INDIAN INSTITUTE OF TECHNOLOGY MADRAS • सिद्धिर्भवति कर्मजा •

IIT Madras

ONLINE DEGREE

Modern Application Development- 1
Professor. Thejesh G N
Software Consultant
Indian Institute of Technology, Madras
Structure of a Flask App

(Refer Slide Time: 0:15)

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

Go

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

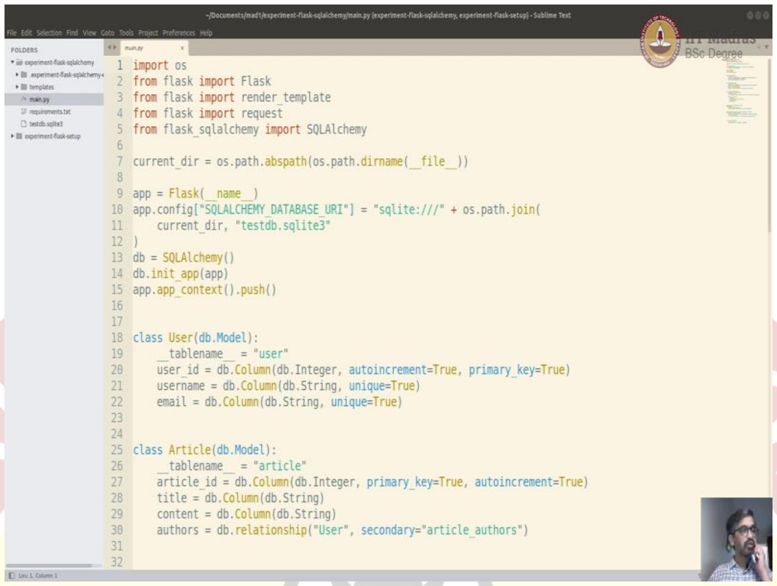
This part of the documentation, which is mostly prose, begins with some background information about Flask, then focuses on step-by-step instructions for web development with Flask.

- 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

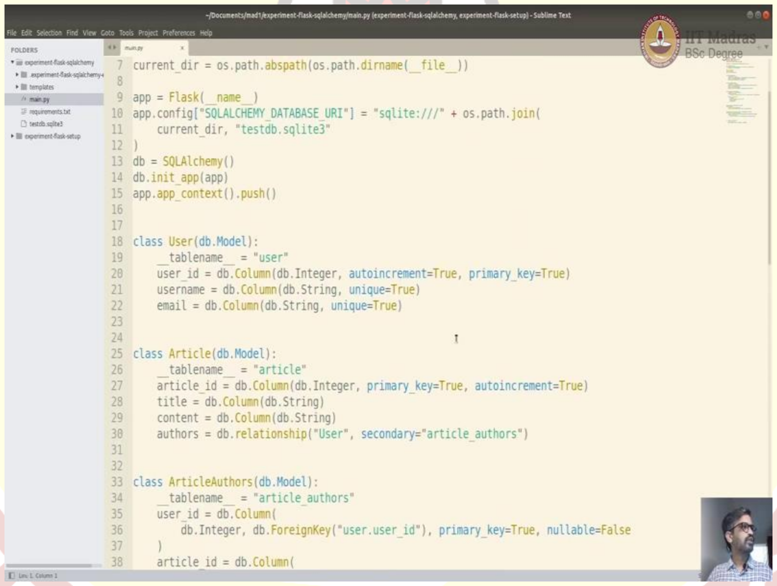
Welcome to the modern application development screencasts. In this short screencast, we will learn how to set up a simple yet complete flask project. Before we start open the following applications on your Ubuntu desktop. So, that they can work along with me a Browser, Chrome or Firefox is fine. Text your IDE any of the open source IDE is fine just to explore it you need 10 color code and you think would work.

I will use one of the open source once suggested before. Terminal the one which comes built into Ubuntu is good enough. And please make sure you also, have Python 3 installed so that it is easy to run the code. Until now, we saw one file flask project literally.

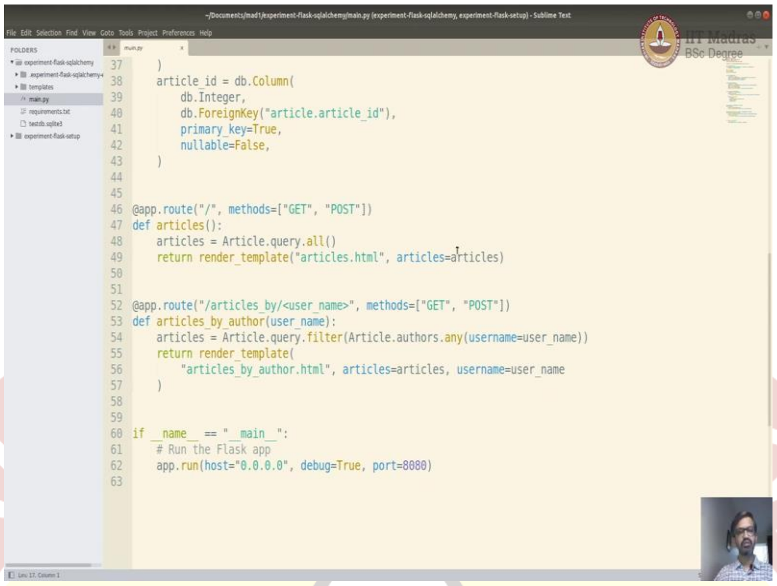
(Refer Slide Time: 1:15)


~/Documents/mad1/experiment-flask-sqlalchemy/main.py (experiment-flask-sqlalchemy, experiment-flask-setup) - Sublime Text
FOLDERS
experiment-flask-sqlalchemy
templates
main.py
requirements.txt
testdb.sqlite3
experiment-flask-setup
IIT Madras
BSc Degree

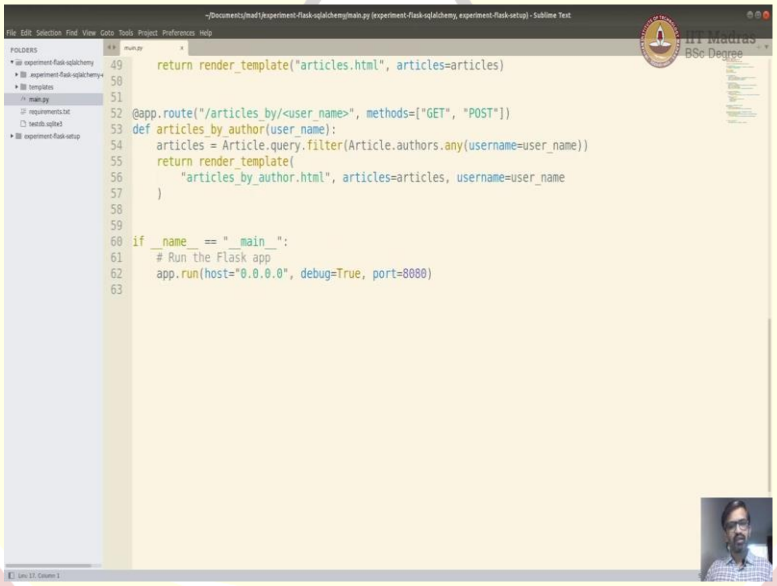
```python
import os
from flask import Flask
from flask import render_template
from flask import request
from flask_sqlalchemy import SQLAlchemy

current_dir = os.path.abspath(os.path.dirname(__file__))

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///" + os.path.join(
    current_dir, "testdb.sqlite3"
)
db = SQLAlchemy()
db.init_app(app)
app.app_context().push()


class User(db.Model):
    __tablename__ = "user"
    user_id = db.Column(db.Integer, autoincrement=True, primary_key=True)
    username = db.Column(db.String, unique=True)
    email = db.Column(db.String, unique=True)


class Article(db.Model):
    __tablename__ = "article"
    article_id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    title = db.Column(db.String)
    content = db.Column(db.String)
    authors = db.relationship("User", secondary="article_authors")

```


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

```python
current_dir = os.path.abspath(os.path.dirname(__file__))

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///" + os.path.join(
    current_dir, "testdb.sqlite3"
)
db = SQLAlchemy()
db.init_app(app)
app.app_context().push()


class User(db.Model):
    __tablename__ = "user"
    user_id = db.Column(db.Integer, autoincrement=True, primary_key=True)
    username = db.Column(db.String, unique=True)
    email = db.Column(db.String, unique=True)


class Article(db.Model):
    __tablename__ = "article"
    article_id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    title = db.Column(db.String)
    content = db.Column(db.String)
    authors = db.relationship("User", secondary="article_authors")


class ArticleAuthors(db.Model):
    __tablename__ = "article_authors"
    user_id = db.Column(
        db.Integer, db.ForeignKey("user.user_id"), primary_key=True, nullable=False
    )
    article_id = db.Column(
```


~/Documents/mad1/experiment-flask-sqlalchemy/main.py (experiment-flask-sqlalchemy, experiment-flask-setup) - Sublime Text
main.py

```python
    )
    article_id = db.Column(
        db.Integer,
        db.ForeignKey("article.article_id"),
        primary_key=True,
        nullable=False,
    )


@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_name
    )


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


~/Documents/mad1/experiment-flask-sqlalchemy/main.py (experiment-flask-sqlalchemy, experiment-flask-setup) - Sublime Text
main.py

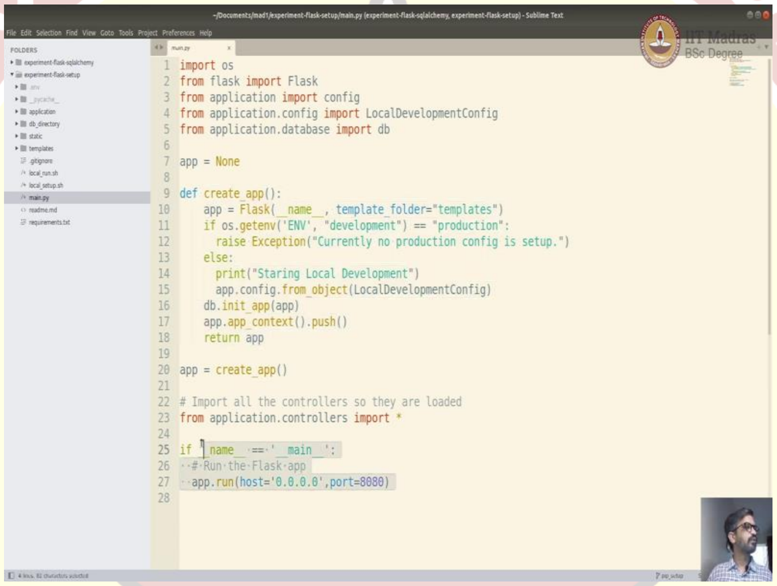
```python
    return render_template("articles.html", articles=articles)


@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_name
    )


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

Let me just show you that actually, we had done an experiment with SQL alchemy and flask where we had written all of the code in just one file in this main file. So, we can see that we are written application setup here or app set up your new setup of the models and then controllers and then your whole application. This is okay this works well if the application is very small. However, as a project grows bigger becomes very difficult to read, understand and maintain.

If all the code is in one single file. So, it is always easy to read if you logically break it. So, like in any Python project, let us use packages to organize code into different logical modules. So, that it is easy to read and find the code and modify it if it is required. It is also a better way of organizing the code so, that where to find what you where to find what. And we will import these modules and packages as in when we require right like in any other Python project.

Before we start, I would like to mention that there are many ways to set up a flask Python project. It depends on team, whom you work with, and the size of the team size of the project, many of the factors that go in. Here I am going to show one way of doing it. I am trying to keep it simple but yet very practical and logical so, that it is easy for all of us to set up and run. So, let me just show you the project setup. So, I am going to close this which we already have.

(Refer Slide Time: 3:12)


~/Documents/mad1/experiment-flask-setup/main.py (experiment-flask-sqlalchemy, experiment-flask-setup) - Sublime Text
main.py

```python
import os
from flask import Flask
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)
    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)
```

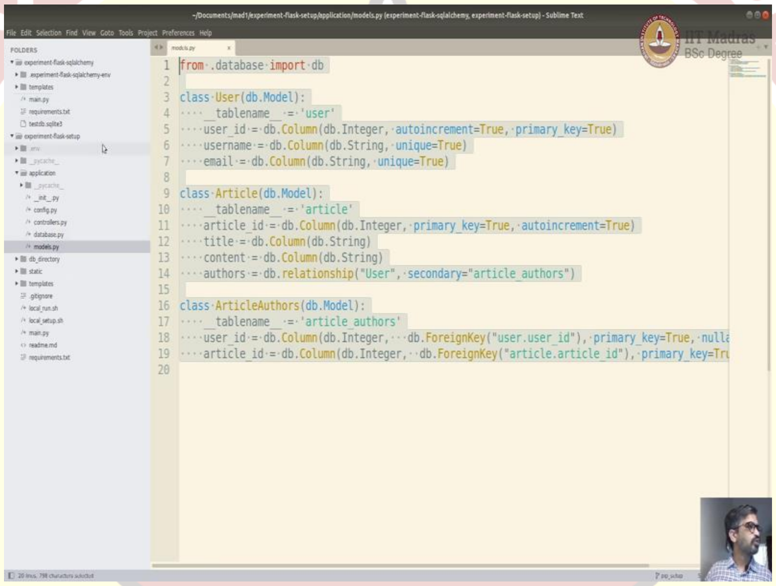
And then open, my new flash setup experimental flash setup. So, here you can see there are there is still a main.py just some basic things. Like for example it has all of my application, or app setup code where I am creating an app and running the app. This is the main functionality of this module or file. So, basically, it could call to create app to create that, then when we actually run it runs the app.

So, we will inside the create app I am just going to read the environment. I am going to tell you more details about reading the environment, basically and configuring. But basically it reads the configuration sets up the db pushes the app into the context and returns the app. The same app is used to run. And we are importing the things that I required to do that for example, db to initialize the db and config to initialize the app config et cetera.

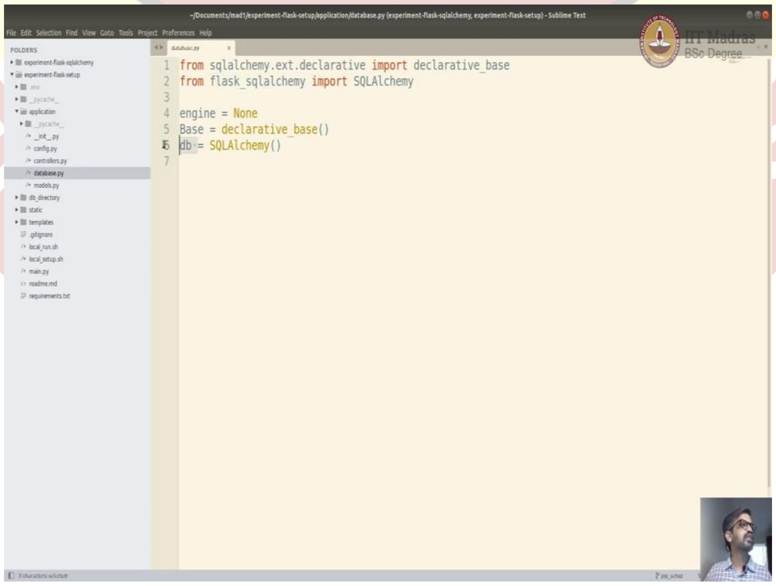
For template folders, were using the default template folders. So, this is the basic setup. And now I am going to explain each one of these. So, let us say what is in the config and what is the local development config. Let us let us see that where it is imported from it actually imported from this package called application.

