")
    else:
      print("Staring Local Development")
      app.config.from_object(LocalDevelopmentConfig)
    db.init_app(app)
    api = Api(app)
    app.app_context().push()
    return app, api

app, api = create_app()

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

if __name__ == '__main__':
  # Run the Flask app
  app.run(host='0.0.0.0',port=8080)
```

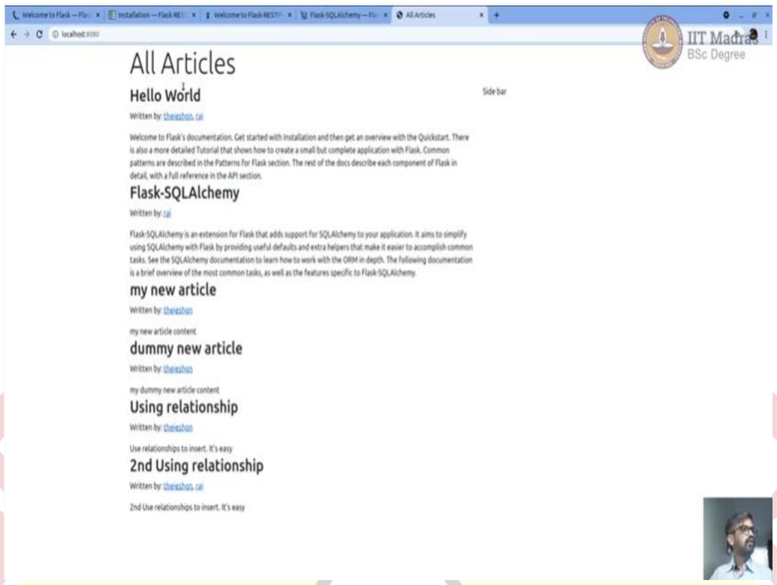


```
from flask import Flask, request
from flask import render_template
from flask import current_app as app
from application.models import Article

@app.route("/", methods=["GET", "POST"])
def articles():
    articles = Article.query.all()
    return render_template("articles.html", articles=articles)

@app.route("/articles_by/<user_name>", methods=["GET", "POST"])
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, username=user_
```

The next method, next thing is to map them to actually to URL handler. Let us go to our main.py. Here actually, we have defined the regular controllers, where you have done the mapping directly, for example, here is the mapping done so here. Since we have class 1, we can just directly define them here. Let us just add it by first we will just import it.

So, we are going to import the UserAPI from .api UserAPI, the class and then just going to map the class to our URL path. I am going to map it to a path called /, /api/user and it takes <string:username> as a thing.

(Refer Slide Time: 11:36)



All Articles

Hello World
Written by: theieshon, raj

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.

Flask-SQLAlchemy
Written by: raj

Flask-SQLAlchemy is an extension for Flask that adds support for SQLAlchemy to your application. It aims to simplify using SQLAlchemy with Flask by providing useful defaults and extra helpers that make it easier to accomplish common tasks. See the SQLAlchemy documentation to learn how to work with the ORM in depth. The following documentation is a brief overview of the most common tasks, as well as the features specific to Flask-SQLAlchemy.

my new article
Written by: theieshon

my new article content

dummy new article
Written by: theieshon

my dummy new article content

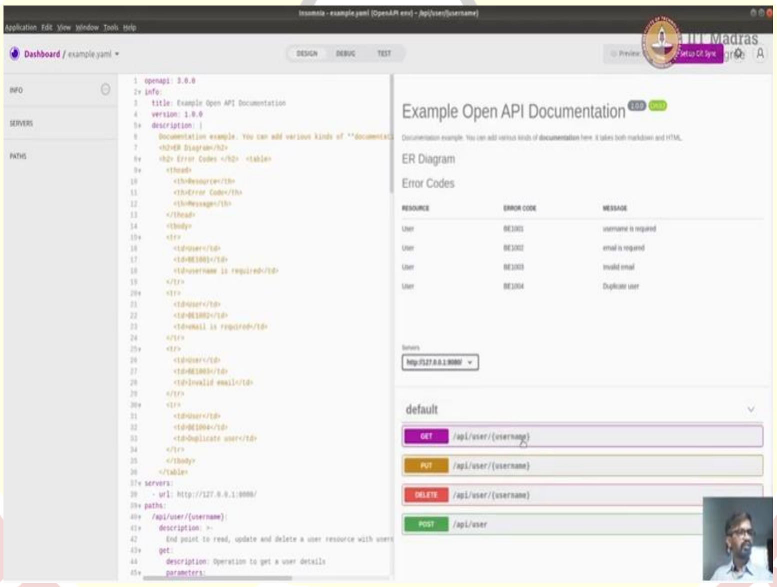
Using relationship
Written by: theieshon

Use relationships to insert. It's easy

2nd Using relationship
Written by: theieshon, raj

2nd Use relationships to insert. It's easy

Side bar



Insomnia - example.yaml (OpenAPI env) - /api/user/{username}

```
openapi: 3.0.0
info:
  title: Example Open API Documentation
  version: 1.0.0
  description: |
    Documentation example. You can add various kinds of **documentati
    <h2>ER Diagram</h2>
    <h2> Error Codes </h2>  <table>
      <thead>
        <th>Resource</th>
        <th>Error Code</th>
        <th>Message</th>
      </thead>
      <tbody>
      <tr>
        <td>User</td>
        <td>BE1001</td>
        <td>username is required</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1002</td>
        <td>email is required</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1003</td>
        <td>Invalid email</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1004</td>
        <td>Duplicate user</td>
      </tr>
      </tbody>
    </table>
servers:
  - url: http://127.0.0.1:8080/
paths:
  /api/user/{username}:
    description: >-
      End point to read, update and delete a user resource with usern
    get:
      description: Operation to get a user details
      parameters:
```

Example Open API Documentation

Documentation example. You can add various kinds of **documentation** here. It takes both markdown and HTML.

ER Diagram

Error Codes

RESOURCE	ERROR CODE	MESSAGE
User	BE1001	username is required
User	BE1002	email is required
User	BE1003	Invalid email
User	BE1004	Duplicate user

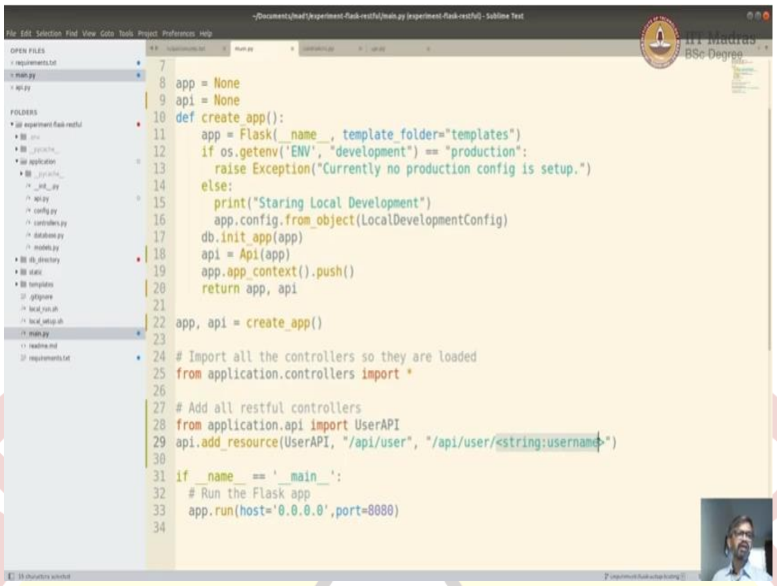
Servers: http://127.0.0.1:8080/

default

```
GET     /api/user/{username}
PUT     /api/user/{username}
DELETE  /api/user/{username}
POST    /api/user
```

Now this is done because in our documentation you could see that the URL parameter present is present for three of them at least get, put and delete. So that is why we have done the same thing and it is of the type string, hence that type here. A string now, there is one more which is post, which does not have a URL parameter, called username as part of its URL. So we just need to map this as well directly, because this is not going to handle it, so that can be done by just extra adding extra parameter here to this function.

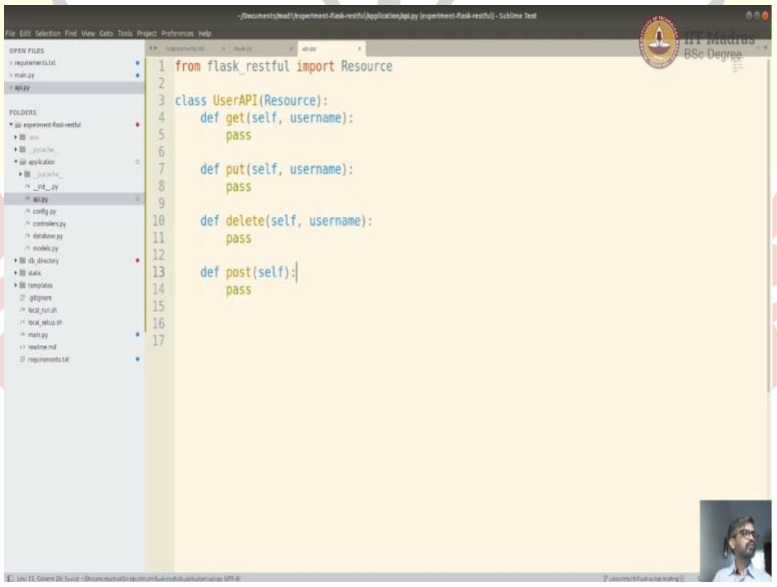
(Refer Slide Time: 12:48)



```
app = None
api = None
def create_app():
    app = Flask(__name__, template_folder="templates")
    if os.getenv('ENV', "development") == "production":
      raise Exception("Currently no production config is setup.")
    else:
      print("Staring Local Development")
      app.config.from_object(LocalDevelopmentConfig)
    db.init_app(app)
    api = Api(app)
    app.app_context().push()
    return app, api

app, api = create_app()

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

# Add all restful controllers
from application.api import UserAPI
api.add_resource(UserAPI, "/api/user", "/api/user/<string:username>")

if __name__ == '__main__':
  # Run the Flask app
  app.run(host='0.0.0.0',port=8080)
```

So, now this user API handles both these URL structures. This and this both the patterns are handled. So when the user access /api/user just goes to UserAPI, /user/username also goes to, if required we can write separate one, but we are writing it is easier to handle with the same thing.

(Refer Slide Time: 13:15)

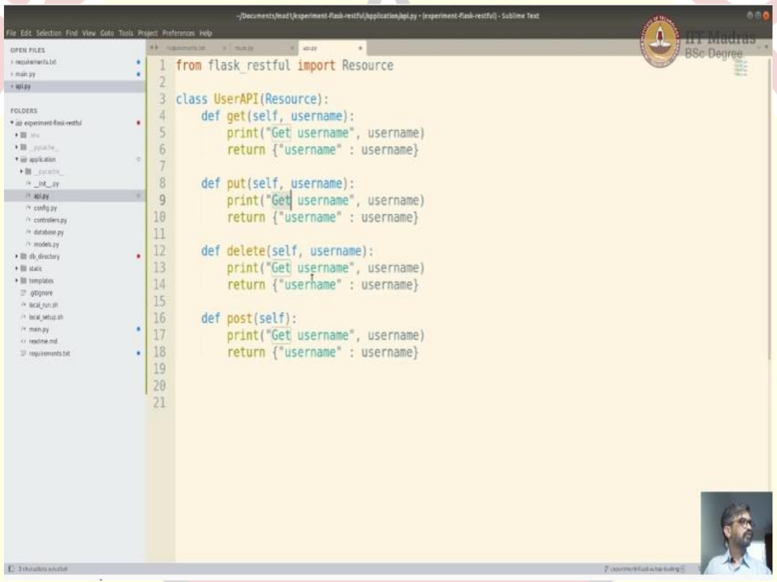