Where we keep all our application code you can see that when a main.py is outside and rest of the modules or files are inside this package called application within that we have config, controllers, database and models. Now models and controllers are very straightforward and easy.

(Refer Slide Time: 5:14)


~/Documents/mad1/experiment-flask-setup/application/models.py (experiment-flask-sqlalchemy, experiment-flask-setup) - Sublime Text

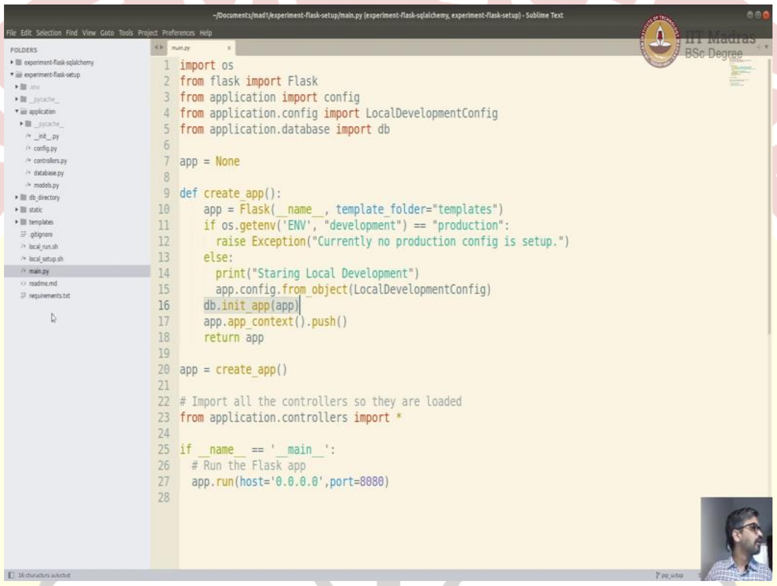
```python
from .database import db

class User(db.Model):
    __tablename__ = 'user'
    user_id = db.Column(db.Integer, autoincrement=True, primary_key=True)
    username = db.Column(db.String, unique=True)
    email = db.Column(db.String, unique=True)

class Article(db.Model):
    __tablename__ = 'article'
    article_id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    title = db.Column(db.String)
    content = db.Column(db.String)
    authors = db.relationship("User", secondary="article_authors")

class ArticleAuthors(db.Model):
    __tablename__ = 'article_authors'
    user_id = db.Column(db.Integer,   db.ForeignKey("user.user_id"), primary_key=True, nulla
    article_id = db.Column(db.Integer,  db.ForeignKey("article.article_id"), primary_key=Tru
```


~/Documents/mad1/experiment-flask-setup/application/database.py (experiment-flask-sqlalchemy, experiment-flask-setup) - Sublime Text

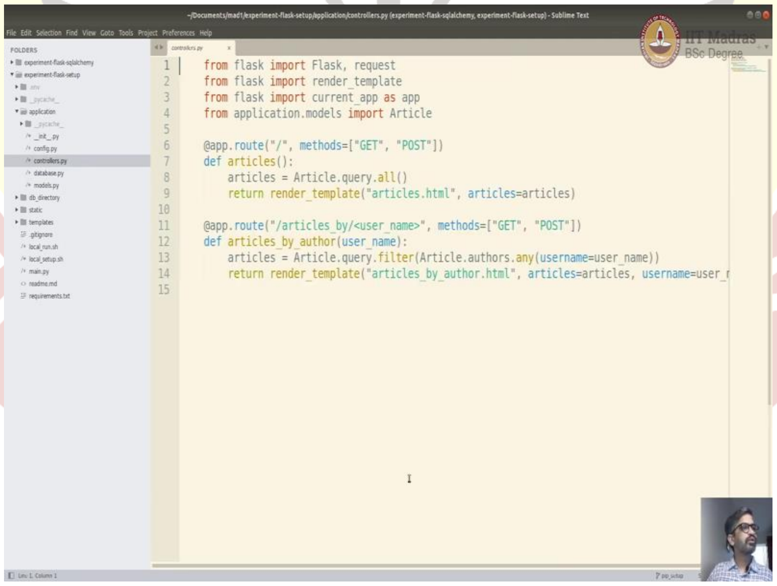
```python
from sqlalchemy.ext.declarative import declarative_base
from flask_sqlalchemy import SQLAlchemy

engine = None
Base = declarative_base()
db = SQLAlchemy()
```

Model have all the models very which is extracted from previous our main.py into separate file called models.py. And we put all of them here. So, that it is easy to find the models in the models .py. Now database.py is not big currently it is a simple thing we just set up sets up our flask SQL Alchemy say exposes db So, that it can be used. So, you can see in that our models, we are importing that and using that.

(Refer Slide Time: 5:51)


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

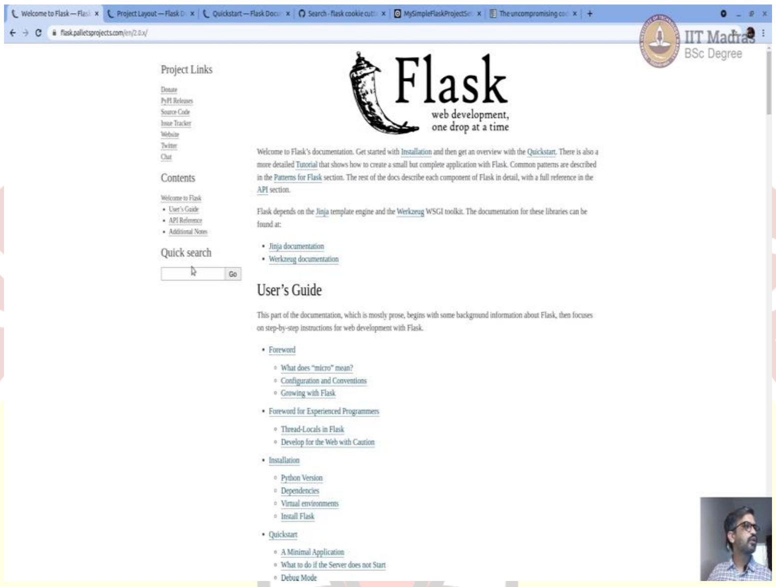
```python
import os
from flask import Flask
from application import config
from application.config import LocalDevelopmentConfig
from application.database import db

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)
    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-setup/application/controllers.py (experiment-flask-sqlalchemy, experiment-flask-setup) - Sublime Text
IIT Madras
BSc Degree

```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_
```

Similarly, in our main.py we are importing that from database and then using that to in it. So, we are done with modules, and database, straightforward. Our controllers is also straightforward we could we are going to put all the controllers into it. But it, you can also extend it if you have lots of controllers you can create separate files for it or modules for it and use it.

(Refer Slide Time: 6:22)



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

Go

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.

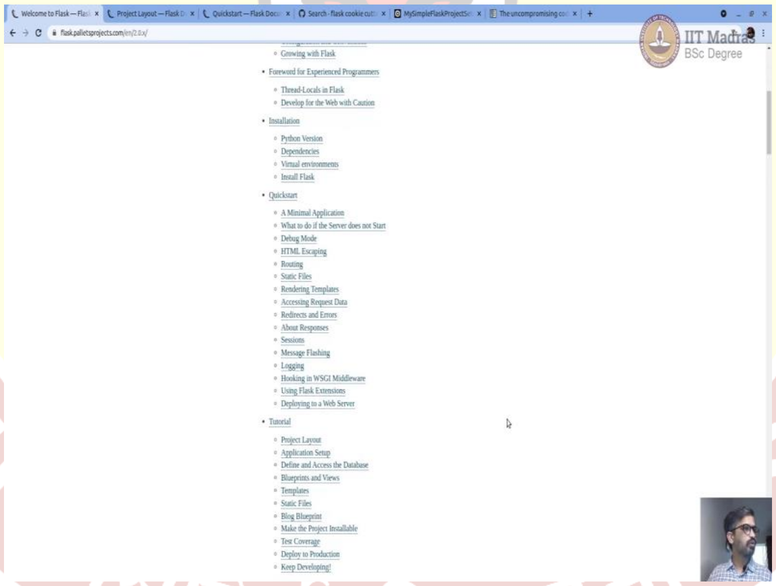
Flask depends on the Jinja template engine and the Werkzeug WSGI toolkit. The documentation for these libraries can be found at:

- 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.

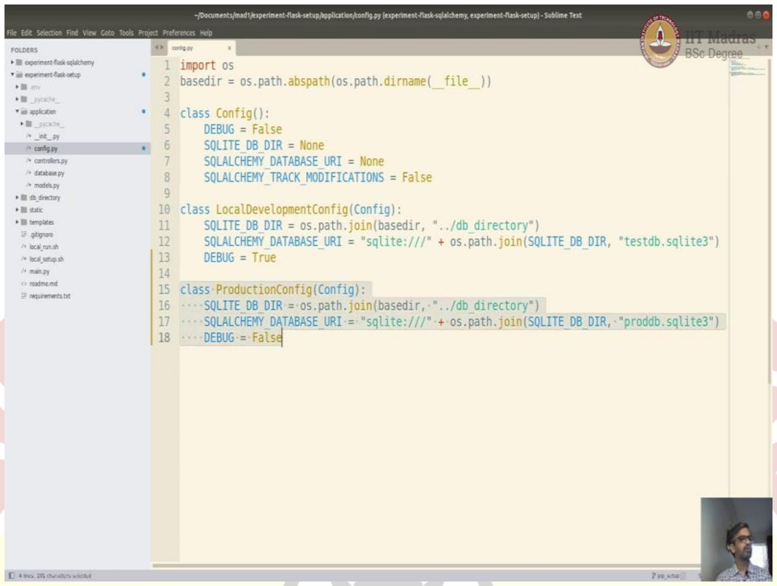
- Foreword
  - What does "micro" mean?
  - Configuration and Conventions
  - Growing with Flask
- Foreword for Experienced Programmers
  - Thread-Locals in Flask
  - Develop for the Web with Caution
- Installation
  - Python Version
  - Dependencies
  - Virtual environments
  - Install Flask
- Quickstart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



  - 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
  - HTML Escaping
  - Routing
  - Static Files
  - Rendering Templates
  - Accessing Request Data
  - Redirects and Errors
  - About Responses
  - Sessions
  - Message Flashing
  - Logging
  - Hooking in WSGI Middleware
  - Using Flask Extensions
  - Deploying to a Web Server
- Tutorial
  - Project Layout
  - Application Setup
  - Define and Access the Database
  - Blueprints and Views
  - Templates
  - Static Files
  - Blog Blueprint
  - Make the Project Installable
  - Test Coverage
  - Deploy to Production
  - Keep Developing!

Also, flask has this concept of blueprint which we are not gone into. But if you really want to use it you can use the blueprint. You can search on the flat blueprints and views, you can try that.

(Refer Slide Time: 6:38)



```python
import os
basedir = os.path.abspath(os.path.dirname(__file__))

class Config():
    DEBUG = False
    SQLITE_DB_DIR = None
    SQLALCHEMY_DATABASE_URI = None
    SQLALCHEMY_TRACK_MODIFICATIONS = False

class LocalDevelopmentConfig(Config):
    SQLITE_DB_DIR = os.path.join(basedir, "../db_directory")
    SQLALCHEMY_DATABASE_URI = "sqlite:///" + os.path.join(SQLITE_DB_DIR, "testdb.sqlite3")
    DEBUG = True

class ProductionConfig(Config):
    SQLITE_DB_DIR = os.path.join(basedir, "../db_directory")
    SQLALCHEMY_DATABASE_URI = "sqlite:///" + os.path.join(SQLITE_DB_DIR, "proddb.sqlite3")
    DEBUG = False
```

But let us go with a simple ones. And yet powerful ones, the standard controllers, with routing, So, they are there in the controllers, it is easy to find. If we really want early want to separate it out you can say I want all the article related controllers in article_controllers. And all the author related controls and the author_underscore controllers etcetera. But here, I am just going to keep one of them.

And then the most important one is config. Now, each application will have certain set of configurations which vary based on the environment where the application is running. Whether it is running locally, or in production, or staging environment, et cetera. Like some of the setting could be like whether the debug statements should be printed out to in the browser. For example, in the local level you want them to be printed.

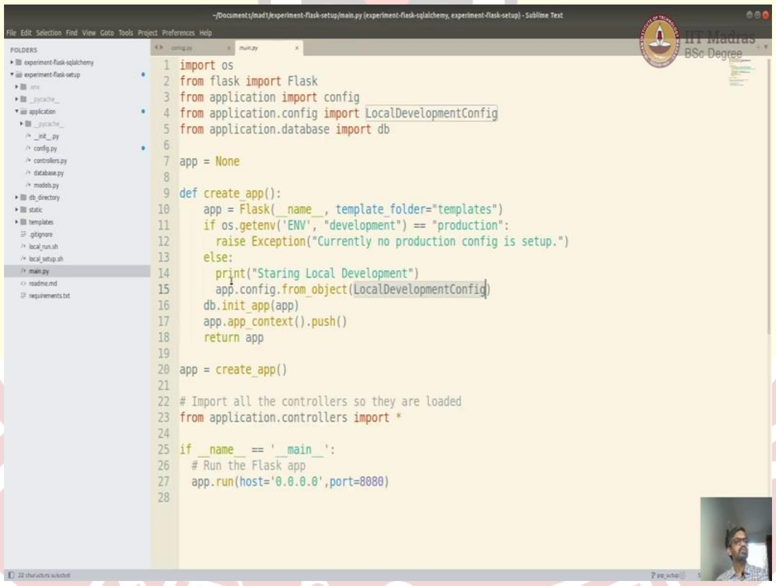
But in production, you do not want it to be printed. Because you will be exposing some secret some information about the app. So, you want to switch off the debug in production. Also, for example, in local development you might be using SQL lite. In production, you might be using a completely different database type it you probably are using Postgres in production or My SQL in production so, you could set up that.

Here, I have you can see that I have config.py. And I have defined a class called config. I here I have defined all the variables that we generally use. And then for local development configure set up a class, another class called local development config which extends this class config and set up some of them. Actually, whatever we want to overwrite.

For example, I just want to overwrite this SQLITE_DB_DIR and SQLALCHAMY_DATABASE_URI. So, I have just done that. So, that I can set up the SQLALCHAMY_DATABASE_URI form a local development environment, if you wanted production. Let us for example here debug is true. And let us say I wanted to set up a production config. Now, I would probably copy paste this. I am calling ProductionConfig. And let us say even that is in db directory.

But let us say if it was called production db instead of test db prod db then and I also wanted to debug to be false because in production I do not want. So, you can set up different environments like this. And based on ENV variable available at the OS you can use one of the config. That is done here.

(Refer Slide Time: 9:28)

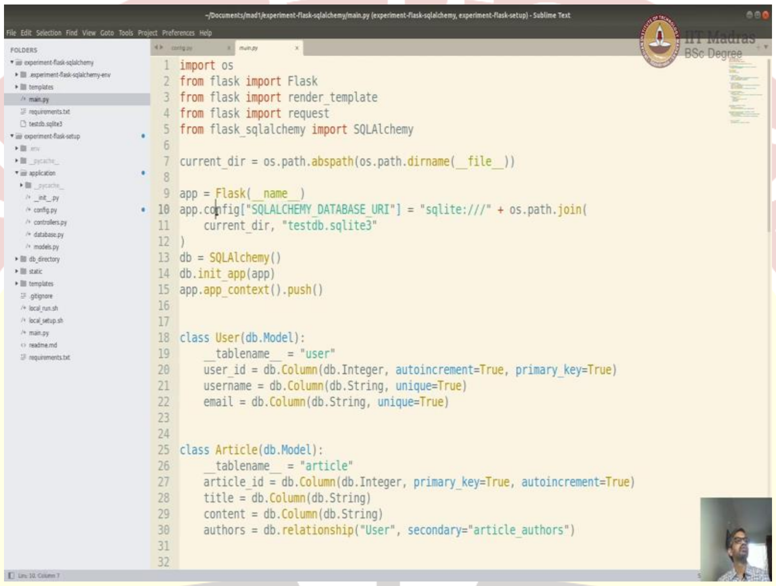

~/Documents/mad1/experiment-flask-setup/main.py (experiment-flask-sqlalchemy, experiment-flask-setup) - Sublime Text
config.py
main.py

```
import os
from flask import Flask
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)
    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)
```

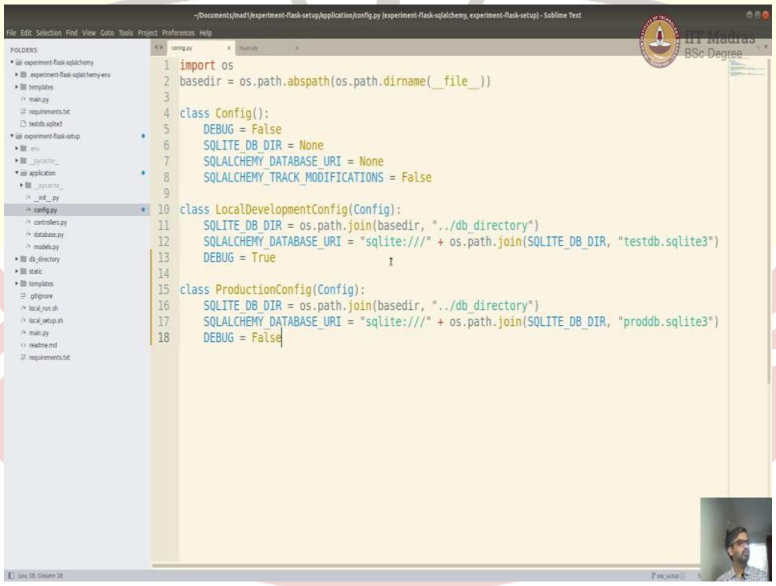
For example, here I am reading from the environment a config called ENV if it is per default values environment if no environment value is set up, then it will assume it to be development. You can just check the voice dot get in with function details then nothing is provided instead of returning null it will return the default value development. If it is production currently, I am just raising an exception because I had not set up any of these values.

If I really want to read it, I can read this. If nothing is mentioned, or if it is development I am going to start studying local development environment I am going to print that. And then going to import all the config from the object local development config it just read each one of them and then set it up in the config.

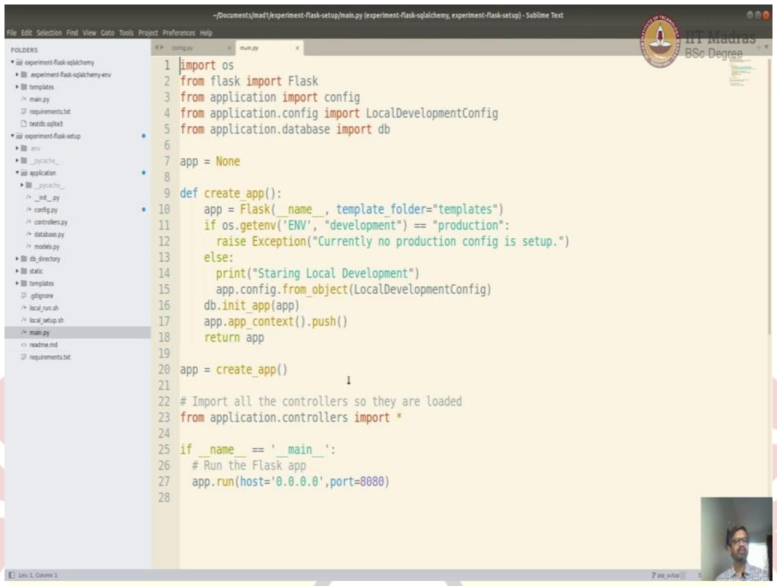
(Refer Slide Time: 10:24)



```
import os
from flask import Flask
from flask import render_template
from flask import request
from flask_sqlalchemy import SQLAlchemy

current_dir = os.path.abspath(os.path.dirname(__file__))

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///" + os.path.join(
    current_dir, "testdb.sqlite3"
)
db = SQLAlchemy()
db.init_app(app)
app.app_context().push()


class User(db.Model):
    __tablename__ = "user"
    user_id = db.Column(db.Integer, autoincrement=True, primary_key=True)
    username = db.Column(db.String, unique=True)
    email = db.Column(db.String, unique=True)


class Article(db.Model):
    __tablename__ = "article"
    article_id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    title = db.Column(db.String)
    content = db.Column(db.String)
    authors = db.relationship("User", secondary="article_authors")
```



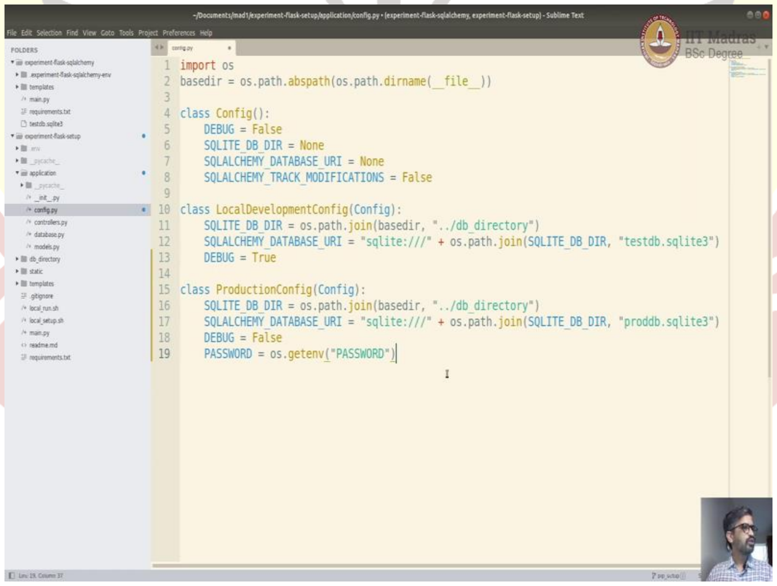
```
import os
basedir = os.path.abspath(os.path.dirname(__file__))

class Config():
    DEBUG = False
    SQLITE_DB_DIR = None
    SQLALCHEMY_DATABASE_URI = None
    SQLALCHEMY_TRACK_MODIFICATIONS = False

class LocalDevelopmentConfig(Config):
    SQLITE_DB_DIR = os.path.join(basedir, "../db_directory")
    SQLALCHEMY_DATABASE_URI = "sqlite:///" + os.path.join(SQLITE_DB_DIR, "testdb.sqlite3")
    DEBUG = True

class ProductionConfig(Config):
    SQLITE_DB_DIR = os.path.join(basedir, "../db_directory")
    SQLALCHEMY_DATABASE_URI = "sqlite:///" + os.path.join(SQLITE_DB_DIR, "proddb.sqlite3")
    DEBUG = False
```



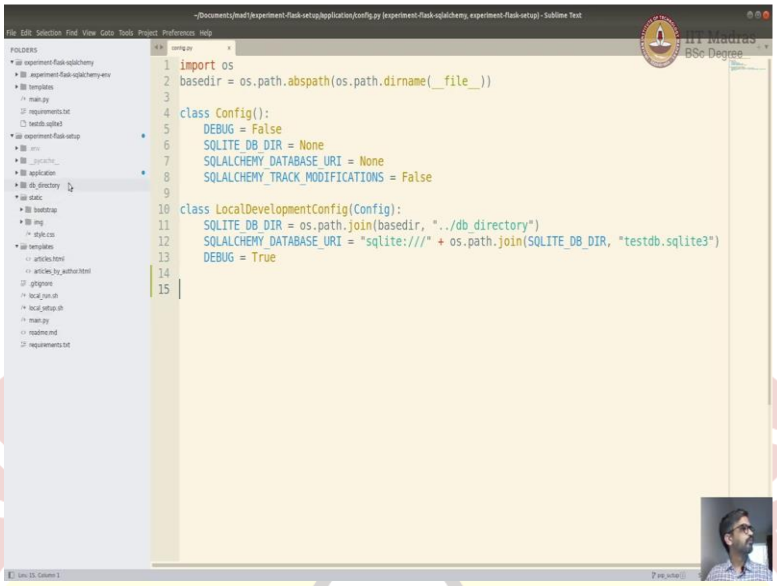
```python
import os
from flask import Flask
from application import config
from application.config import LocalDevelopmentConfig
from application.database import db

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)
    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)
```

Like we have learned in the previous one for SQL alchemy to work we have to set up SQL alchemy and commit database URI into the config. Now, here in the previous experiment, we were setting it up directly. Now we are not doing that we are actually reading that from setting that up in the config and then reading it from the config in the main.py. That makes it easy to change and maintain all the config in the setup.

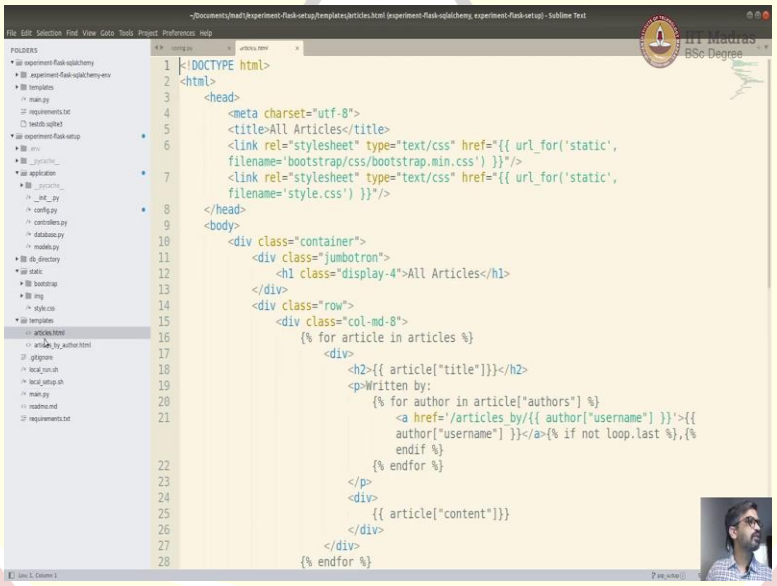
(Refer Slide Time: 10:52)



```python
import os
basedir = os.path.abspath(os.path.dirname(__file__))

class Config():
    DEBUG = False
    SQLITE_DB_DIR = None
    SQLALCHEMY_DATABASE_URI = None
    SQLALCHEMY_TRACK_MODIFICATIONS = False

class LocalDevelopmentConfig(Config):
    SQLITE_DB_DIR = os.path.join(basedir, "../db_directory")
    SQLALCHEMY_DATABASE_URI = "sqlite:///" + os.path.join(SQLITE_DB_DIR, "testdb.sqlite3")
    DEBUG = True

class ProductionConfig(Config):
    SQLITE_DB_DIR = os.path.join(basedir, "../db_directory")
    SQLALCHEMY_DATABASE_URI = "sqlite:///" + os.path.join(SQLITE_DB_DIR, "proddb.sqlite3")
    DEBUG = False
    PASSWORD = os.getenv("PASSWORD")
```



```python
import os
basedir = os.path.abspath(os.path.dirname(__file__))

class Config():
    DEBUG = False
    SQLITE_DB_DIR = None
    SQLALCHEMY_DATABASE_URI = None
    SQLALCHEMY_TRACK_MODIFICATIONS = False

class LocalDevelopmentConfig(Config):
    SQLITE_DB_DIR = os.path.join(basedir, "../db_directory")
    SQLALCHEMY_DATABASE_URI = "sqlite:///" + os.path.join(SQLITE_DB_DIR, "testdb.sqlite3")
    DEBUG = True

```