```
from flask_restful import Resource

class UserAPI(Resource):
    def get(self, username):
        pass

    def put(self, username):
        pass

    def delete(self, username):
        pass

    def post(self):
        pass
```

Now when you actually do this, where does this? How do you access this username name in our Functions, so it actually when you pass it as part of URL, it comes as part of the function. Input parameters, so your get will get user name, your put, will get user name, your delete will get user name, but your post will not get user name and hence this will be mapped to the post. The rest of them will be mapped to this.

So, this is again this structure, all of this is handled like this, because we have the documentation which prescribes it as to do it in that way, because in the get we have username, in the put we have username, in the delete we have username, in the post we do not have username as part of the thing, because we are going to create a new user as part of post, and we have done that. So now, basic structure is done. Now the part is, we can just start it and try to see whether we are getting actually values or not getting values, etc., etc.

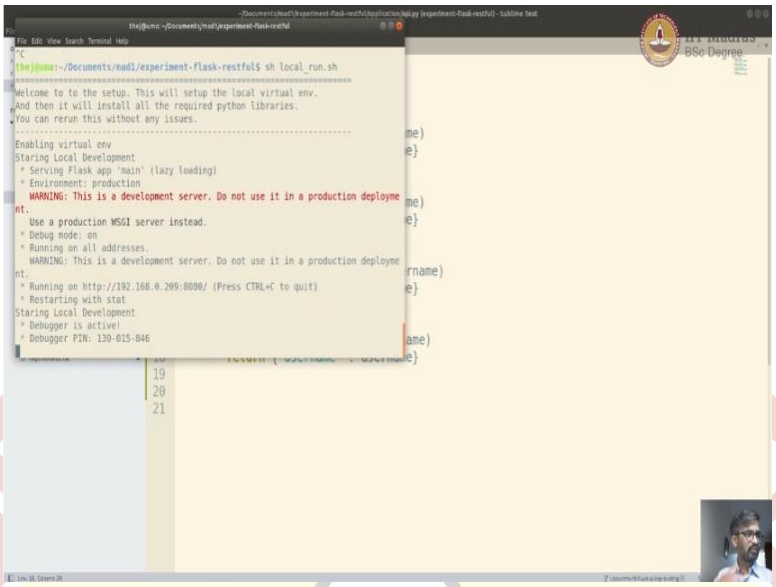
(Refer Slide Time: 15:14)



```
from flask_restful import Resource

class UserAPI(Resource):
    def get(self, username):
        print("Get username", username)
        return {"username" : username}

    def put(self, username):
        print("Get username", username)
        return {"username" : username}

    def delete(self, username):
        print("Get username", username)
        return {"username" : username}

    def post(self):
        print("Get username", username)
        return {"username" : username}
```

Let us just do at least get, let us not do that, let us just print username and return the same thing as a JSON. Just for us to try this is not the end result we want to see, but just to see whether everything is wired properly and all of them have been linked properly. After that we can get into main application development. So now what we have done, we have initialized the app we have created it. We have imported the user API, we have done the URL mapping and the same thing here.

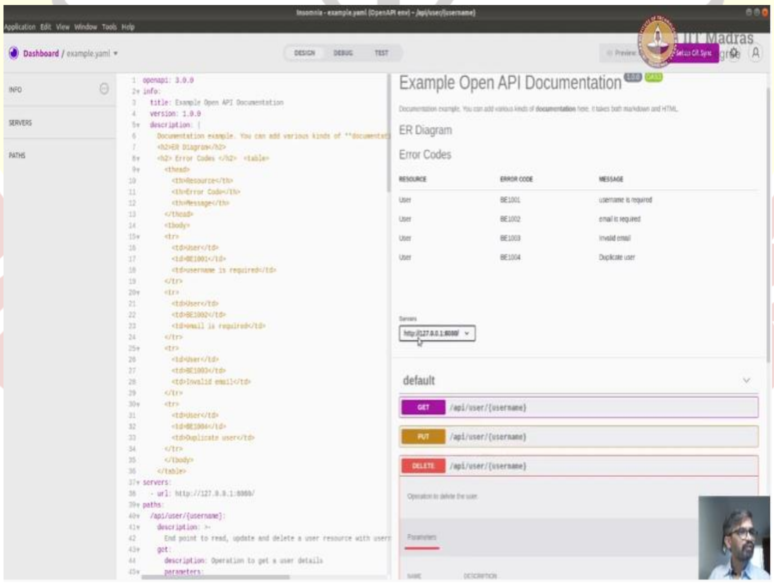
(Refer Slide Time: 15:34)



```
^C
thej@uma:~/Documents/mad1/experiment-flask-restful$ sh local_run.sh
======================================================================
Welcome to to the setup. This will setup the local virtual env.
And then it will install all the required python libraries.
You can rerun this without any issues.
----------------------------------------------------------------------
Enabling virtual env
Staring Local Development
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   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 deployme
nt.
 * Running on http://192.168.0.209:8080/ (Press CTRL+C to quit)
 * Restarting with stat
Staring Local Development
 * Debugger is active!
 * Debugger PIN: 130-015-846
```

So, let us not just do forget we will also get for other, we will also do for others. So that now it is easy to check it out whether all of them are working, so, let us make it get, put, delete, we are not doing anything, we are just printing, post just to try. So let us see whether our server has started it is running just want to restart it, just in case.

(Refer Slide Time: 15:48)



```
openapi: 3.0.0
info:
  title: Example Open API Documentation
  version: 1.0.0
  description: |
    Documentation example. You can add various kinds of **documentat
    <h2>ER Diagram</h2>
    <h2> Error Codes </h2>  <table>
      <thead>
        <th>Resource</th>
        <th>Error Code</th>
        <th>Message</th>
      </thead>
      <tbody>
      <tr>
        <td>User</td>
        <td>BE1001</td>
        <td>username is required</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1002</td>
        <td>email is required</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1003</td>
        <td>Invalid email</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1004</td>
        <td>Duplicate user</td>
      </tr>
      </tbody>
    </table>
servers:
  - url: http://127.0.0.1:8080/
paths:
  /api/user/{username}:
    description: >-
      End point to read, update and delete a user resource with usern
    get:
      description: Operation to get a user details
      parameters:
```

Example Open API Documentation

Documentation example. You can add various kinds of **documentation** here. It takes both markdown and HTML.

ER Diagram

Error Codes

RESOURCE	ERROR CODE	MESSAGE
User	BE1001	username is required
User	BE1002	email is required
User	BE1003	Invalid email
User	BE1004	Duplicate user

Servers: http://127.0.0.1:8080/

default

GET /api/user/{username}

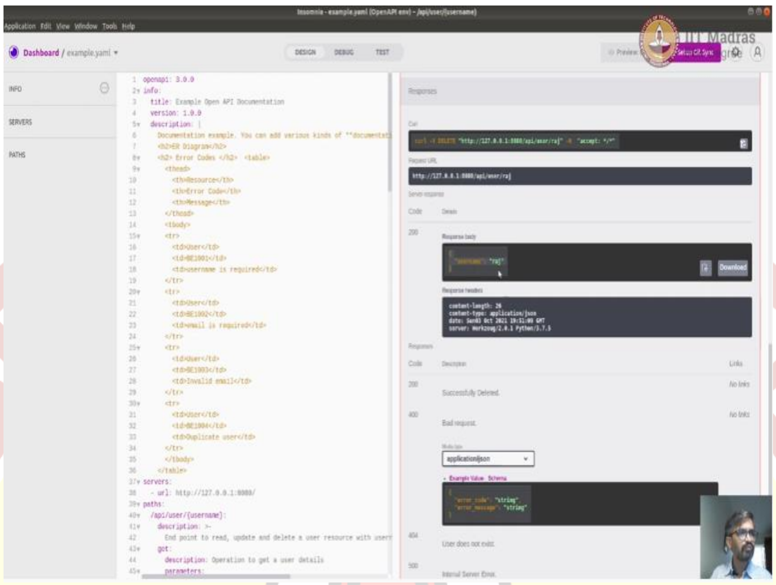
PUT /api/user/{username}

DELETE /api/user/{username}

Operation to delete the user

Now, let us go back to our documentation and go to delete and click on just make sure first, the URL is mapped well, it is 127.0.0.1:8080/api/user; okay, it is running on 8080 yeah, it is running on 8080 and it is localhost and /api/user, this maps to our mapping here /api/user/username; so everything seems to be correct. So, let us try delete you, can click on try it? You can enter the username as Raj click on execute. It is run, you can see here it is run and it is returning username as Raj.

(Refer Slide Time: 16:41)


Insomnia - example.yaml (OpenAPI env) - /api/user/{username}
Application Edit View Window Tools Help
Dashboard / example.yaml
DESIGN
DEBUG
TEST
INFO
SERVERS
PATHS
Preview
Setup Git Sync
Download

```
openapi: 3.0.0
info:
  title: Example Open API Documentation
  version: 1.0.0
  description: |
    Documentation example. You can add various kinds of **documentati
    <h2>ER Diagram</h2>
    <h2> Error Codes </h2>  <table>
      <thead>
        <th>Resource</th>
        <th>Error Code</th>
        <th>Message</th>
      </thead>
      <tbody>
      <tr>
        <td>User</td>
        <td>BE1001</td>
        <td>username is required</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1002</td>
        <td>email is required</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1003</td>
        <td>Invalid email</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1004</td>
        <td>Duplicate user</td>
      </tr>
      </tbody>
    </table>
servers:
  - url: http://127.0.0.1:8080/
paths:
  /api/user/{username}:
    description: >-
      End point to read, update and delete a user resource with userr
    get:
      description: Operation to get a user details
      parameters:
```

Responses

Curl

```
curl -X DELETE "http://127.0.0.1:8080/api/user/raj" -H "accept: */*"
```

Request URL

```
http://127.0.0.1:8080/api/user/raj
```

Server response

Code	Details
200	Response body

```
{
  "username": "raj"
}
```

Response headers

```
content-length: 26
content-type: application/json
date: Sun03 Oct 2021 19:51:09 GMT
server: Werkzeug/2.0.1 Python/3.7.5
```

Responses

Code	Description	Links
200	Successfully Deleted.	No links
400	Bad request.	No links
404	User does not exist.	
500	Internal Server Error.	

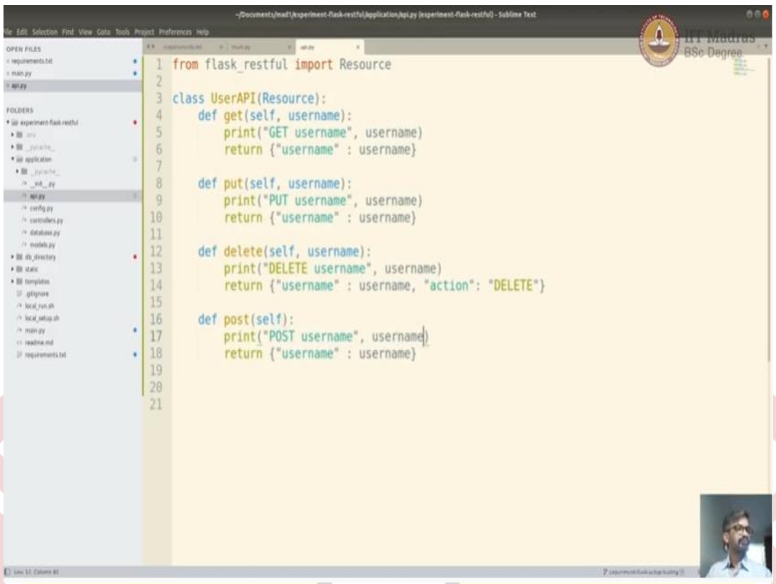
Media type: application/json

Example Value Schema