```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>All Articles</title>
        <link rel="stylesheet" type="text/css" href="{{ url_for('static',
        filename='bootstrap/css/bootstrap.min.css') }}"/>
        <link rel="stylesheet" type="text/css" href="{{ url_for('static',
        filename='style.css') }}"/>
    </head>
    <body>
        <div class="container">
            <div class="jumbotron">
                <h1 class="display-4">All Articles</h1>
            </div>
            <div class="row">
                <div class="col-md-8">
                    {% for article in articles %}
                        <div>
                            <h2>{{ article["title"]}}</h2>
                            <p>Written by:
                                {% for author in article["authors"] %}
                                    <a href='/articles_by/{{ author["username"] }}'>{{
                                    author["username"] }}</a>{% if not loop.last %},{%
                                    endif %}
                                {% endfor %}
                            </p>
                            <div>
                                {{ article["content"]}}
                            </div>
                        </div>
                    {% endfor %}
```

Sometimes, for example if you had if you are probably accessing Postgres database, and you want to read the password and let us say some variable like this password, and you may not want to actually put it into the code you can actually get again do os.getenv(). And then read it from the environment instead of actually putting into this file. So, that you do not have to commit passwords into your an object or saving it into your local file.

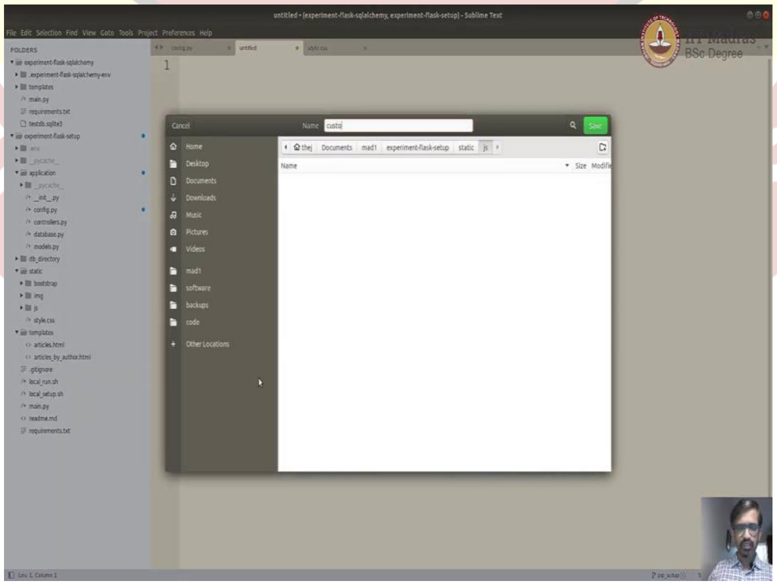
Because it is generally a bad idea to put passwords into your code inside your code do not directly put like I do not want to put x let us say your password is my secret password. And you do not want to put it here you really want to read it from the os.genenv(). And just say password it will read it from environment and set it up so, that your code does not have any passwords. And it has rare chance of leaking it then. That is one of the uses of this.

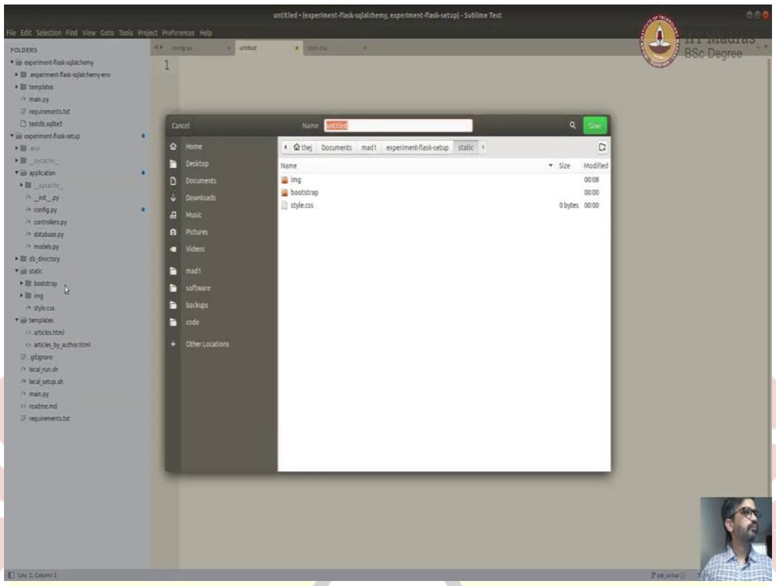
So, now I am just going to remove this because you are just going to experiment in local in config. And since we do not have any real secrets I am not reading anything from OS environment or any other environment conflicts. So, I am not reading it. I am just keep the debugger as true because I just want to see it. And the local development. So, now you are learnt all of it and then templates is in the template folder.

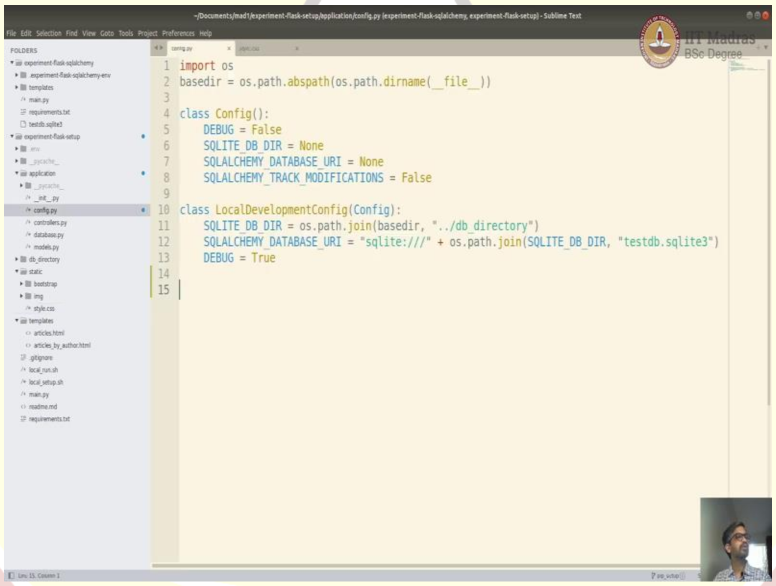
Again, you can inside the template, you can create sub directories for each of the type and organize the code the way you want and make it more meaningful in terms of organizing. If not in this case, it is pretty straightforward to directly put it under Templates folder. Now, there is a static folder. By default Python application or rather flask application serves all static files from a folder called static or from a route called static.

So, whatever you add into static folder can be directly accessed in your file. For example, like this like for example, instead of directly taking it from bootstrap, I am actually reading it from my local static folder. So, you can check for a URI for static which gives you a URI for static slash static router. And then, I am linking it to this file so, that it you can import it. Similarly, for local style if you have local style.

(Refer Slide Time: 13:45)



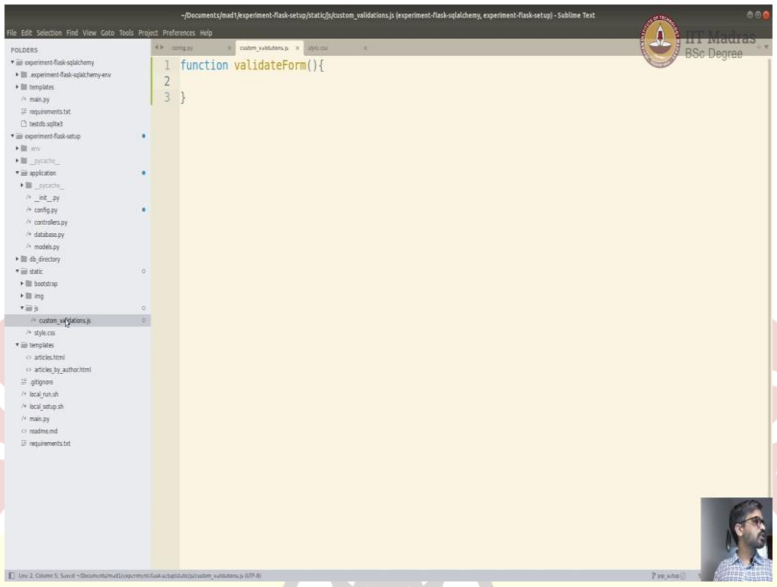

untitled • (experiment-flask-sqlalchemy, experiment-flask-setup) - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
Cancel
Name untitled
Save
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
mad1
software
backups
code
Other Locations
thej Documents mad1 experiment-flask-setup static
Name Size Modified
img
bootstrap
style.css
IIT Madras BSc Degree


~/Documents/mad1/experiment-flask-setup/application/config.py (experiment-flask-sqlalchemy, experiment-flask-setup) - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
config.py
style.css
IIT Madras BSc Degree

```python
import os
basedir = os.path.abspath(os.path.dirname(__file__))

class Config():
    DEBUG = False
    SQLITE_DB_DIR = None
    SQLALCHEMY_DATABASE_URI = None
    SQLALCHEMY_TRACK_MODIFICATIONS = False

class LocalDevelopmentConfig(Config):
    SQLITE_DB_DIR = os.path.join(basedir, "../db_directory")
    SQLALCHEMY_DATABASE_URI = "sqlite:///" + os.path.join(SQLITE_DB_DIR, "testdb.sqlite3")
    DEBUG = True

```

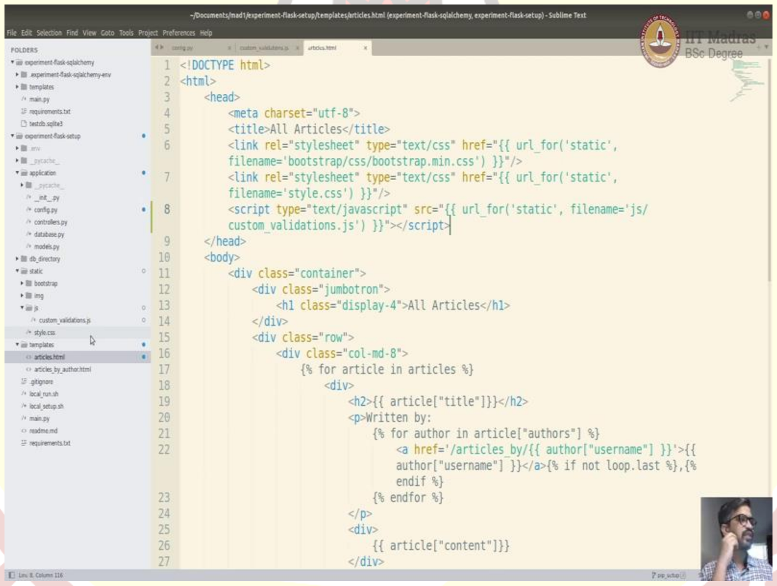
I have not I have a file here, I am not putting anything but if you want to overwrite and put your own styles you can put it here. Similarly, you can like for example, if you are writing some JavaScript you can let us say create a new it will say JavaScript folder, js folder. Let us say on not file folder just to keep it cleaner, I will create a folder called js inside that I will say let us say custom. Let us say we have written some validationjs.js.

(Refer Slide Time: 14:32)



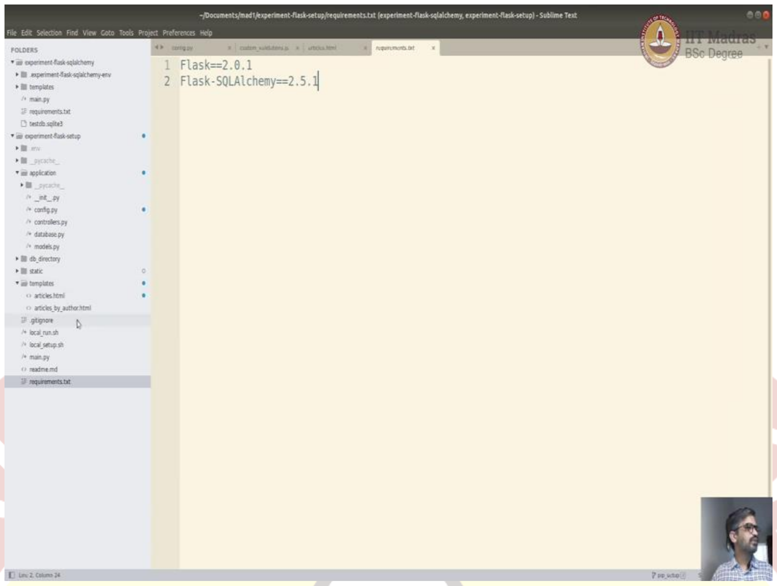
~/Documents/mad1/experiment-flask-setup/static/js/custom_validations.js (experiment-flask-sqlalchemy, experiment-flask-setup) - Sublime Text

```
function validateForm(){

}
```



~/Documents/mad1/experiment-flask-setup/templates/articles.html (experiment-flask-sqlalchemy, experiment-flask-setup) - Sublime Text

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>All Articles</title>
        <link rel="stylesheet" type="text/css" href="{{ url_for('static',
        filename='bootstrap/css/bootstrap.min.css') }}"/>
        <link rel="stylesheet" type="text/css" href="{{ url_for('static',
        filename='style.css') }}"/>
        <script type="text/javascript" src="{{ url_for('static', filename='js/
        custom_validations.js') }}"></script>
    </head>
    <body>
        <div class="container">
            <div class="jumbotron">
                <h1 class="display-4">All Articles</h1>
            </div>
            <div class="row">
                <div class="col-md-8">
                    {% for article in articles %}
                        <div>
                            <h2>{{ article["title"]}}</h2>
                            <p>Written by:
                                {% for author in article["authors"] %}
                                    <a href='/articles_by/{{ author["username"] }}'>{{
                                    author["username"] }}</a>{% if not loop.last %},{%
                                    endif %}
                                {% endfor %}
                            </p>
                            <div>
                                {{ article["content"]}}
                            </div>
```



```
Flask==2.0.1
Flask-SQLAlchemy==2.5.1
```

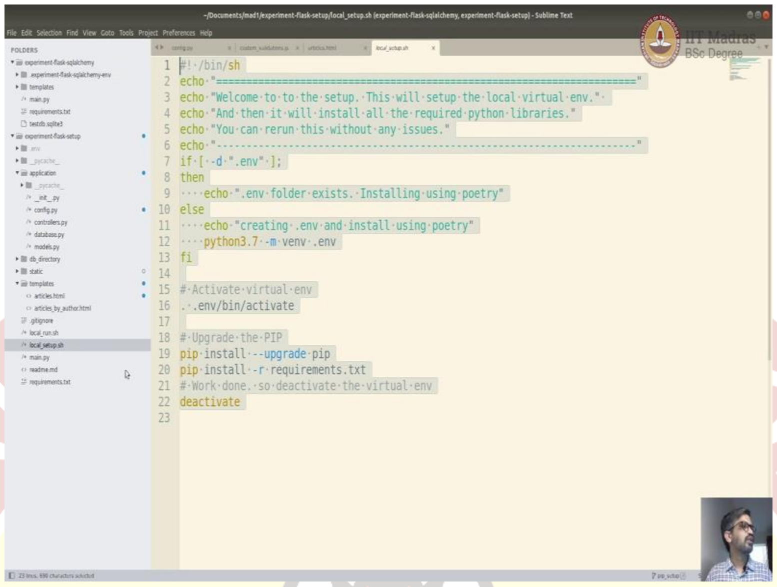
I am just maybe we had a function called validate form. I am not going to do anything but I am just saying. I can just write console.log but I am not going to do anything, just leave it like that. Now, if you really want to use this inside your template. It is straightforward to do it, you can just you can write your script tag source. You can actually do this itself. Path for static but file is not style css.

It is js slash lets what is the filename, custom validations. I am just going to copy it. Custom validation structures. So, you can include and the URI will be correct and it will be loaded from locally from your static folder. That is how we use static folder. Of course, you can host it somewhere else on a cdm or somewhere. But for a simple flask app you can you can still use it to serve.

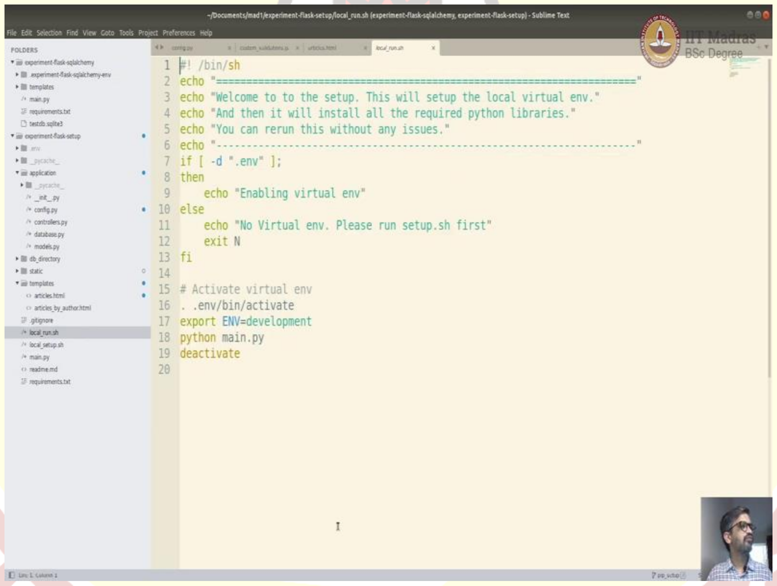
So, this is done. And then what are the files are left which as usual have a requirements.txt, where I mentioned all the requirements flask and flask SQL alchemy here are pinned to the version which version I want to use. And if you do not mention it, it will use the latest version. I think it is always important to put the version that you stick to a specific version it makes it easy to debug.

And if you want to compare your and your teammates setups, it becomes easy to compare and check and stuff like that. So, it is always easy to do that. Now locally to do a run or to set up my project I have written to bash script.

(Refer Slide Time: 16:38)



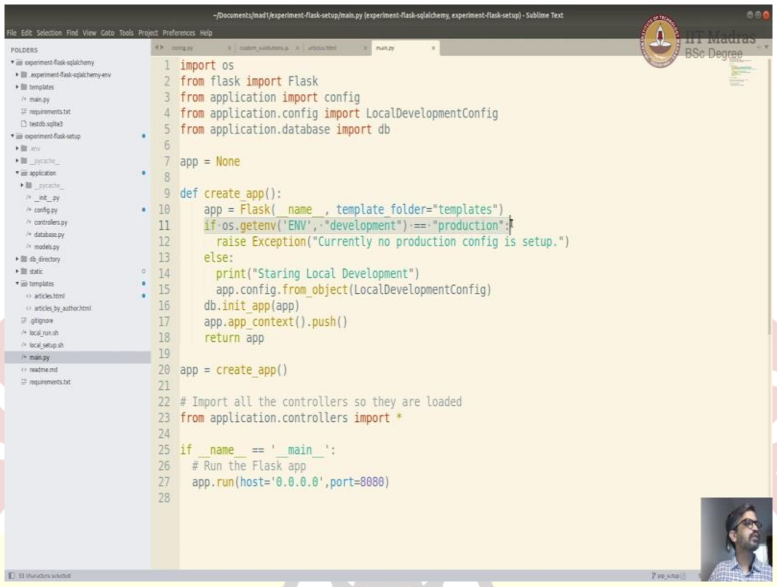
```
#! /bin/sh
echo "======================================================================"
echo "Welcome to to the setup. This will setup the local virtual env."
echo "And then it will install all the required python libraries."
echo "You can rerun this without any issues."
echo "----------------------------------------------------------------------"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



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

# Activate virtual env
. .env/bin/activate
export ENV=development
python main.py
deactivate
```

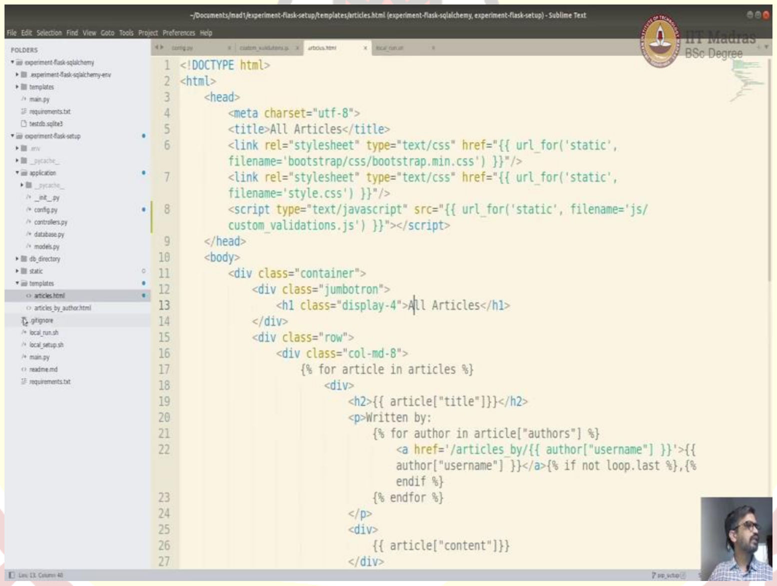
You do not have to write it we because we always keep repeating activating setting up the environment like you would have seen previously. I have just written a script for that, and similarly for run. But I will show you without running these you can directly run by writing the commands directly instead of running this sh but we can share this sh files or share files for you to make it easy to experiment or.

(Refer Slide Time: 17:11)



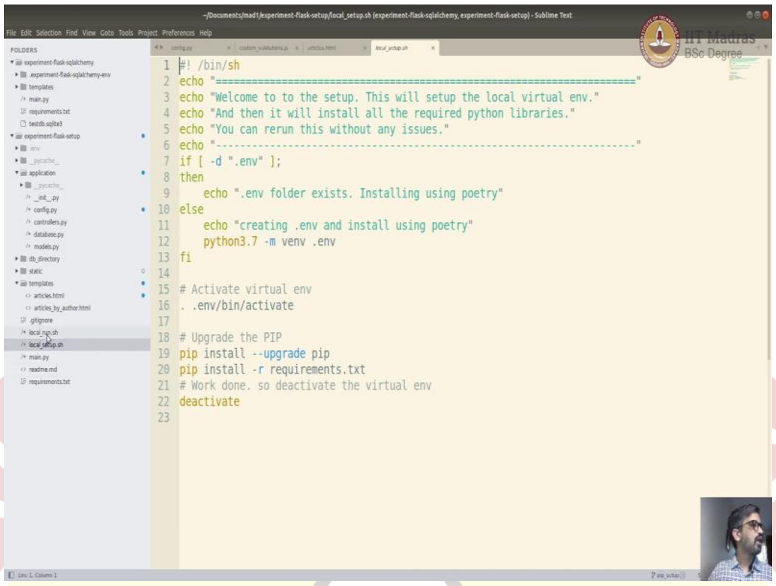
~/Documents/mad1/experiment-flask-setup/main.py (experiment-flask-sqlalchemy, experiment-flask-setup) - Sublime Text

```python
import os
from flask import Flask
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)
    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-setup/templates/articles.html (experiment-flask-sqlalchemy, experiment-flask-setup) - Sublime Text

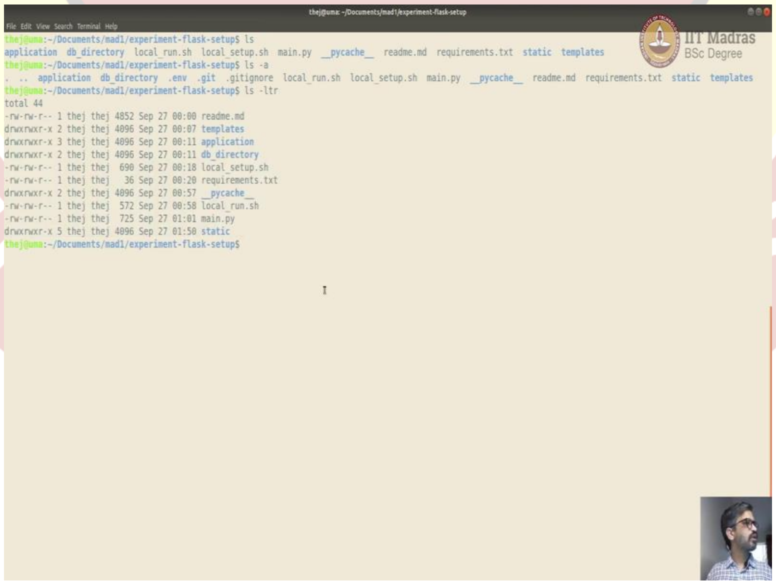
```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>All Articles</title>
        <link rel="stylesheet" type="text/css" href="{{ url_for('static',
        filename='bootstrap/css/bootstrap.min.css') }}"/>
        <link rel="stylesheet" type="text/css" href="{{ url_for('static',
        filename='style.css') }}"/>
        <script type="text/javascript" src="{{ url_for('static', filename='js/
        custom_validations.js') }}"></script>
    </head>
    <body>
        <div class="container">
            <div class="jumbotron">
                <h1 class="display-4">All Articles</h1>
            </div>
            <div class="row">
                <div class="col-md-8">
                    {% for article in articles %}
                        <div>
                            <h2>{{ article["title"]}}</h2>
                            <p>Written by:
                                {% for author in article["authors"] %}
                                    <a href='/articles_by/{{ author["username"] }}'>{{
                                    author["username"] }}</a>{% if not loop.last %},{%
                                    endif %}
                                {% endfor %}
                            </p>
                            <div>
                                {{ article["content"]}}
                            </div>
```



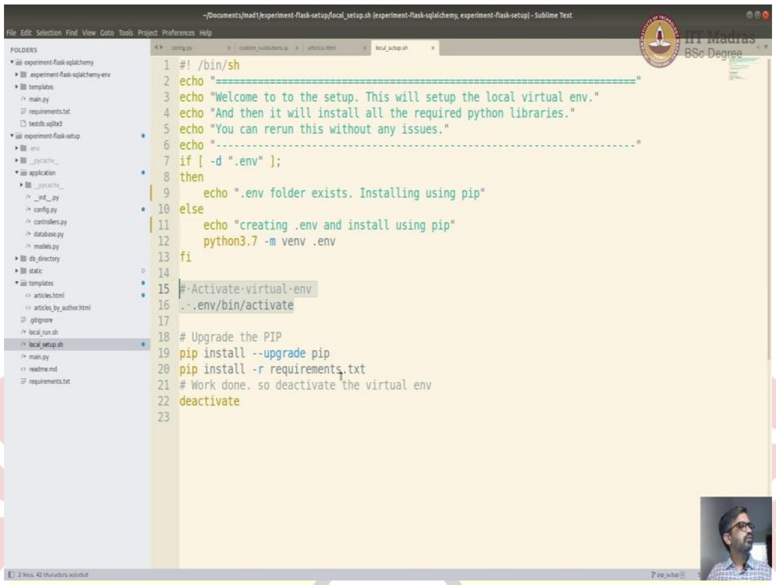
```
#! /bin/sh
echo "======================================================================"
echo "Welcome to to the setup. This will setup the local virtual env."
echo "And then it will install all the required python libraries."
echo "You can rerun this without any issues."
echo "----------------------------------------------------------------------"
if [ -d ".env" ];
then
    echo ".env folder exists. Installing using poetry"
else
    echo "creating .env and install using poetry"
    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, now let us go back to one last thing. So, here is where we are actually reading the environment. So, when you actually run I am setting it up as development here you can see export ENV =development. So, this gets read here in the main.py at sets up the configuration. Now if I actually wanted to read it run in production or any other environment, I will go back and set up a environment as per that specific environment and run it. So, now we can just run it just to try it locally.

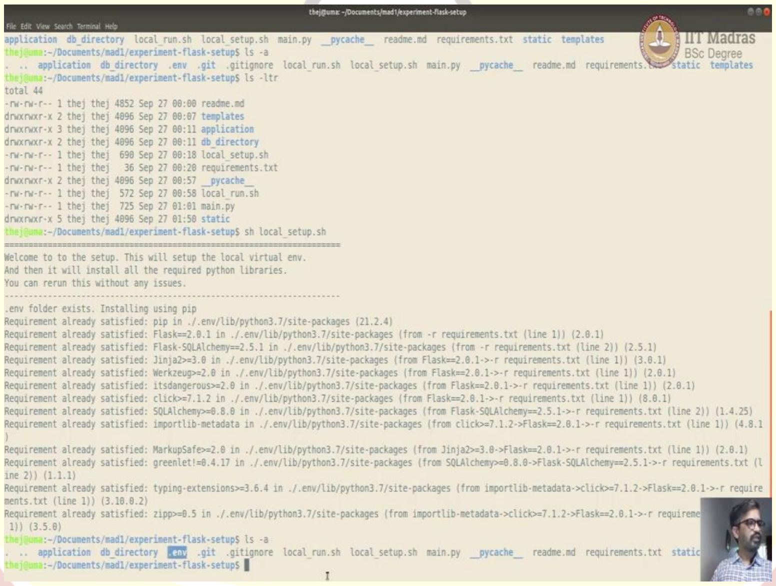
(Refer Slide Time: 17:48)



```
thej@uma:~/Documents/mad1/experiment-flask-setup$ ls
application  db_directory  local_run.sh  local_setup.sh  main.py  __pycache__  readme.md  requirements.txt  static  templates
thej@uma:~/Documents/mad1/experiment-flask-setup$ ls -a
.  ..  application  db_directory  .env  .git  .gitignore  local_run.sh  local_setup.sh  main.py  __pycache__  readme.md  requirements.txt  static  templates
thej@uma:~/Documents/mad1/experiment-flask-setup$ ls -ltr
total 44
-rw-rw-r-- 1 thej thej 4852 Sep 27 00:00 readme.md
drwxrwxr-x 2 thej thej 4096 Sep 27 00:07 templates
drwxrwxr-x 3 thej thej 4096 Sep 27 00:11 application
drwxrwxr-x 2 thej thej 4096 Sep 27 00:11 db_directory
-rw-rw-r-- 1 thej thej  690 Sep 27 00:18 local_setup.sh
-rw-rw-r-- 1 thej thej   36 Sep 27 00:20 requirements.txt
drwxrwxr-x 2 thej thej 4096 Sep 27 00:57 __pycache__
-rw-rw-r-- 1 thej thej  572 Sep 27 00:58 local_run.sh
-rw-rw-r-- 1 thej thej  725 Sep 27 01:01 main.py
drwxrwxr-x 5 thej thej 4096 Sep 27 01:50 static
thej@uma:~/Documents/mad1/experiment-flask-setup$
```



```
#! /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
```