```
{
  "error_code": "string",
  "error_message": "string"
}
```

Is that what we were trying to do? Yes, we were trying to print Return username as Raj. That is correct as part of delete2, but here you can see in the print statement. It is called getting called “Delete/api/user/raj” and we are returning here this username called username and that is what we are getting as a response here. Now, just to make it clear you can also I mean this is for just our debugging purpose action. I can call it delete just to make sure that it is working, I am just going to restart just to make sure most of the time it should restart automatically.

(Refer Slide Time: 18:16)

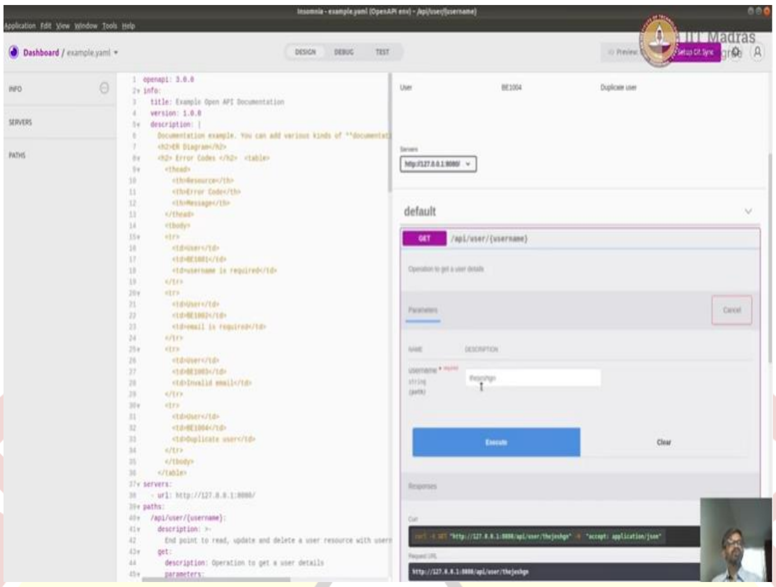


```python
from flask_restful import Resource

class UserAPI(Resource):
    def get(self, username):
        print("GET username", username)
        return {"username" : username}

    def put(self, username):
        print("PUT username", username)
        return {"username" : username}

    def delete(self, username):
        print("DELETE username", username)
        return {"username" : username, "action": "DELETE"}

    def post(self):
        print("POST username", username)
        return {"username" : username}
```

If not, you can just restart just to make sure it is running now I will try to execute once more, execute here, you can see. Username action is going to delete, so it is all been wired. Well, let us check just to post on other things as well. Just to make sure all of our functions are mapped, try it out execute. So, this is throwing error, because this does not have a username here. This is not, there is no username here, so it cannot print an username.

It is just post, let us save it. Let us restart it and try it again: okay seems to be wired, all post is wired, delete is wired, I am just going to cancel, this delete is wired, let us try out get execute, get is wired and then put let us just click and try it out and execute. Yeah, you are getting the result here, whatever you are passing here, so they all seem to have been wired. Now we like to actually See the documentation and do the real business work.

(Refer Slide Time: 19:30)



```
openapi: 3.0.0
info:
  title: Example Open API Documentation
  version: 1.0.0
  description: |
    Documentation example. You can add various kinds of **documentat
    <h2>ER Diagram</h2>
    <h2> Error Codes </h2>  <table>
      <thead>
        <th>Resource</th>
        <th>Error Code</th>
        <th>Message</th>
      </thead>
      <tbody>
      <tr>
        <td>User</td>
        <td>BE1001</td>
        <td>username is required</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1002</td>
        <td>email is required</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1003</td>
        <td>Invalid email</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1004</td>
        <td>Duplicate user</td>
      </tr>
      </tbody>
    </table>
servers:
  - url: http://127.0.0.1:8080/
paths:
  /api/user/{username}:
    description: >-
      End point to read, update and delete a user resource with usern
    get:
      description: Operation to get a user details
      parameters:
```

User	BE1004	Duplicate user

Servers: http://127.0.0.1:8080/

default

GET /api/user/{username}

Operation to get a user details

NAME	DESCRIPTION
username * required string (path)	thejeshgn

```
curl -X GET "http://127.0.0.1:8080/api/user/thejeshgn" -H "accept: application/json"
```

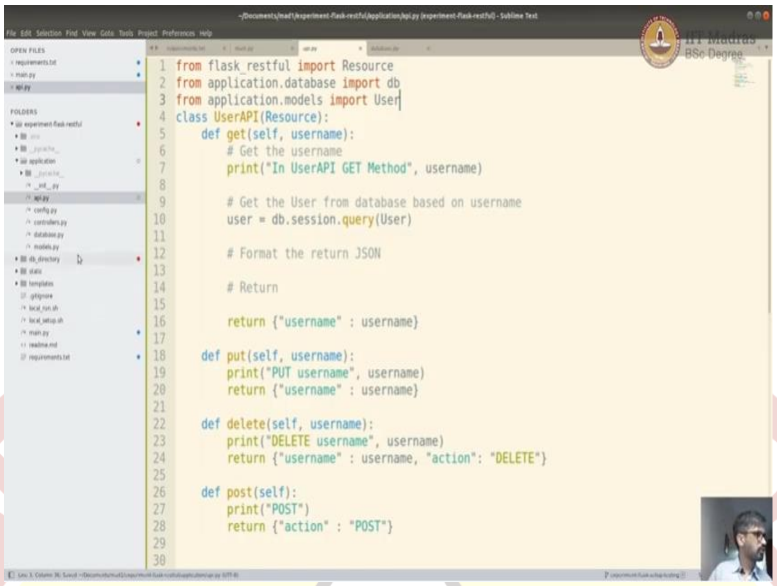
Request URL

```
http://127.0.0.1:8080/api/user/thejeshgn
```

So, what is get is actually generally whatever is the username passed, get that user from the database and return the results? If everything works well, you have to get a user id, username and email. If there is any error, we should send an error message, or if there is a resource not formed, we have to written 404. If internal server ratio we have to give 500, we have a standard, way of doing errors, which is like error, code and error message. I am just going to use the same thing for get as well in case of any error, but in case of get there is not much validation you just either it is not found or it is found. So we do not really need error message here.

So let us try and implement get so what we are getting here is username. Now, what are the steps involved? Get the username, ok and get the user from database based on username format. The return JSON return, now format the return JSON also has to be based on user. If there is no user found, we have to send 404.

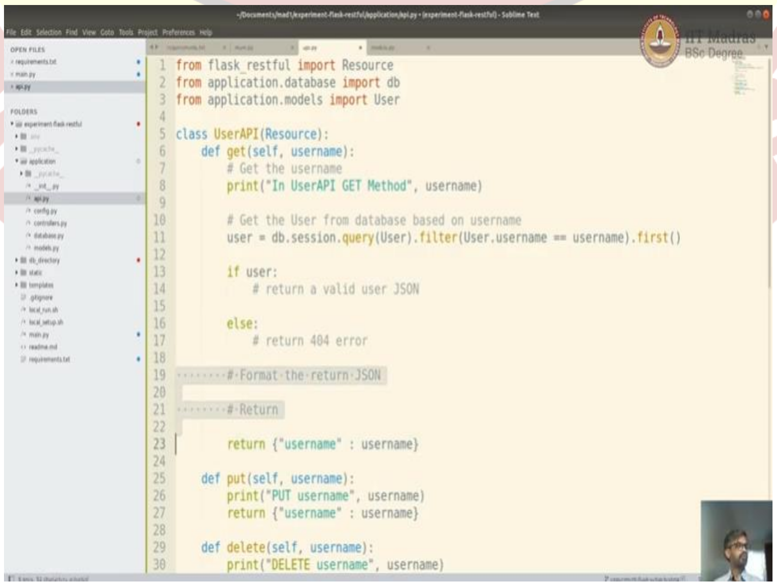
(Refer Slide Time: 22:52)



```
from flask_restful import Resource
from application.database import db
from application.models import User
class UserAPI(Resource):
    def get(self, username):
        # Get the username
        print("In UserAPI GET Method", username)

        # Get the User from database based on username
        user = db.session.query(User)

        # Format the return JSON

        # Return

        return {"username" : username}

    def put(self, username):
        print("PUT username", username)
        return {"username" : username}

    def delete(self, username):
        print("DELETE username", username)
        return {"username" : username, "action": "DELETE"}

    def post(self):
        print("POST")
        return {"action" : "POST"}
```

Let us see that once we reach that point, first get the username. We are already getting the username. We do not need to do much, I am Just going to print it. Just to make sure that we are getting actually in user API, get method and I am just going to print username. So I am just this is just for debugging in case, if you are trying to see why you are not getting something now the next is get username from Database based on the username. We have done this before we need to just import the db and run db.session.filter. So, let us do that using SQLAlchemy from application dot database import db.

(Refer Slide Time: 24:40)



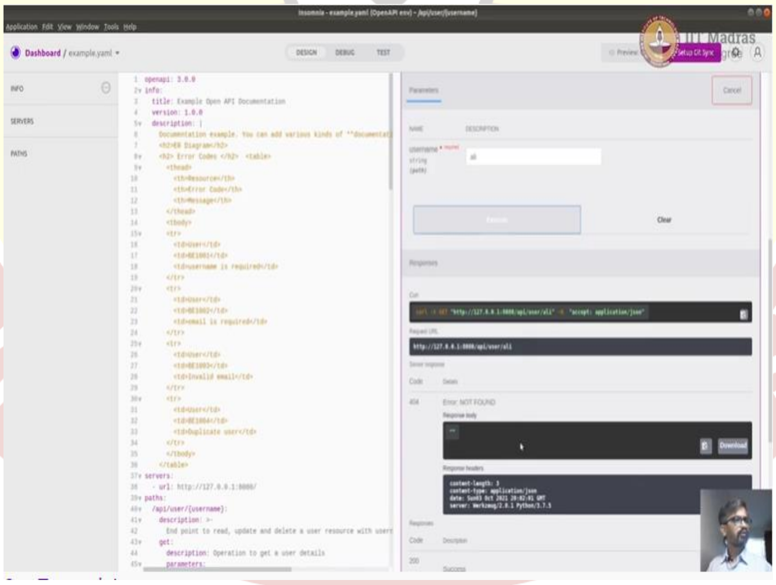
```
from flask_restful import Resource
from application.database import db
from application.models import User

class UserAPI(Resource):
    def get(self, username):
        # Get the username
        print("In UserAPI GET Method", username)

        # Get the User from database based on username
        user = db.session.query(User).filter(User.username == username).first()

        if user:
            # return a valid user JSON

        else:
            # return 404 error

        # Format the return JSON

        # Return

        return {"username" : username}

    def put(self, username):
        print("PUT username", username)
        return {"username" : username}

    def delete(self, username):
        print("DELETE username", username)
```

So now we have application.database. We are Importing this DB, now we can do user =db.session.query(User), query the user model. So if you are acquiring the user model, we have to import it so from application.models, Import user. So I am just going to make sure that model name is correct. Yes, it is user, now we are acquiring the user and we need to filter it.

Let us do that filtering, it is just dot filter we are going to filter it based on the user.username that is user.username == username, whatever we are passing and then get the first one. Now it if the, if the username is actually sent correctly, it should return the correct username. If it is not sent properly, then it should result a wrong user.

So, we can just do. I mean if the you, if there is a wrong username or the username, does not exist in the database, this user will be null if it is null, we have to send 404, not found, because that user is not found error. So we can just check for that condition if user return the, I am just going to write the comment first, return: a valid user JSON else, return 404 error. So that is what we are going to do now.

(Refer Slide Time: 27:18)


Insomnia - example.yaml (OpenAPI env) - /api/user/{username}
Application File View Window Tools Help
Dashboard / example.yaml
DESIGN
DEBUG
TEST
INFO
SERVERS
PATHS
Parameters
Cancel
Clear
Download

```
openapi: 3.0.0
info:
  title: Example Open API Documentation
  version: 1.0.0
  description: |
    Documentation example. You can add various kinds of **documentat
    <h2>ER Diagram</h2>
    <h2> Error Codes </h2>  <table>
      <thead>
        <th>Resource</th>
        <th>Error Code</th>
        <th>Message</th>
      </thead>
      <tbody>
      <tr>
        <td>User</td>
        <td>BE1001</td>
        <td>username is required</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1002</td>
        <td>email is required</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1003</td>
        <td>Invalid email</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1004</td>
        <td>Duplicate user</td>
      </tr>
      </tbody>
    </table>
servers:
  - url: http://127.0.0.1:8080/
paths:
  /api/user/{username}:
    description: >-
      End point to read, update and delete a user resource with usern
    get:
      description: Operation to get a user details
      parameters:
```

```
curl -X GET "http://127.0.0.1:8080/api/user/ali" -H "accept: application/json"
```

Request URL

```
http://127.0.0.1:8080/api/user/ali
```

Server response

Code	Details
404	Error: NOT FOUND

Response body

```
""
```

Response headers

```
content-length: 3
content-type: application/json
date: Sun03 Oct 2021 20:02:01 GMT
server: Werkzeug/2.0.1 Python/3.7.5
```

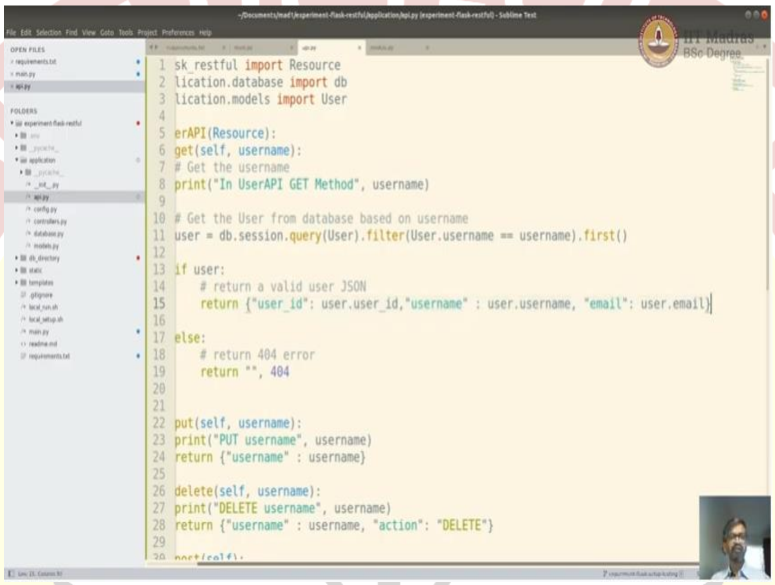
Responses

Code	Description
200	Success

So, here or just going to return, 404 we will do how, we will see how to do it better. So usually you can return the response and the response code as parameters in for this function. So that is why this is written like that. Then, let us see, while returning the valid JSON, the format should be as defined. Here it is user id, username and email. We can pull it out from, pull it out from our user thing.

So, user object or user model, so we can call user.username, just making sure the columns are correct, user id, username, email, ok seems to be correct: let us try it out. Okay, I am going back here, make thejeshgn as a thing, so this has to change execute. You can see user id, username and email has changed. Now if I give some random thing, let me open the database. We have only two user. Let us give ali as a username and execute you found that you are not getting any response but you are getting not found as a thing.

(Refer Slide Time: 28:18)



```
sk_restful import Resource
lication.database import db
lication.models import User

erAPI(Resource):
get(self, username):
# Get the username
print("In UserAPI GET Method", username)

# Get the User from database based on username
user = db.session.query(User).filter(User.username == username).first()

if user:
    # return a valid user JSON
    return {"user_id": user.user_id,"username" : user.username, "email": user.email}

else:
    # return 404 error
    return "", 404


put(self, username):
print("PUT username", username)
return {"username" : username}

delete(self, username):
print("DELETE username", username)
return {"username" : username, "action": "DELETE"}

post(self):
```

But here response is empty, I do not want to get empty JSON. I just want no body, so you can just create an empty string, let us hit that start again and click on that execute. So, nothing is returned and not found. You can probably return none too, let us see if it can return none, not sure about this, but we can try, yeah. I think you can return null, but empty string is a much better option if you are just doing bare bone like that. Okay, so this is how you do it, but this is not great, because we are trying to create JSON on the fly and we really cannot control.

(Refer Slide Time: 28:28)

Project Links

Donate
PyPI Releases
Source Code
Issue Tracker
Website
Twitter
Chat

Contents

Welcome to Flask
- User's Guide
- API Reference
- Additional Notes

Quick search

Flask
web development, one drop at a time

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.

Flask depends on the Jinja template engine and the Werkzeug WSGI toolkit. The documentation for these libraries can be found at:

- Jinja documentation
- Werkzeug documentation

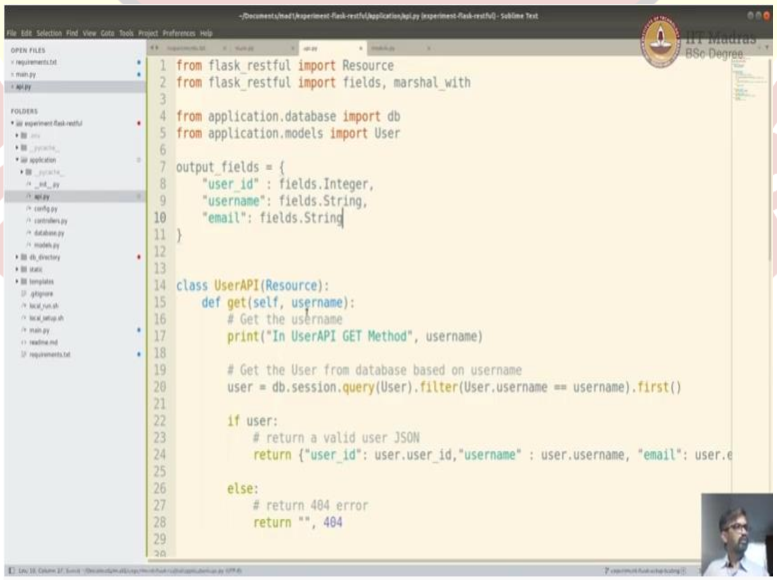
User's Guide

This part of the documentation, which is mostly prose, begins with some background information about Flask, then focuses on step-by-step instructions for web development with Flask.

- Foreword
  - What does "micro" mean?
  - Configuration and Conventions
  - Growing with Flask
- Foreword for Experienced Programmers
  - Thread-Locals in Flask
  - Develop for the Web with Caution
- Installation
  - Python Version
  - Dependencies
  - Virtual environments
  - Install Flask
- Quickstart
  - A Minimal Application
  - What to do if the Server does not Start
  - Debug Mode

Formatting what if there is a date and we wanted to format the date in certain way or we did not want to include certain columns in etc, etc. So for that we use something called marshalling to marshal the output so that it is easier for us to format this return JSON in a very standardized way. So you can go to, Flask rest full and check for marshalling, not here flask restful output field here, so you can send something called Marshall with or you can add a notation Marshall.

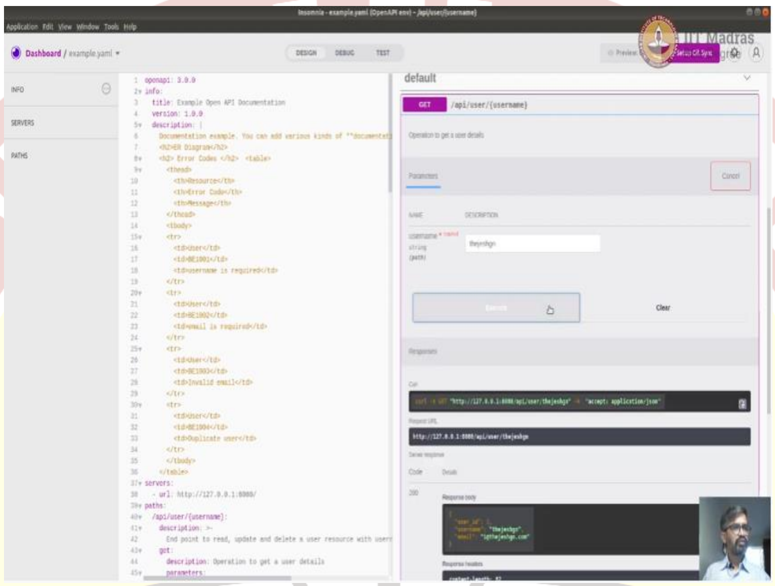
(Refer Slide Time: 30:26)



```python
from flask_restful import Resource
from flask_restful import fields, marshal_with

from application.database import db
from application.models import User

output_fields = {
    "user_id" : fields.Integer,
    "username": fields.String,
    "email": fields.String
}


class UserAPI(Resource):
    def get(self, username):
        # Get the username
        print("In UserAPI GET Method", username)

        # Get the User from database based on username
        user = db.session.query(User).filter(User.username == username).first()

        if user:
            # return a valid user JSON
            return {"user_id": user.user_id,"username" : user.username, "email": user.e

        else:
            # return 404 error
            return "", 404
```

So, you actually define the fields which you wanted to be part of the output, and then use that, let us define our output fields first, if I let us call our output fields, I wanted to have user_id of type String. But to do that we need to import fields from flask restful. I am just going to do that. Sorry, I am going to import that flask restful import fields and Marshall with so this fields dot. So we want the user id to be integer. So we are going to call it fields. Integer is the type and then similarly I want to user name. I want that to be string. Similarly, I want email also to be string.

(Refer Slide Time: 31:11)



```
openapi: 3.0.0
info:
  title: Example Open API Documentation
  version: 1.0.0
  description: |
    Documentation example. You can add various kinds of **documentati
    <h2>ER Diagram</h2>
    <h2> Error Codes </h2>  <table>
      <thead>
        <th>Resource</th>
        <th>Error Code</th>
        <th>Message</th>
      </thead>
      <tbody>
      <tr>
        <td>User</td>
        <td>BE1001</td>
        <td>username is required</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1002</td>
        <td>email is required</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1003</td>
        <td>Invalid email</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1004</td>
        <td>Duplicate user</td>
      </tr>
      </tbody>
    </table>
servers:
  - url: http://127.0.0.1:8080/
paths:
  /api/user/{username}:
    description: >-
      End point to read, update and delete a user resource with userr
    get:
      description: Operation to get a user details
      parameters:
```

GET /api/user/{username}

Operation to get a user details

username * required string (path): thejeshgn

```
curl -X GET "http://127.0.0.1:8080/api/user/thejeshgn" -H "accept: application/json"
```

Request URL