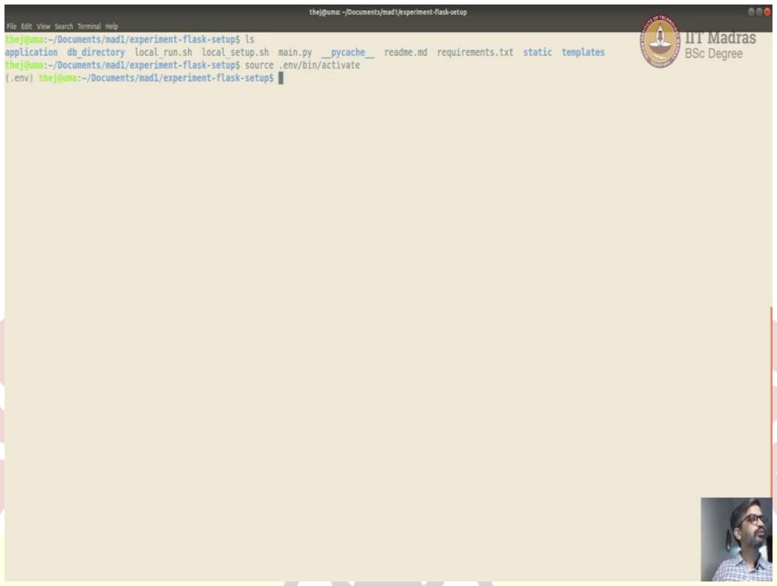


```
application  db_directory  local_run.sh  local_setup.sh  main.py  __pycache__  readme.md  requirements.txt  static  templates
thej@uma:~/Documents/mad1/experiment-flask-setup$ ls -a
.  ..  application  db_directory  .env  .git  .gitignore  local_run.sh  local_setup.sh  main.py  __pycache__  readme.md  requirements.txt  static  templates
thej@uma:~/Documents/mad1/experiment-flask-setup$ ls -ltr
total 44
-rw-rw-r-- 1 thej thej 4852 Sep 27 00:00 readme.md
drwxrwxr-x 2 thej thej 4096 Sep 27 00:07 templates
drwxrwxr-x 3 thej thej 4096 Sep 27 00:11 application
drwxrwxr-x 2 thej thej 4096 Sep 27 00:11 db_directory
-rw-rw-r-- 1 thej thej  690 Sep 27 00:18 local_setup.sh
-rw-rw-r-- 1 thej thej   36 Sep 27 00:20 requirements.txt
drwxrwxr-x 2 thej thej 4096 Sep 27 00:57 __pycache__
-rw-rw-r-- 1 thej thej  572 Sep 27 00:58 local_run.sh
-rw-rw-r-- 1 thej thej  725 Sep 27 01:01 main.py
drwxrwxr-x 5 thej thej 4096 Sep 27 01:50 static
thej@uma:~/Documents/mad1/experiment-flask-setup$ 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.
----------------------------------------------------------------------
.env folder exists. Installing using pip
Requirement already satisfied: pip in ./.env/lib/python3.7/site-packages (21.2.4)
Requirement already satisfied: Flask==2.0.1 in ./.env/lib/python3.7/site-packages (from -r requirements.txt (line 1)) (2.0.1)
Requirement already satisfied: Flask-SQLAlchemy==2.5.1 in ./.env/lib/python3.7/site-packages (from -r requirements.txt (line 2)) (2.5.1)
Requirement already satisfied: Jinja2>=3.0 in ./.env/lib/python3.7/site-packages (from Flask==2.0.1->-r requirements.txt (line 1)) (3.0.1)
Requirement already satisfied: Werkzeug>=2.0 in ./.env/lib/python3.7/site-packages (from Flask==2.0.1->-r requirements.txt (line 1)) (2.0.1)
Requirement already satisfied: itsdangerous>=2.0 in ./.env/lib/python3.7/site-packages (from Flask==2.0.1->-r requirements.txt (line 1)) (2.0.1)
Requirement already satisfied: click>=7.1.2 in ./.env/lib/python3.7/site-packages (from Flask==2.0.1->-r requirements.txt (line 1)) (8.0.1)
Requirement already satisfied: SQLAlchemy>=0.8.0 in ./.env/lib/python3.7/site-packages (from Flask-SQLAlchemy==2.5.1->-r requirements.txt (line 2)) (1.4.25)
Requirement already satisfied: importlib-metadata in ./.env/lib/python3.7/site-packages (from click>=7.1.2->Flask==2.0.1->-r requirements.txt (line 1)) (4.8.1
)
Requirement already satisfied: MarkupSafe>=2.0 in ./.env/lib/python3.7/site-packages (from Jinja2>=3.0->Flask==2.0.1->-r requirements.txt (line 1)) (2.0.1)
Requirement already satisfied: greenlet!=0.4.17 in ./.env/lib/python3.7/site-packages (from SQLAlchemy>=0.8.0->Flask-SQLAlchemy==2.5.1->-r requirements.txt (l
ine 2)) (1.1.1)
Requirement already satisfied: typing-extensions>=3.6.4 in ./.env/lib/python3.7/site-packages (from importlib-metadata->click>=7.1.2->Flask==2.0.1->-r require
ments.txt (line 1)) (3.10.0.2)
Requirement already satisfied: zipp>=0.5 in ./.env/lib/python3.7/site-packages (from importlib-metadata->click>=7.1.2->Flask==2.0.1->-r requireme
 1)) (3.5.0)
thej@uma:~/Documents/mad1/experiment-flask-setup$ ls -a
.  ..  application  db_directory  .env  .git  .gitignore  local_run.sh  local_setup.sh  main.py  __pycache__  readme.md  requirements.txt  static
thej@uma:~/Documents/mad1/experiment-flask-setup$
```

Already set up the environment. Like you can check, let me just say ls - ltr you can see that I have not set up environment. So, let us set up environment and run it. So, here, let me just check in local setup here you can see that it checks for a folder called dot env if the folder exists. Now, I am written installing using poetry. But it is just not poetry. And then creating a virtual environment inside that dot env if the folder does not exist then activating it.

And then installing all the requirements, we can do it manually as well. Or we can do it using this script. So, let me just do it using this script. So, sh local setup. So, it created a dot env folder. It seems like it already exists. So, it is just up reinstalling everything. So, we can check ls – a.env exists.

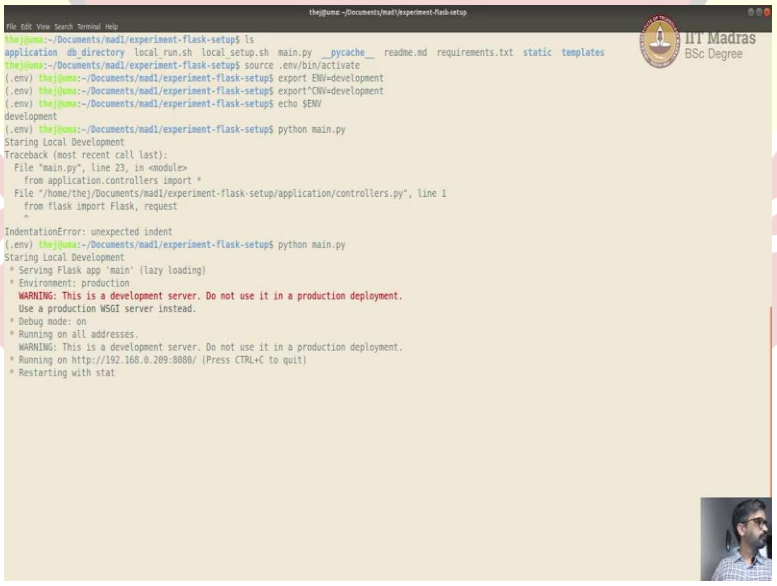
(Refer Slide Time: 19:06)



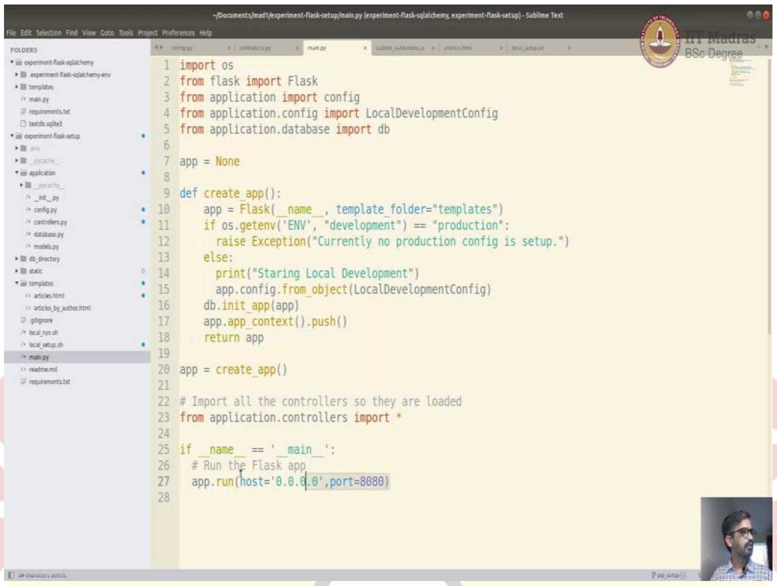
```
thej@uma:~/Documents/mad1/experiment-flask-setup$ ls
application  db_directory  local_run.sh  local_setup.sh  main.py  __pycache__  readme.md  requirements.txt  static  templates
thej@uma:~/Documents/mad1/experiment-flask-setup$ source .env/bin/activate
(.env) thej@uma:~/Documents/mad1/experiment-flask-setup$ 
```

Now, you can again to run you can actually just activate it and set up the environment and run. So, let us do it. I am going to just do it manually instead of running local run just to make sure that we all know what we are doing. So, I just have to source.env virtual environment. Bin activate. That is how you source it. It is enabled now. And then we have to export the environment as development because we want development config to be used.

(Refer Slide Time: 19:44)



```
thej@uma:~/Documents/mad1/experiment-flask-setup$ ls
application  db_directory  local_run.sh  local_setup.sh  main.py  __pycache__  readme.md  requirements.txt  static  templates
thej@uma:~/Documents/mad1/experiment-flask-setup$ source .env/bin/activate
(.env) thej@uma:~/Documents/mad1/experiment-flask-setup$ export ENV=development
(.env) thej@uma:~/Documents/mad1/experiment-flask-setup$ export^CNV=development
(.env) thej@uma:~/Documents/mad1/experiment-flask-setup$ echo $ENV
development
(.env) thej@uma:~/Documents/mad1/experiment-flask-setup$ python main.py
Staring Local Development
Traceback (most recent call last):
  File "main.py", line 23, in <module>
    from application.controllers import *
  File "/home/thej/Documents/mad1/experiment-flask-setup/application/controllers.py", line 1
    from flask import Flask, request
    ^
IndentationError: unexpected indent
(.env) thej@uma:~/Documents/mad1/experiment-flask-setup$ python main.py
Staring Local Development
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://192.168.0.209:8080/ (Press CTRL+C to quit)
 * Restarting with stat
```

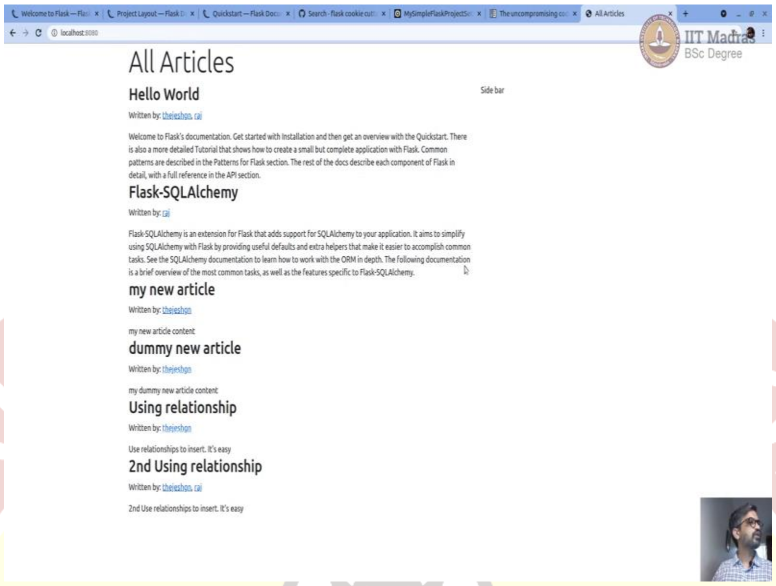


```python
import os
from flask import Flask
from application import config
from application.config import LocalDevelopmentConfig
from application.database import db

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)
    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)
```

So, I am going to do export ENV = development. If you want to check whether it is set up properly then you can just do echo develop env. So, you can see that it is printing the value development. Now we have set I just have to run it to run it I just have to do python main.py. So, there is some indentation problem let us go check it. I must by mistake done something. That mistake.

So, it is you can see it is printing a warning we are starting a development environment do not use it in production. It is also because we are using the default development server on the flask. in production, we will be using a different kind of so. So, it is active running locally on port 8080 because that is what we have asked it to run here.

(Refer Slide Time: 20:52)



All Articles

Hello World

Written by: thejeshgn, 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: thejeshgn

my new article content

dummy new article

Written by: thejeshgn

my dummy new article content

Using relationship

Written by: thejeshgn

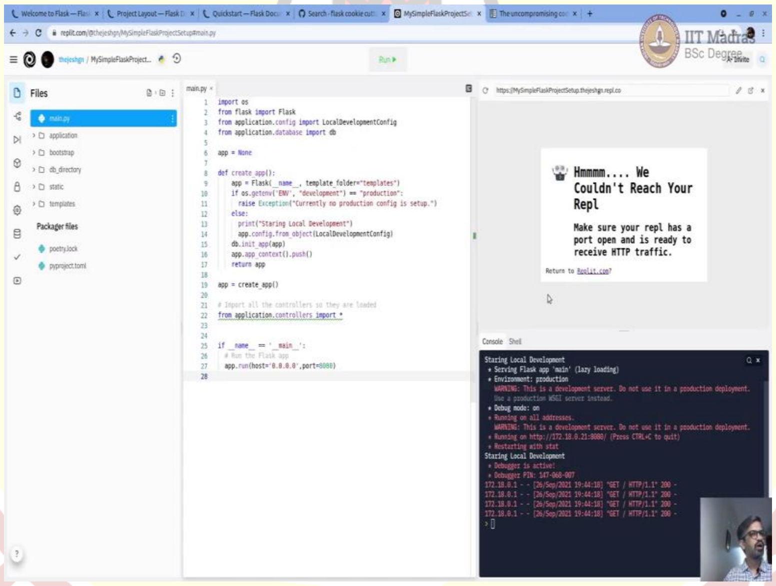
Use relationships to insert. It's easy

2nd Using relationship

Written by: thejeshgn, raj

2nd Use relationships to insert. It's easy

Side bar



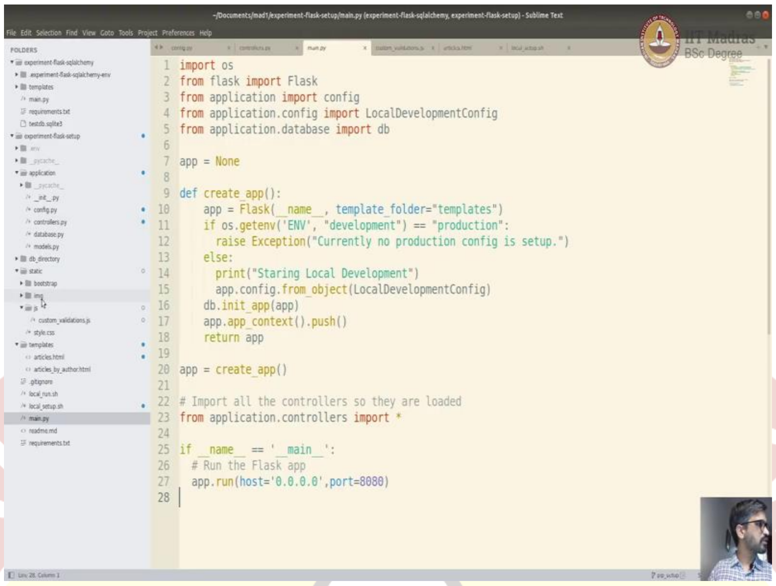
```
import os
from flask import Flask
from application.config import LocalDevelopmentConfig
from application.database import db

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)
    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)
```

Hmmmm.... We Couldn't Reach Your Repl

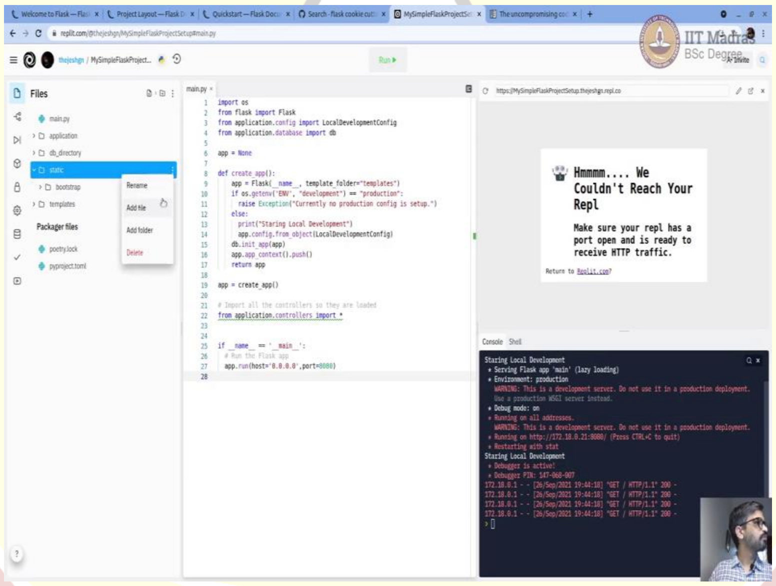
Make sure your repl has a port open and is ready to receive HTTP traffic.