```
http://127.0.0.1:8080/api/user/thejeshgn
```

Code 200

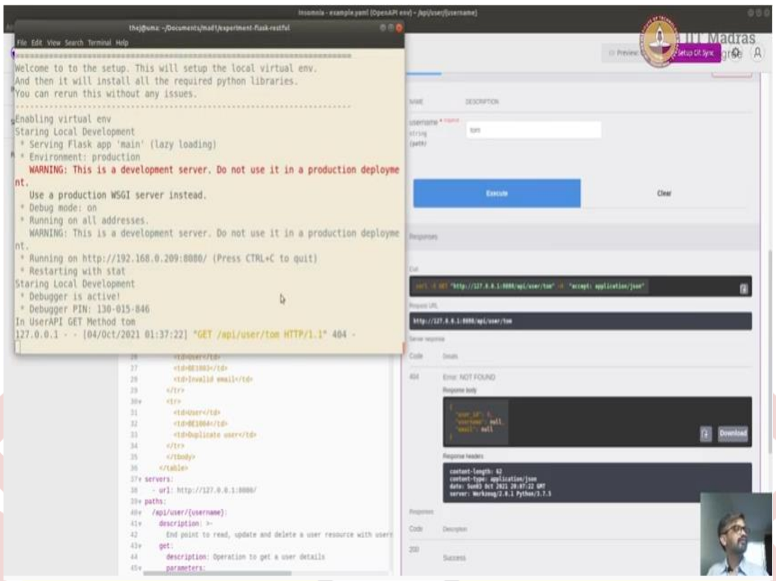
Response body

```
{
  "user_id": 1,
  "username": "thejeshgn",
  "email": "i@thejeshgn.com"
}
```

Now, you can just tag or annotate this function, with Marshall with and pass on, the name Of the fields that we created output fields, so that the response will be marshalled with this format like this format, so, instead of returning actually whole JSON, you can just return user and that should actually return you, this formatted output. Let us see that that is working, ok, let us go here and thejeshgn, click on execute. You can see that it is working.

Now, why this is useful is for example, if you do not want this to be there in your output, you can just delete it here and you do not have to do anything anywhere. You can just keep this like and as it is, and then you can, let me just restart the server and then, if you call execute you can see that that is vanished. You did not have to do much. You can add many other things here, you can do formatting in certain ways like, for example, if you are doing price or cost you can add, rs or, like commas, for your bigger numbers and stuff like that it makes it very useful to format your output and easy also.

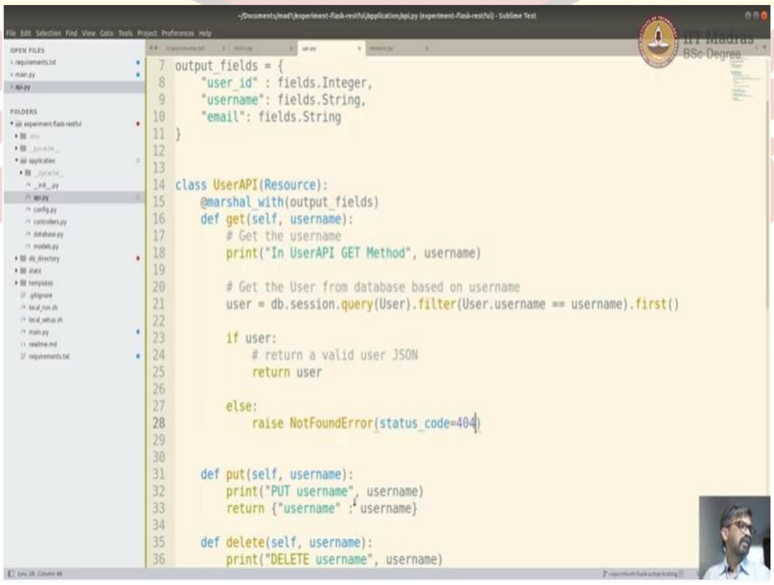
(Refer Slide Time: 32:32)



```
================================================================================
Welcome to to the setup. This will setup the local virtual env.
And then it will install all the required python libraries.
You can rerun this without any issues.
--------------------------------------------------------------------------------
Enabling virtual env
Staring Local Development
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   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 deployme
nt.
 * Running on http://192.168.0.209:8080/ (Press CTRL+C to quit)
 * Restarting with stat
Staring Local Development
 * Debugger is active!
 * Debugger PIN: 130-015-846
In UserAPI GET Method tom
127.0.0.1 - - [04/Oct/2021 01:37:22] "GET /api/user/tom HTTP/1.1" 404 -
```

Now, this is okay, but how will it react to this? So, let us see now that we have done it, we have to modify it, but I just wanted to show it throws the error I am just going to do tt or tom, so it is obviously not found it is still doing not found, but it is still trying to format, the body or whatever in this form way and that is why it is throwing this error. Anyway I think we also have to handle the error in a better way. We have to throw an exception to make it, make it easy on uniform way of handling errors, because in future we will handle errors as an error message in the rest of in the rest of the doc API’s.

(Refer Slide Time: 33:51)

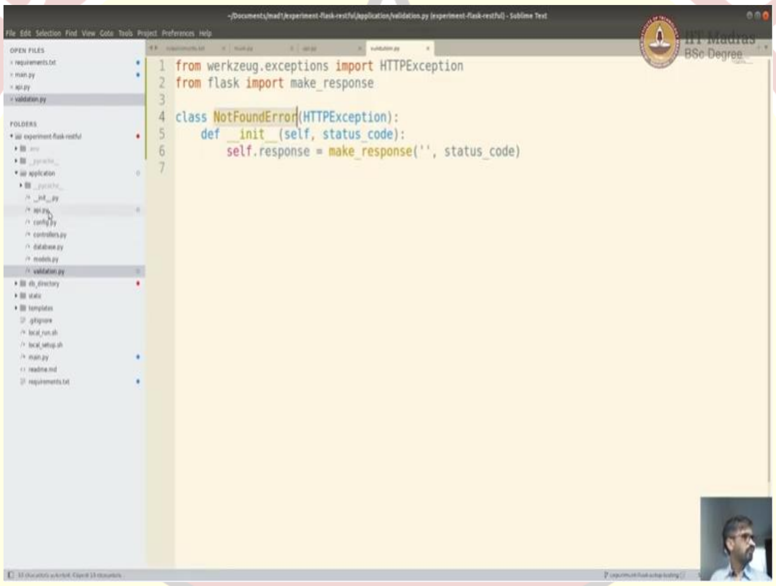


```python
output_fields = {
    "user_id" : fields.Integer,
    "username": fields.String,
    "email": fields.String
}


class UserAPI(Resource):
    @marshal_with(output_fields)
    def get(self, username):
        # Get the username
        print("In UserAPI GET Method", username)

        # Get the User from database based on username
        user = db.session.query(User).filter(User.username == username).first()

        if user:
            # return a valid user JSON
            return user

        else:
            raise NotFoundError(status_code=404)


    def put(self, username):
        print("PUT username", username)
        return {"username" : username}

    def delete(self, username):
        print("DELETE username", username)
```

Hence, I think it is better to set up that error, validation, class and exception class, so we can handle it well, the way we handle it is. If the user is none, then I want to be able to throw an exception and just be handled by the framework. For example, I want to be able to do raise NotFoundError and I just want to say, status_code=404 and I think, it should be able to handle the rest of the things.

Now, we can write our own way of hand. I mean there are built-in ways to handle this, but I am going to write a custom way of handling this so that in future, if you want to build your own custom way, you can do it very easily.

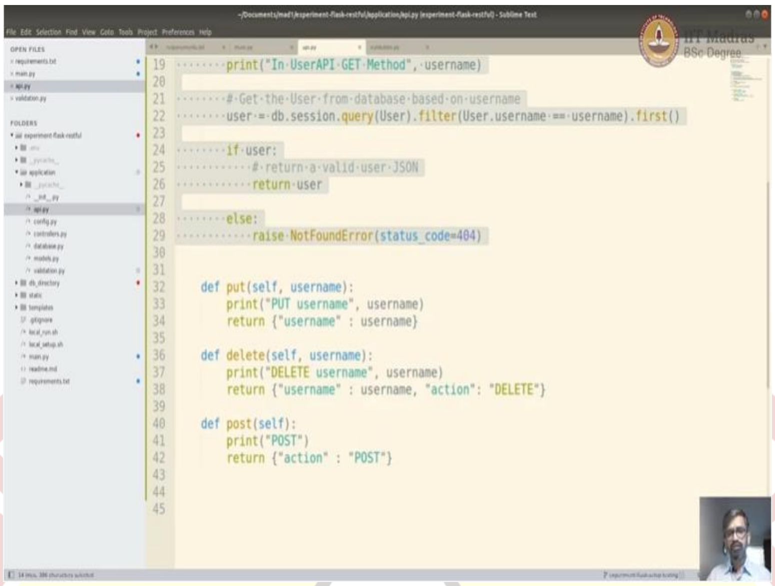
(Refer Slide Time: 35:24)



```
from werkzeug.exceptions import HTTPException
from flask import make_response

class NotFoundError(HTTPException):
    def __init__(self, status_code):
        self.response = make_response('', status_code)
```

So, let us create a class called validation or a file called validation, module.py. I just want to throw http exception as a class, so that all our exceptions are extended of that and handled in a proper way. So I am just going to import that here. That is the class that we are going to extend and I am just going to define a new class called NotFoundError and which extends HTTPException and then I am going to define the constructor for it, which takes http as code and, makes a response with empty message and just a status_code, this make response is a built-in function in flask, so I am just going to use that.

(Refer Slide Time: 38:03)



```
        print("In UserAPI GET Method", username)

        # Get the User from database based on username
        user = db.session.query(User).filter(User.username == username).first()

        if user:
            # return a valid user JSON
            return user

        else:
            raise NotFoundError(status_code=404)


    def put(self, username):
        print("PUT username", username)
        return {"username" : username}

    def delete(self, username):
        print("DELETE username", username)
        return {"username" : username, "action": "DELETE"}

    def post(self):
        print("POST")
        return {"action" : "POST"}
```

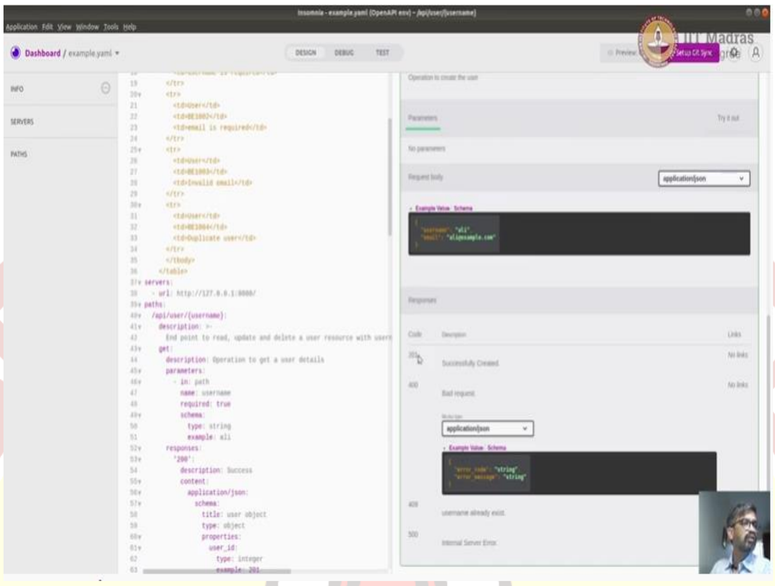
So here is a built-in class, a built-in function and we are going to use that built-in function to run the create the response and return it. So now I am just going to throw this arrow in our API, so same error. I just need to import from application.validation, import, NotFoundError. Ok, let us see if it works, Let us just restart just to make sure. Ok, it is the same thing. I think we should not get this. We should get an empty body, execute it.

Yes, it is just not found, it is empty body. This content length, you can see is 0 or it is not returning anything. It is just returning the error code, but if I send Thejeshgn and click on execute, oh spelling mistake, hence not found thegeshgn. Now we can see that it returns the valid JSON and, like you can see, the content type is valid JSON, and this is the format which was defined and it matches the format.

You can see that this is what we are getting, and this is what the example response was and as defined, and when there is error we are going to do error. Actually, there is no specific 400 because we are not sending any input other than: thegeshgn, which is parameter. So we do not need to do much of this validation same with 500.

I think, by default it will throw an error if there is an internal error or you can do a try catch here in case something happens. While doing this, you can do a try catch and throw a 500 error as a measure, as a good way to throw in like a handling the exception, for example, if you are using those database on a different server. And if that server goes down or if there is a connection error, instead of showing some random message, you can do a try catch and throw a really a 500 error with a good message so that downstream developer will know what is error.

(Refer Slide Time: 39:14)



```
          </tr>
          <tr>
            <td>User</td>
            <td>BE1002</td>
            <td>email is required</td>
          </tr>
          <tr>
            <td>User</td>
            <td>BE1003</td>
            <td>Invalid email</td>
          </tr>
          <tr>
            <td>User</td>
            <td>BE1004</td>
            <td>Duplicate user</td>
          </tr>
          </tbody>
        </table>
servers:
  - url: http://127.0.0.1:8080/
paths:
  /api/user/{username}:
    description: >-
      End point to read, update and delete a user resource with usern
    get:
      description: Operation to get a user details
      parameters:
        - in: path
          name: username
          required: true
          schema:
            type: string
            example: ali
      responses:
        '200':
          description: Success
          content:
            application/json:
              schema:
                title: user object
                type: object
                properties:
                  user_id:
                    type: integer
                    example: 201
```

Operation to create the user

Parameters

No parameters

Request body: application/json

```
{
  "username": "ali",
  "email": "ali@example.com"
}
```

Responses

Code	Description	Links
201	Successfully Created	No links
400	Bad request	No links
409	username already exist	
500	Internal Server Error	

```
{
  "error_code": "string",
  "error_message": "string"
}
```

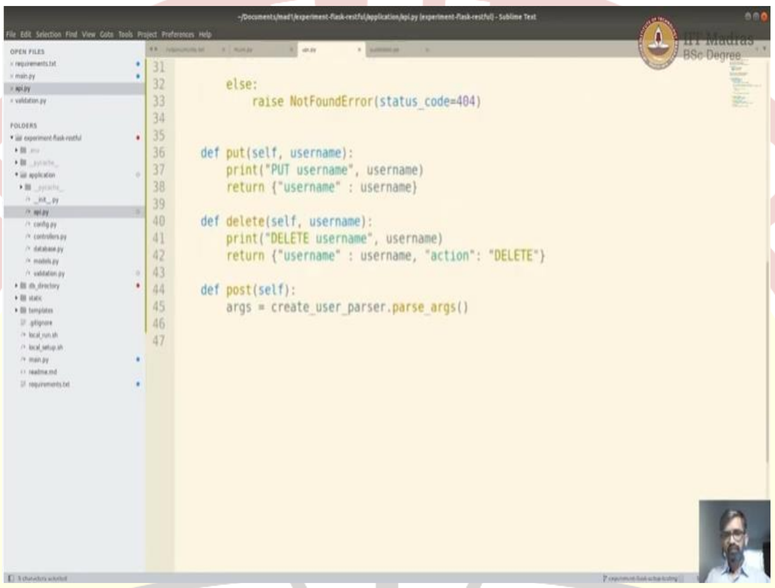
Now we done a decent job with our get. Let us do another one called, probably useful one for us to learn a post, post is where you are going to insert a row into the database or you can create An user. Let us look at the documentation as usual, so here is the post, I am just going to cancel this here. Is the input schema, It takes username, Ali and email, so it takes the URL of this format /api/user and it needs to be Posted at that on error on 200. You will get the whole body on 201. This is showing the previous one. Actually, let me just refresh it on come in a valid insert. It just returns 201 and successfully created.

If there is an error, it will write an error. Sometimes it could also be that it will return. 200 people would return 201 and actually written the actual object. So we can look at both how we can do both of them just for our information, but here according we should have to do according to the documentation, hence according to the documentation on creation. We are not going to return any response body we are just going to written 201.

On error we are going to respond with error code and error message, and there are a couple of errors defined here that user name is required. Email is required, invalid email, duplicate, user and etc. We will try and implement all of these errors, because it is important and that is what has been defined in the documentation.

Let us see how the input format is this now in flask, you can get the JSON directly from request. If you want, you can do that, you can just get the request directly, but a restful flask restful gives you an option to pass the request using called it something called request parser. We are going to use that in a limited form, because we are going to write our own custom exceptions to throw an error.

(Refer Slide Time: 41:58)

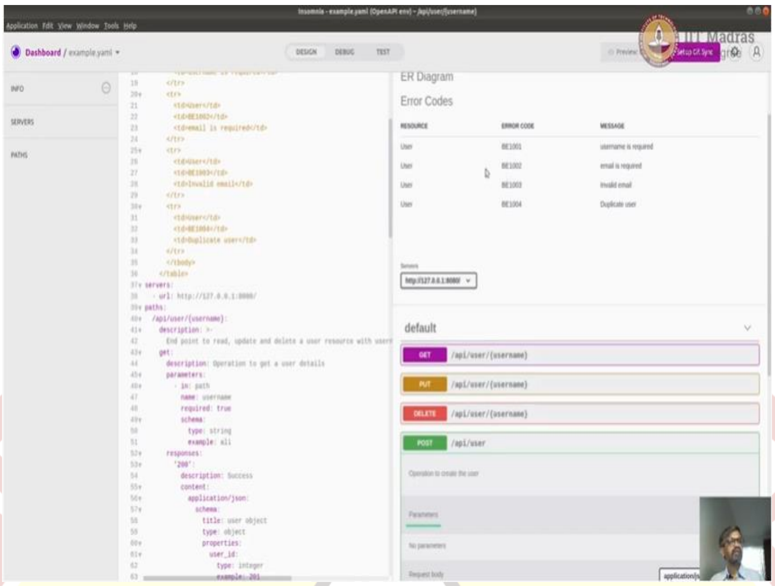


```
        else:
            raise NotFoundError(status_code=404)


    def put(self, username):
        print("PUT username", username)
        return {"username" : username}

    def delete(self, username):
        print("DELETE username", username)
        return {"username" : username, "action": "DELETE"}

    def post(self):
        args = create_user_parser.parse_args()
```

We are not going to use a inbuilt error because inbuilt error will not match The format of this. It has a different format of throwing error since we are going to use our own exception class to throw an error. Let us see um, so I am just going to write the details for a parser, so I am going to create a create user parser, which is a request parser. I am just going to add the argument username and email. This will be part of our request.

So, if you are going to use the in build one, you can define, type is equal to string and whether it is required, not required. All of that we are not going to do that because I do not want to use the inbuilt validator. I am just going to use this just to pass simple ones. We can even go ahead and pass using JSON. request.getjson and do the thing, but it is ok, let us do using this once you do that I am just going to do get the args from the parse. So when you do this parse args, it is going to pass the request and then written the args, args will have username and email. So let us get those values. So I am going to get username.

(Refer Slide Time: 43:07)


Insomnia - example.yaml (OpenAPI env) - /api/user/{username}
Application Edit View Window Tools Help
Dashboard / example.yaml
DESIGN
DEBUG
TEST
Preview
Setup Git Sync
INFO
SERVERS
PATHS
IIT Madras

```
      </tr>
      <tr>
        <td>User</td>
        <td>BE1002</td>
        <td>email is required</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1003</td>
        <td>Invalid email</td>
      </tr>
      <tr>
        <td>User</td>
        <td>BE1004</td>
        <td>Duplicate user</td>
      </tr>
      </tbody>
    </table>
servers:
  - url: http://127.0.0.1:8080/
paths:
  /api/user/{username}:
    description: >-
      End point to read, update and delete a user resource with usern
    get:
      description: Operation to get a user details
      parameters:
        - in: path
          name: username
          required: true
          schema:
            type: string
            example: ali
      responses:
        '200':
          description: Success
          content:
            application/json:
              schema:
                title: user object
                type: object
                properties:
                  user_id:
                    type: integer
                    example: 201
```

ER Diagram

Error Codes

RESOURCE	ERROR CODE	MESSAGE
User	BE1001	username is required
User	BE1002	email is required
User	BE1003	Invalid email
User	BE1004	Duplicate user

Servers
http://127.0.0.1:8080/

default

GET /api/user/{username}
PUT /api/user/{username}
DELETE /api/user/{username}
POST /api/user

Operation to create the user

Parameters

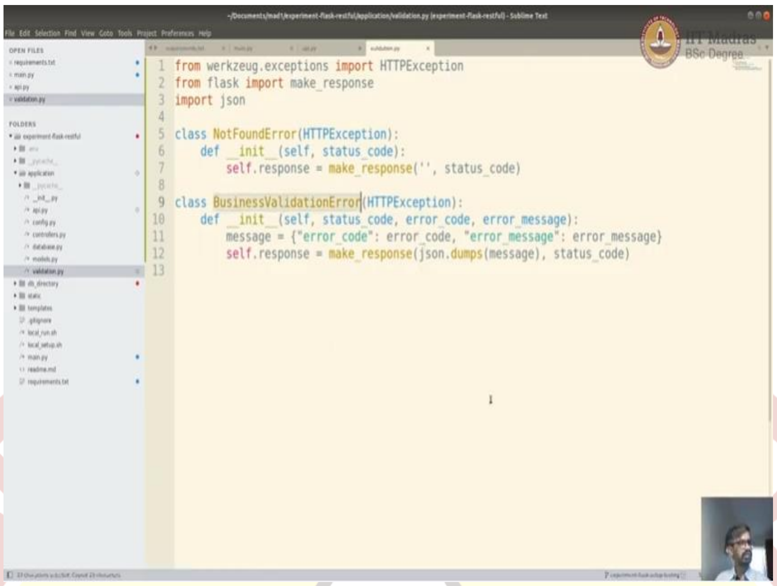
No parameters

Request body

If the there is no username it will return, none I am trying to get email. If there is no email, it will return none, straightforward. So now, in our case, username is a valid required field and even email is required free. So if there is no username found, I want to throw an error.

Similarly, if the email is none, I want to throw an error. So, let us just check I mean you can add more validation, I am Just going to show some simple validation. If username is none, I want to throw a business error, validation, business validation error with status code = 400 error code BE1001 and my error messages username is required, which is the same format that we have defined here like with the error code BE1001 and username is required as a message.

(Refer Slide Time: 44:05)



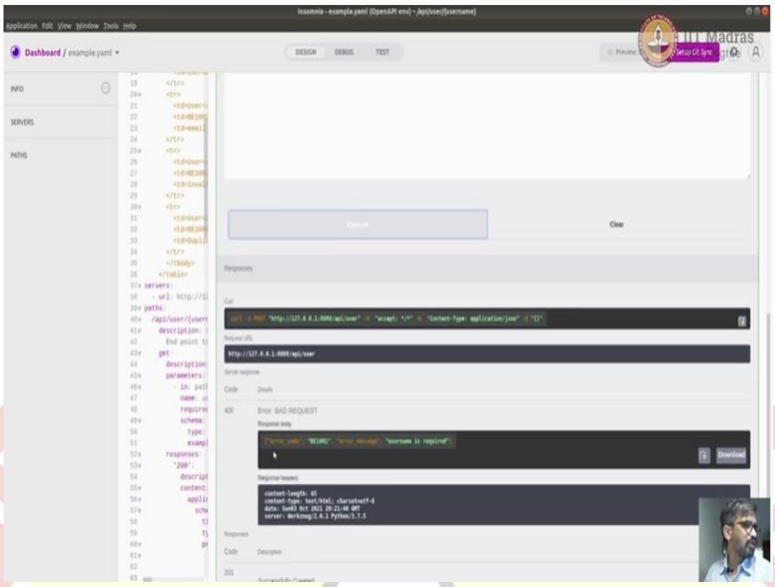
```python
from werkzeug.exceptions import HTTPException
from flask import make_response
import json