Return to Replit.com?

```
Staring Local Development
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.18.0.21:8080/ (Press CTRL+C to quit)
 * Restarting with stat
Staring Local Development
 * Debugger is active!
 * Debugger PIN: 147-068-007
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
> 
```



```python
import os
from flask import Flask
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)
    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)
```



Hmmmm.... We Couldn't Reach Your Repl

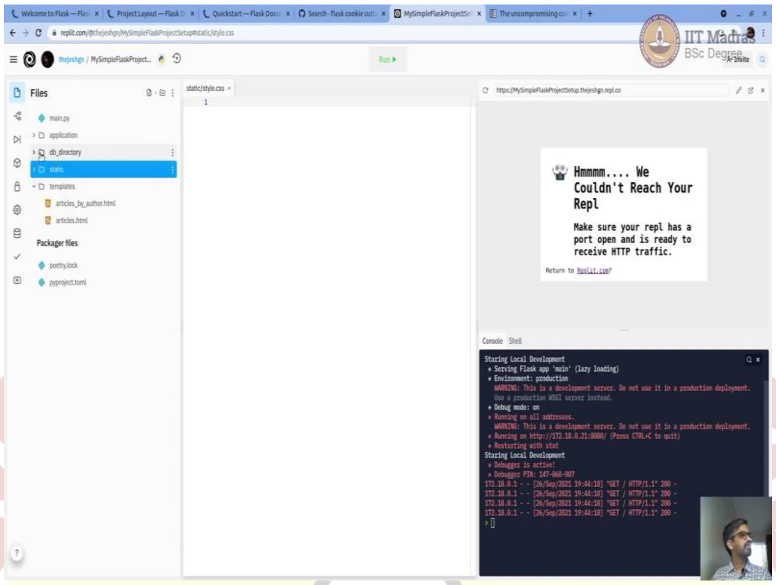
Make sure your repl has a port open and is ready to receive HTTP traffic.

Return to Replit.com?

```
Staring Local Development
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.18.0.21:8080/ (Press CTRL+C to quit)
 * Restarting with stat
Staring Local Development
 * Debugger is active!
 * Debugger PIN: 147-068-907
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
> 
```

So, I am just going to move it. Let us see. Where did it go? Not here it is got moved it took some time. It is just a network issue. So, it got moved inside it this is correct part of it. I have not put other unused files here. But you can upload them or create new files like style.css like you can just create add file was it in this path? Or was it inside css no style css directly under. So, let me just create add file style css. That is it.

(Refer Slide Time: 22:29)

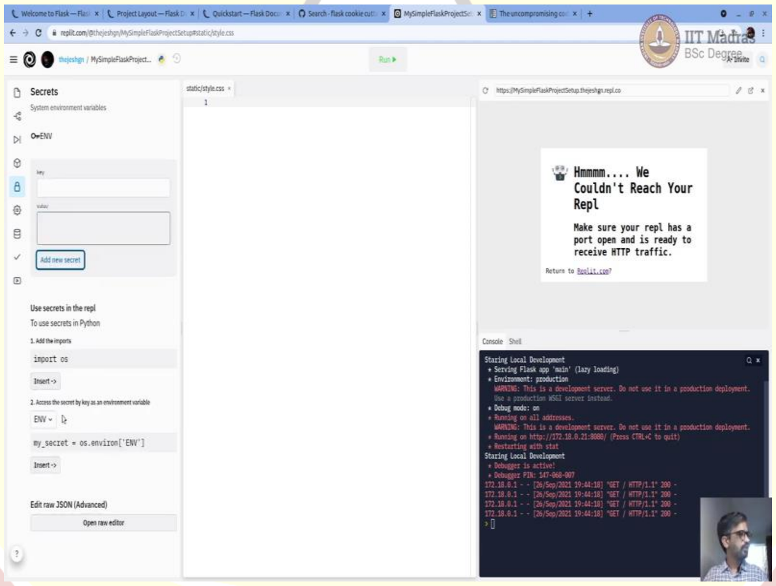


Hmmmm.... We Couldn't Reach Your Repl

Make sure your repl has a port open and is ready to receive HTTP traffic.

Return to Replit.com?

```
Staring Local Development
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.18.0.21:8080/ (Press CTRL+C to quit)
 * Restarting with stat
Staring Local Development
 * Debugger is active!
 * Debugger PIN: 147-068-007
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
> []
```


replit.com/@thejeshgn/MySimpleFlaskProjectSetup#static/style.css
thejeshgn / MySimpleFlaskProject...
Run ▸
Secrets
System environment variables
ENV
key
value
Add new secret
Insert ->
Insert ->
Open raw editor
static/style.css
Console
Shell

Use secrets in the repl

To use secrets in Python

1. Add the imports

```
import os
```

2. Access the secret by key as an environment variable

ENV

```
my_secret = os.environ['ENV']
```

Edit raw JSON (Advanced)

Hmmmm.... We Couldn't Reach Your Repl

Make sure your repl has a port open and is ready to receive HTTP traffic.

Return to Replit.com?

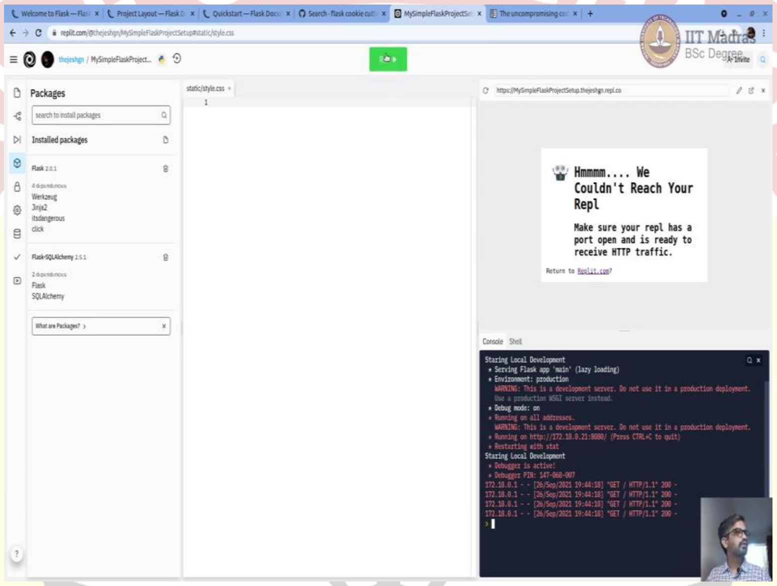
```
Staring Local Development
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.18.0.21:8080/ (Press CTRL+C to quit)
 * Restarting with stat
Staring Local Development
 * Debugger is active!
 * Debugger PIN: 147-068-007
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
> []
```

I mean I am not going to add anything, if you want to customize something you can. And then there is a template folder with templates. All of it is there and our db is been moved into db directory which is same here our db I have put it under db directory it could be anywhere in on the os and we are just clearly mentioning this is a db directory and putting the db inside that. So, that is why if you look at my config it is in the it says that we get SQL alchemy database URI by joining this string which says SQ lite.

And by joining a SQLite db directory, which is db directory and the name of the file which is SQ lite file. Similarly, now here we are actually setting environment by running export ENV = development. You cannot do that directly. I mean, you can probably but it is better way to set up using here environment variables. So, we can go to environment variables here. I have already set it up you can click I have set it up ENV = development.

But if I delete it it is not that difficult. You can go to environment. Let me click again. So, complete new one ENV set it up to develop and based on your settings, but I am just going to use the same development thing and add it is now, now it is set up.

(Refer Slide Time: 24:10)

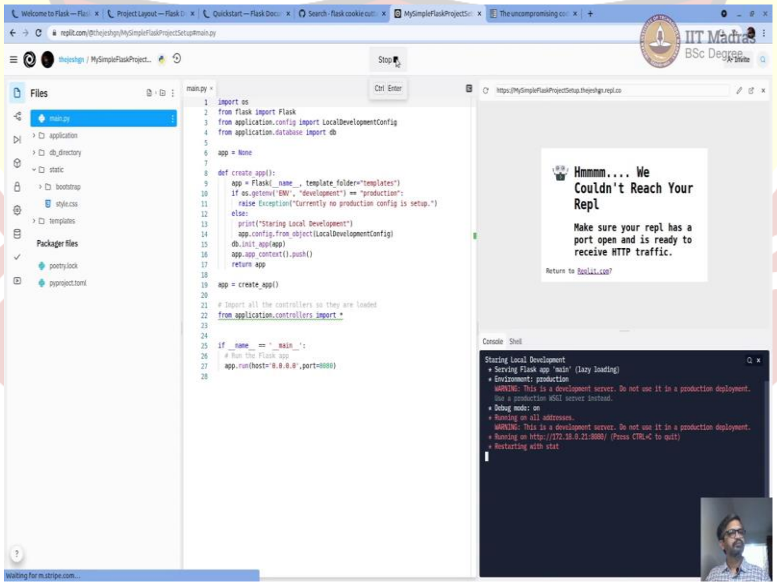


Hmmmm.... We Couldn't Reach Your Repl

Make sure your repl has a port open and is ready to receive HTTP traffic.

Return to Replit.com?

```
Staring Local Development
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.18.0.21:8080/ (Press CTRL+C to quit)
 * Restarting with stat
Staring Local Development
 * Debugger is active!
 * Debugger PIN: 147-068-007
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 19:44:18] "GET / HTTP/1.1" 200 -
>
```



```python
import os
from flask import Flask
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)
    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)
```

Hmmmm.... We Couldn't Reach Your Repl

Make sure your repl has a port open and is ready to receive HTTP traffic.

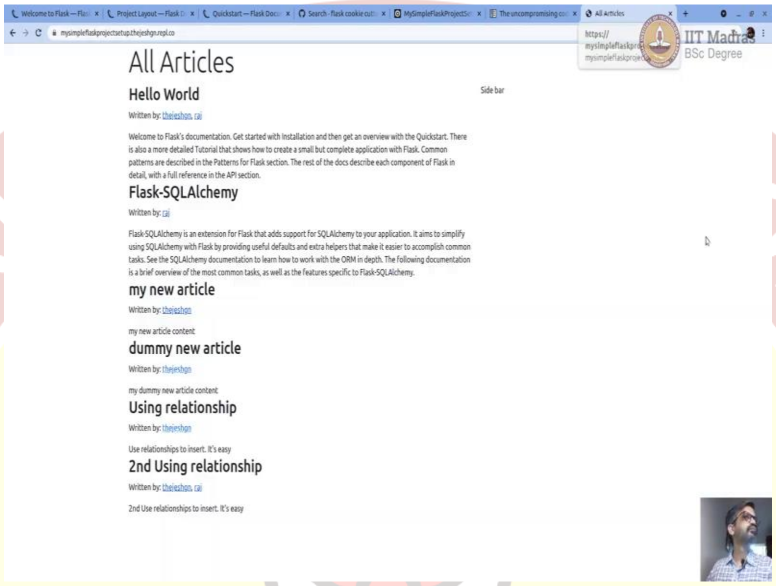
Return to Replit.com?

```
Staring Local Development
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.18.0.21:8080/ (Press CTRL+C to quit)
 * Restarting with stat
```

You can check the values like etcetera. Now you can go back to your file explorer. Now everything is set up like just like how we had it locally. The thing is I have also pre added the packages already like I showed you last time I have already added flask and flask SQL alchemy.

Because it is not like I said it lapel it users poetry to do that. Now I am just going to run it. Let us go back to this main and run. It might take a couple of seconds. It is running. So, you can see that.

(Refer Slide Time: 24:50)



mysimpleflaskprojectsetup.thejeshgn.repl.co

All Articles

Side bar

Hello World

Written by: thejeshgn, 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: thejeshgn

my new article content

dummy new article

Written by: thejeshgn

my dummy new article content

Using relationship

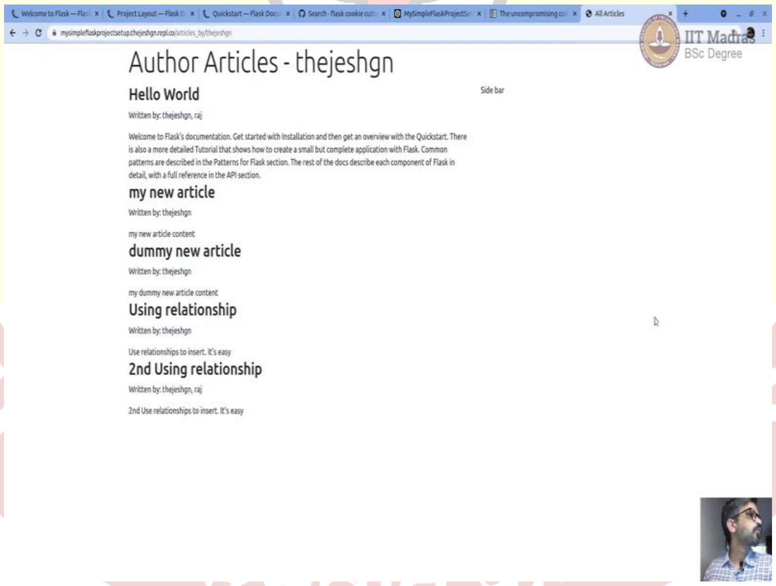
Written by: thejeshgn

Use relationships to insert. It's easy

2nd Using relationship

Written by: thejeshgn, raj

2nd Use relationships to insert. It's easy



mysimpleflaskprojectsetup.thejeshgn.repl.co/articles_by/thejeshgn

Author Articles - thejeshgn

Side bar

Hello World

Written by: thejeshgn, 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.

my new article

Written by: thejeshgn

my new article content

dummy new article

Written by: thejeshgn

my dummy new article content

Using relationship

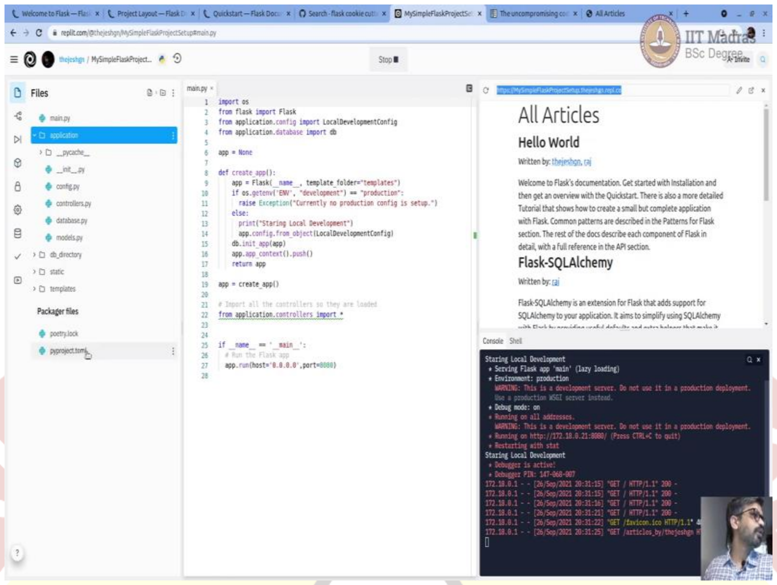
Written by: thejeshgn

Use relationships to insert. It's easy

2nd Using relationship

Written by: thejeshgn, raj

2nd Use relationships to insert. It's easy



main.py

```python
import os
from flask import Flask
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)
    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)
```