class NotFoundError(HTTPException):
    def __init__(self, status_code):
        self.response = make_response('', status_code)

class BusinessValidationError(HTTPException):
    def __init__(self, status_code, error_code, error_message):
        message = {"error_code": error_code, "error_message": error_message}
        self.response = make_response(json.dumps(message), status_code)
```

Now, we have defined it, we have used it, but we are not implemented it, so I am going to just implement it, in the same validation class, I am going to define a class which extend the same http exception and format also will look similar just that our input will take more attributes, so it will take https status code, error code, error message, so error code and error message has to be sent as part of the body. You can see that here. If there is an error, if this is the body and error code is 400 and we need to do, we need to set up this as the response, so I am going to make a response with these two.

So, message is equal to error code and error message okay and then just going to return this. Now, you cannot return the object. You have to return the string, so we have to do json.dumps, which converts an object into JSON string. So you can just import JSON and do json.dumps.

Now we have to this is ready to be thrown or to be used. So, let us go to business validation, error and use this business validation error here, so we are already used it, but we need to import it. Ok, so now we have it here. Let us try just see if it works, we will just put a pass if to end it, if nothing happens, just run it or a pass is not defined, so you have to define like pass loads, define it, define it as an import it here. Okay, I am going to import that also, so we do not get that error. Ok running: let us go here just going to extend this part to make it clearer.

(Refer Slide Time: 46:32)



```
curl -X POST "http://127.0.0.1:8080/api/user" -H "accept: */*" -H "Content-Type: application/json" -d "{}"
```

```
http://127.0.0.1:8080/api/user
```

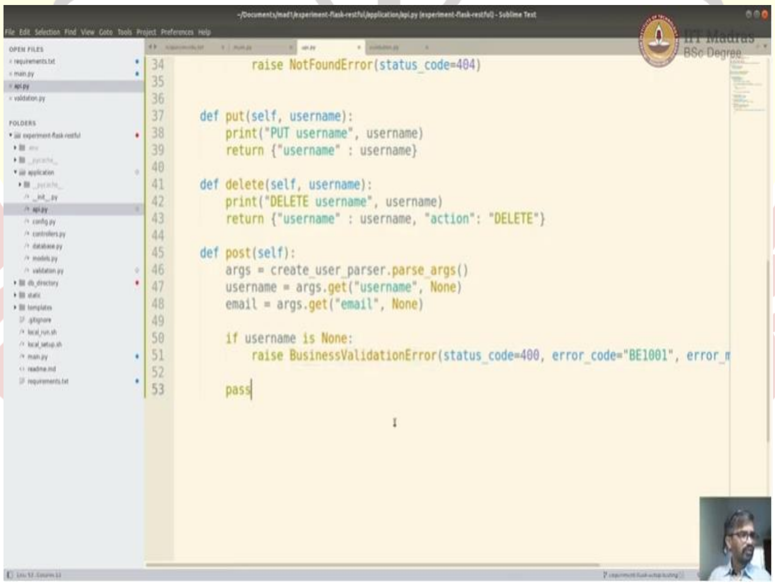
400 Error: BAD REQUEST

```
{"error_code": "BE1001", "error_message": "username is required"}
```

```
content-length: 65
content-type: text/html; charset=utf-8
date: Sun03 Oct 2021 20:21:40 GMT
server: Werkzeug/2.0.1 Python/3.7.5
```

We are here, we are going to click on try it I am not going to give anything, click on execute, it ran through an error, bad request BE1001 username is required. You can see it is returning 400. Now, let us just put some username, execute it is not doing anything right because. If this is valid, we are not doing the rest.

(Refer Slide Time: 47:07)

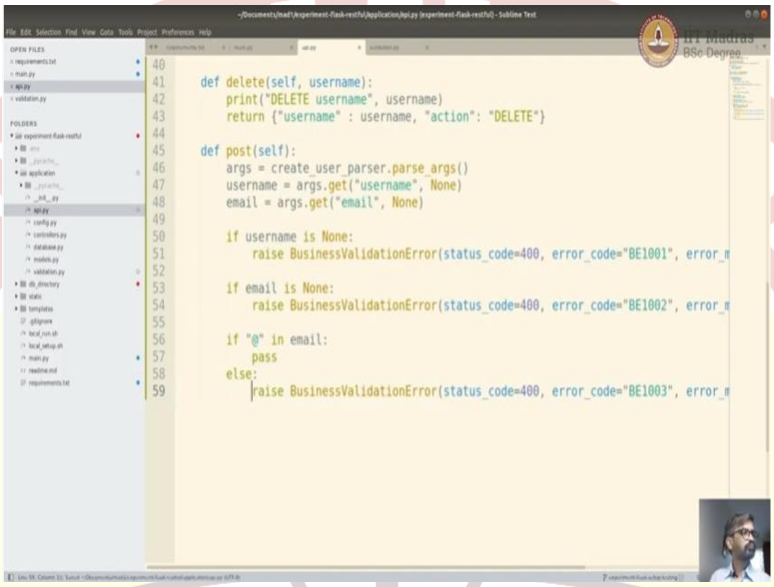


```python
            raise NotFoundError(status_code=404)


    def put(self, username):
        print("PUT username", username)
        return {"username" : username}

    def delete(self, username):
        print("DELETE username", username)
        return {"username" : username, "action": "DELETE"}

    def post(self):
        args = create_user_parser.parse_args()
        username = args.get("username", None)
        email = args.get("email", None)

        if username is None:
            raise BusinessValidationError(status_code=400, error_code="BE1001", error_m

        pass
```

So, this statement and this business validation is working now, let us continue with the same flow and if email is none will do the same, and what is the error code if for valid email, BE1002 and BE here I means just business error, could be any error code as per the designer.

Whatever they give BE1002 email is required, I am just going to use. Email is required, you are, going to use the same message and same error code. That is provided by the design, documentation. I think that the idea of the design documentation that we need to stick to it. Now, we are given the user name, we will try to execute it through an error with email is required. You can see that error code BE1002, email is required.

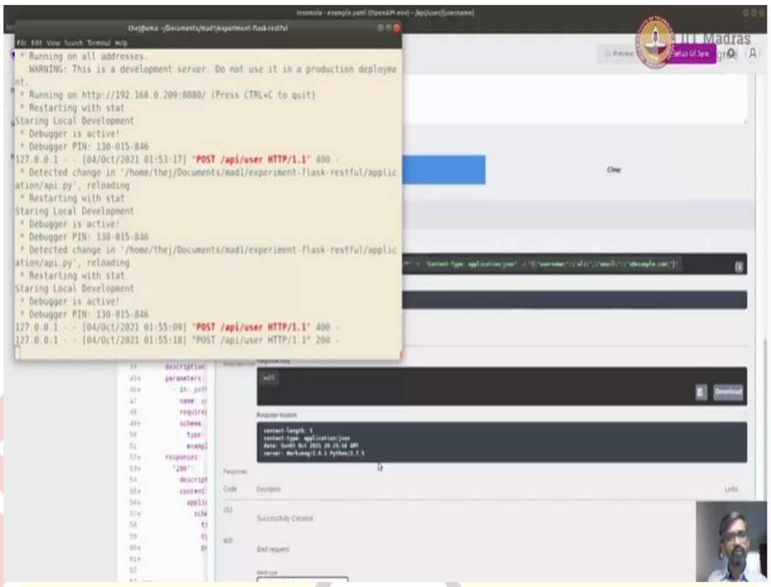
(Refer Slide Time: 49:28)



```python
    def delete(self, username):
        print("DELETE username", username)
        return {"username" : username, "action": "DELETE"}

    def post(self):
        args = create_user_parser.parse_args()
        username = args.get("username", None)
        email = args.get("email", None)

        if username is None:
            raise BusinessValidationError(status_code=400, error_code="BE1001", error_m

        if email is None:
            raise BusinessValidationError(status_code=400, error_code="BE1002", error_m

        if "@" in email:
            pass
        else:
            raise BusinessValidationError(status_code=400, error_code="BE1003", error_m
```

Let us see if email is valid, I mean you could validate the email in many ways, I am going to simply check whether the email has @ symbol it is very complex to validate an email. There are many ways you can do that you need to have a dot, you need to have a valid domain name, etc, etc, but I am just going to go with the simplest validation here, I am just going to check if email has at symbol and if it has at symbol, I am just going to consider it as valid okay, if at in email, this checks that email string has @ symbol, then I am not going to do anything continue with the with my function else, I am going to throw an error. The error would BE1003 invalid email.

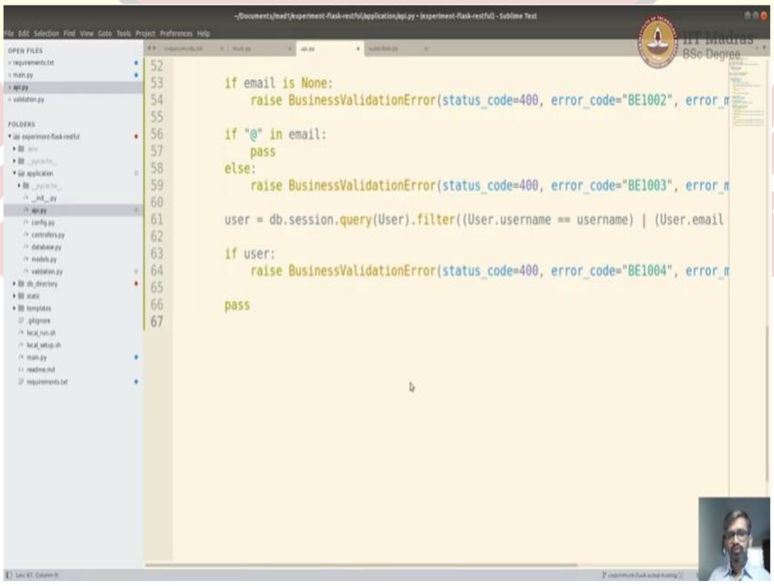
(Refer Slide Time: 50:08)



```
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployme
nt.
 * Running on http://192.168.0.209:8080/ (Press CTRL+C to quit)
 * Restarting with stat
Staring Local Development
 * Debugger is active!
 * Debugger PIN: 130-015-846
127.0.0.1 - - [04/Oct/2021 01:53:17] "POST /api/user HTTP/1.1" 400 -
 * Detected change in '/home/thej/Documents/mad1/experiment-flask-restful/applic
ation/api.py', reloading
 * Restarting with stat
Staring Local Development
 * Debugger is active!
 * Debugger PIN: 130-015-846
 * Detected change in '/home/thej/Documents/mad1/experiment-flask-restful/applic
ation/api.py', reloading
 * Restarting with stat
Staring Local Development
 * Debugger is active!
 * Debugger PIN: 130-015-846
127.0.0.1 - - [04/Oct/2021 01:55:09] "POST /api/user HTTP/1.1" 400 -
127.0.0.1 - - [04/Oct/2021 01:55:18] "POST /api/user HTTP/1.1" 200 -
```

Ok, let us see if our server is running. Yes, let us go here. Let us add an email accept example .com, execute seems to be valid, but email is required is showing spelling mistake here, email execute. So, like everything worked because we have done after that, we are not returning anything. We are returning m null or empty. Hence the thing null here, but if I give a wrong email according to us, if it does not have an @ symbol, it is a wrong email. So, let us execute then it shows invalid email.

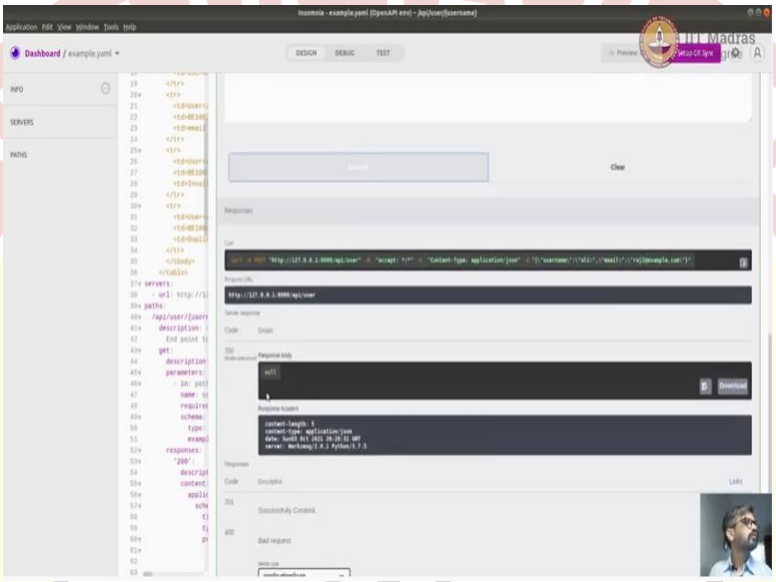
(Refer Slide Time: 52:38)



```
        if email is None:
            raise BusinessValidationError(status_code=400, error_code="BE1002", error_m

        if "@" in email:
            pass
        else:
            raise BusinessValidationError(status_code=400, error_code="BE1003", error_m

        user = db.session.query(User).filter((User.username == username) | (User.email

        if user:
            raise BusinessValidationError(status_code=400, error_code="BE1004", error_m

        pass
```

Next, we want to check whether the this username or email already exists in the database. If it exists in database, then we do not want to insert else we will insert. Now you can check one by one or you can check in with an or clause. We will go with an or clause, a simple check, because I think here, though the error is actually duplicate user. It can be a duplicate user. If it email, it can, email is duplicated or even if the username is duplicated, so we will just check with the r class much easier. I am just going to paste the query here, so you can check that it is a similar query, as in getdb.session.query(user).filter((user=username) | (user. email = email)), then get the first one.

(Refer Slide Time: 53:18)



```
curl -X POST "http://127.0.0.1:8080/api/user" -H "accept: */*" -H "Content-Type: application/json" -d "{\"username\":\"ali\",\"email\":\"raj@example.com\"}"
```

```
http://127.0.0.1:8080/api/user
```

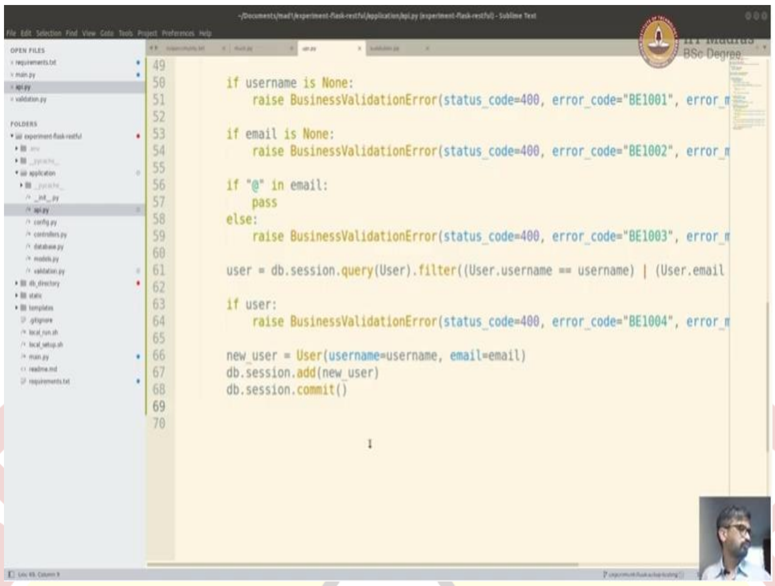
```
null
```

```
content-length: 5
content-type: application/json
date: Sun03 Oct 2021 20:20:32 GMT
server: Werkzeug/2.0.1 Python/3.7.5
```

If this returns a value, then we have to throw an error again, we will throw a business error, saying, duplicate user. So we will do 1004. We will see duplicate user. So now what we have to do, we have to check it. I am just going to return else; I am not going to do currently anything I am just going to do pass will implement inserting the next step, I am just going to just check this step. Whether this step is Working, I am going to restart the server, so we are given Ali an example, let us return, existing user or existing email and see whether it throws an error Ali is a non-existing.

Let us use this raj@example.com as a email and execute should throw an error duplicate user. Now, if I make it raj2, it does not throw any error, but if I make the user as raj, then it throws an error duplicate user. If both are also duplicate, they given, then it should throw an error, throws an error, depends on what error message. If you want to give specific error message that duplicate Email id, please do not use, then you have to check specifically and then through an error. Then we will have to probably write two, queries and check it and then throw an error.

(Refer Slide Time: 54:45)

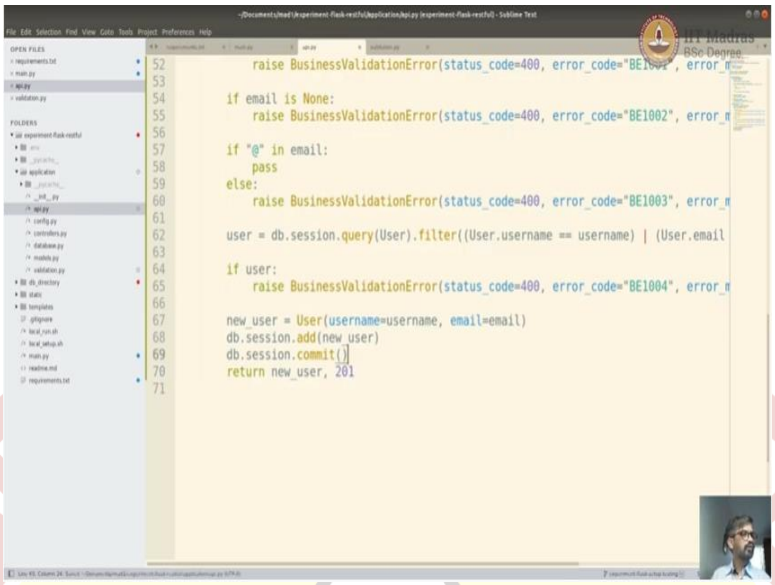


```
        if username is None:
            raise BusinessValidationError(status_code=400, error_code="BE1001", error_m

        if email is None:
            raise BusinessValidationError(status_code=400, error_code="BE1002", error_m

        if "@" in email:
            pass
        else:
            raise BusinessValidationError(status_code=400, error_code="BE1003", error_m

        user = db.session.query(User).filter((User.username == username) | (User.email

        if user:
            raise BusinessValidationError(status_code=400, error_code="BE1004", error_m

        new_user = User(username=username, email=email)
        db.session.add(new_user)
        db.session.commit()
```

Then, if everything goes well now, we want to insert, to insert we are going to create a new user, that is by using the user model, and it just needs two inputs, one is username= username, what we got and email=email, what we got. We have created a new user, then we are going to add it to session and then commit and then either we can return a new user or return nothing just return 201. Now, let us see if everything goes well, it will return 201. Let us see, let us create another user called raj2, raj2@example.com click on execute 201. It returned 201.

So, 201 is created, and then you can see that it has returned empty body. We could make it completely empty. I am just returning empty string here and then let us go to the database and check you can see raj2 has been inserted. Now, let us say our documentation was different and along with 201, instead of just returning empty body, we wanted to return the whole object. It is much easier.

(Refer Slide Time: 56:28)

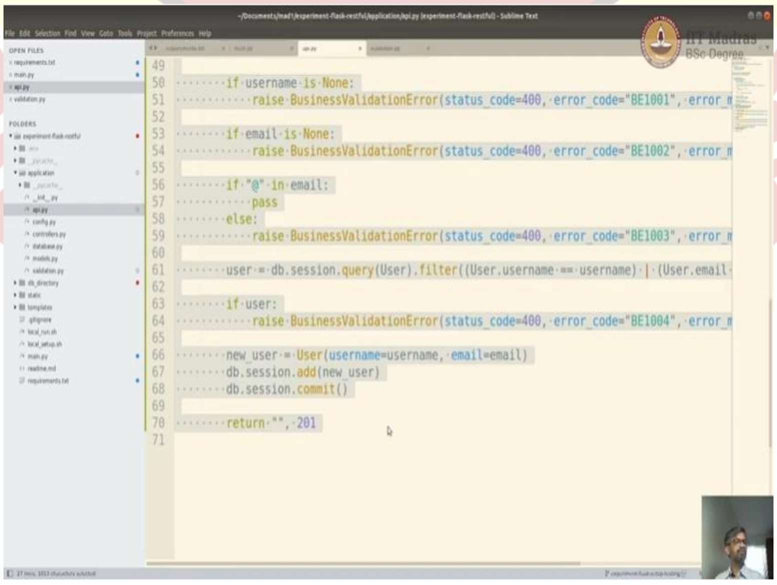


```
            raise BusinessValidationError(status_code=400, error_code="BE1001", error_m

        if email is None:
            raise BusinessValidationError(status_code=400, error_code="BE1002", error_m

        if "@" in email:
            pass
        else:
            raise BusinessValidationError(status_code=400, error_code="BE1003", error_m

        user = db.session.query(User).filter((User.username == username) | (User.email

        if user:
            raise BusinessValidationError(status_code=400, error_code="BE1004", error_m

        new_user = User(username=username, email=email)
        db.session.add(new_user)
        db.session.commit()
        return new_user, 201
```

We could just use the same marshall fields here. We can just add here and then return. Instead of this, you can just return new user and then let us try it with inserting another user go up, raj3 and execute you can see that it is returning actual object along with 201.

It is all the returning object, but you should stand to the documentation that was provided to you, which is in 201. There is nothing, no body to be returned, so you can always do that. So we just made back to like what I had before just to follow the documentation.

(Refer Slide Time: 58:48)



```

        if username is None:
            raise BusinessValidationError(status_code=400, error_code="BE1001", error_m

        if email is None:
            raise BusinessValidationError(status_code=400, error_code="BE1002", error_m

        if "@" in email:
            pass
        else:
            raise BusinessValidationError(status_code=400, error_code="BE1003", error_m

        user = db.session.query(User).filter((User.username == username) | (User.email

        if user:
            raise BusinessValidationError(status_code=400, error_code="BE1004", error_m

        new_user = User(username=username, email=email)
        db.session.add(new_user)
        db.session.commit()

        return "", 201
```

So, I just made it as before. Just to follow the documentation, nothing will be returned, 201 status code will be returned and hence no match link as well. So that actually completes our post request. It is quite straightforward and simple; I have actually simplified a lot of stuff here. You could make it much more elaborate in terms of validation, checking, etc.