All Articles

Hello World

Written by: thejeshgn, raj

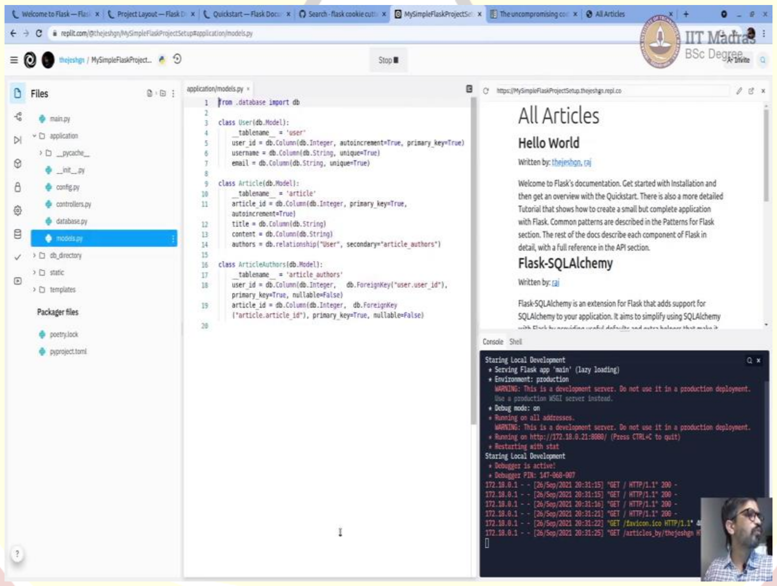
Welcome to Flask's documentation. Get started with Installation and then get an overview with the Quickstart. There is also a more detailed Tutorial that shows how to create a small but complete application with Flask. Common patterns are described in the Patterns for Flask section. The rest of the docs 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

```
Staring Local Development
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.18.0.21:8080/ (Press CTRL+C to quit)
 * Restarting with stat
Staring Local Development
 * Debugger is active!
 * Debugger PIN: 147-068-007
172.18.0.1 - - [26/Sep/2021 20:31:15] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:15] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:16] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:21] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:22] "GET /favicon.ico HTTP/1.1" 4
172.18.0.1 - - [26/Sep/2021 20:31:25] "GET /articles_by/thejeshgn H
```



application/models.py

```python
from .database import db

class User(db.Model):
    __tablename__ = 'user'
    user_id = db.Column(db.Integer, autoincrement=True, primary_key=True)
    username = db.Column(db.String, unique=True)
    email = db.Column(db.String, unique=True)

class Article(db.Model):
    __tablename__ = 'article'
    article_id = db.Column(db.Integer, primary_key=True,
    autoincrement=True)
    title = db.Column(db.String)
    content = db.Column(db.String)
    authors = db.relationship("User", secondary="article_authors")

class ArticleAuthors(db.Model):
    __tablename__ = 'article_authors'
    user_id = db.Column(db.Integer,   db.ForeignKey("user.user_id"),
    primary_key=True, nullable=False)
    article_id = db.Column(db.Integer,  db.ForeignKey
    ("article.article_id"), primary_key=True, nullable=False)
```

All Articles

Hello World

Written by: thejeshgn, raj

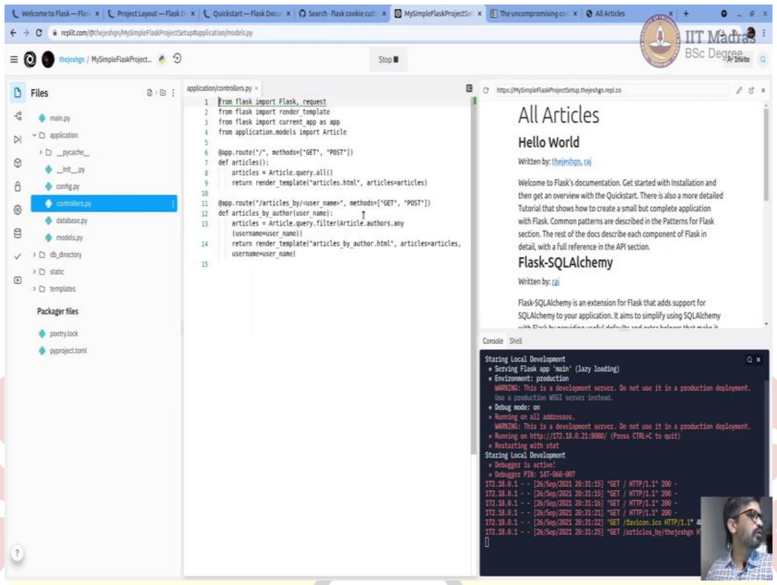
Welcome to Flask's documentation. Get started with Installation and then get an overview with the Quickstart. There is also a more detailed Tutorial that shows how to create a small but complete application with Flask. Common patterns are described in the Patterns for Flask section. The rest of the docs 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

```
Staring Local Development
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.18.0.21:8080/ (Press CTRL+C to quit)
 * Restarting with stat
Staring Local Development
 * Debugger is active!
 * Debugger PIN: 147-068-007
172.18.0.1 - - [26/Sep/2021 20:31:15] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:15] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:16] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:21] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:22] "GET /favicon.ico HTTP/1.1" 4
172.18.0.1 - - [26/Sep/2021 20:31:25] "GET /articles_by/thejeshgn H
```



```
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_name)
```

All Articles

Hello World

Written by: thejeshgn, 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

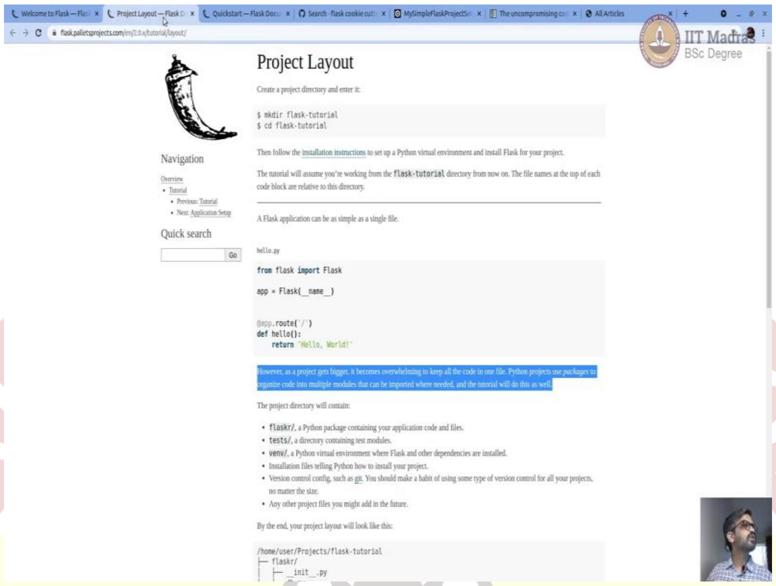
Flask-SQLAlchemy is an extension for Flask that adds support for SQLAlchemy to your application. It aims to simplify using SQLAlchemy

```
Staring Local Development
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.18.0.21:8080/ (Press CTRL+C to quit)
 * Restarting with stat
Staring Local Development
 * Debugger is active!
 * Debugger PIN: 147-068-007
172.18.0.1 - - [26/Sep/2021 20:31:15] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:15] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:16] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:21] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:22] "GET /favicon.ico HTTP/1.1" 4
172.18.0.1 - - [26/Sep/2021 20:31:25] "GET /articles_by/thejeshgn H
```

It is running. Similar project structure everything. Now we have a good project or decent project structure to do all your work. So, now if you want to extend more, you can go to models and add more models and define relationship. Let us say, for example, you want to do tags, each article, you want to add tags. So, you can define a model called tags. And then add your controllers get tags or get articles by tags and stuff like that.

So, you can keep working on it by yourself, or if you are working with a team if it is a collaboration with your team. So, maybe we can share the URL of this set up. So, you can you can fork it and try it out exam. Now, there are many, like I said, in the beginning, there are many ways to set this up.

(Refer Slide Time: 25:54)



Project Layout

Create a project directory and enter it:

```
$ mkdir flask-tutorial
$ cd flask-tutorial
```

Then follow the installation instructions to set up a Python virtual environment and install Flask for your project.

The tutorial will assume you're working from the `flask-tutorial` directory from now on. The file names at the top of each code block are relative to this directory.

A Flask application can be as simple as a single file.

hello.py

```
from flask import Flask

app = Flask(__name__)


@app.route('/')
def hello():
    return 'Hello, World!'
```

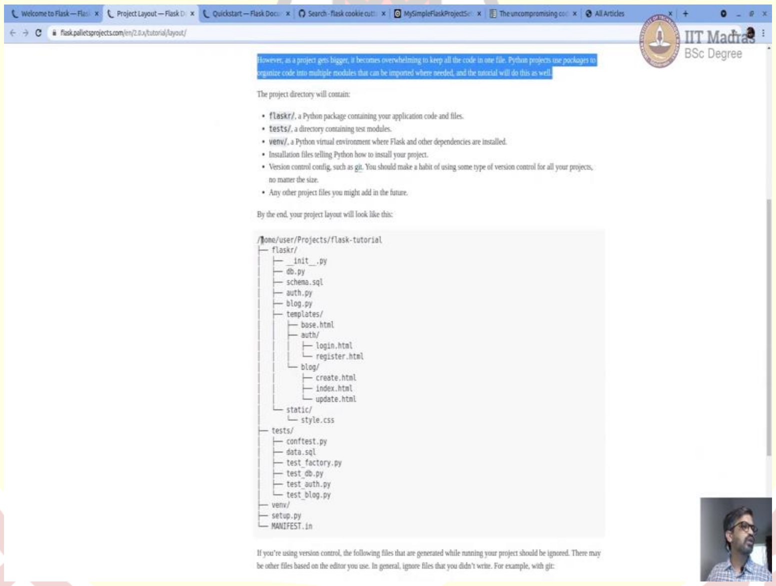
However, as a project gets bigger, it becomes overwhelming to keep all the code in one file. Python projects use packages to organize code into multiple modules that can be imported where needed, and the tutorial will do this as well.

The project directory will contain:

- `flaskr/`, a Python package containing your application code and files.
- `tests/`, a directory containing test modules.
- `venv/`, a Python virtual environment where Flask and other dependencies are installed.
- Installation files telling Python how to install your project.
- Version control config, such as git. You should make a habit of using some type of version control for all your projects, no matter the size.
- Any other project files you might add in the future.

By the end, your project layout will look like this:

```
/home/user/Projects/flask-tutorial
├── flaskr/
│   ├── __init__.py
```



However, as a project gets bigger, it becomes overwhelming to keep all the code in one file. Python projects use packages to organize code into multiple modules that can be imported where needed, and the tutorial will do this as well.

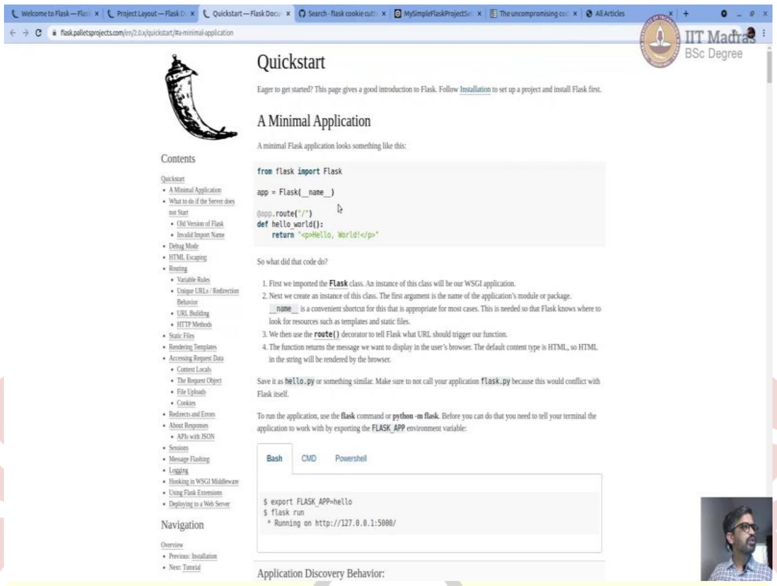
The project directory will contain:

- `flaskr/`, a Python package containing your application code and files.
- `tests/`, a directory containing test modules.
- `venv/`, a Python virtual environment where Flask and other dependencies are installed.
- Installation files telling Python how to install your project.
- Version control config, such as git. You should make a habit of using some type of version control for all your projects, no matter the size.
- Any other project files you might add in the future.

By the end, your project layout will look like this:

```
/home/user/Projects/flask-tutorial
├── flaskr/
│   ├── __init__.py
│   ├── db.py
│   ├── schema.sql
│   ├── auth.py
│   ├── blog.py
│   ├── templates/
│   │   ├── base.html
│   │   ├── auth/
│   │   │   ├── login.html
│   │   │   └── register.html
│   │   └── blog/
│   │       ├── create.html
│   │       ├── index.html
│   │       └── update.html
│   └── static/
│       └── style.css
├── tests/
│   ├── conftest.py
│   ├── data.sql
│   ├── test_factory.py
│   ├── test_db.py
│   ├── test_auth.py
│   └── test_blog.py
├── venv/
├── setup.py
└── MANIFEST.in
```

If you're using version control, the following files that are generated while running your project should be ignored. There may be other files based on the editor you use. In general, ignore files that you didn't write. For example, with git:



Contents

Quickstart
- A Minimal Application
- What to do if the Server does not Start
  - Old Version of Flask
  - Invalid Import Name
- Debug Mode
- HTML Escaping
- Routing
  - Variable Rules
  - Unique URLs / Redirection Behavior
  - URL Building
  - HTTP Methods
- Static Files
- Rendering Templates
- Accessing Request Data
  - Context Locals
  - The Request Object
  - File Uploads
  - Cookies
- Redirects and Errors
- About Responses
  - APIs with JSON
- Sessions
- Message Flashing
- Logging
- Hooking in WSGI Middleware
- Using Flask Extensions
- Deploying to a Web Server

Navigation

Overview
- Previous: Installation
- Next: Tutorial

Quickstart

Eager to get started? This page gives a good introduction to Flask. Follow Installation to set up a project and install Flask first.

A Minimal Application

A minimal Flask application looks something like this:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

So what did that code do?

1. First we imported the **Flask** class. An instance of this class will be our WSGI application.
2. Next we create an instance of this class. The first argument is the name of the application's module or package. `__name__` is a convenient shortcut for this that is appropriate for most cases. This is needed so that Flask knows where to look for resources such as templates and static files.
3. We then use the `route()` decorator to tell Flask what URL should trigger our function.
4. The function returns the message we want to display in the user's browser. The default content type is HTML, so HTML in the string will be rendered by the browser.

Save it as `hello.py` or something similar. Make sure to not call your application `flask.py` because this would conflict with Flask itself.

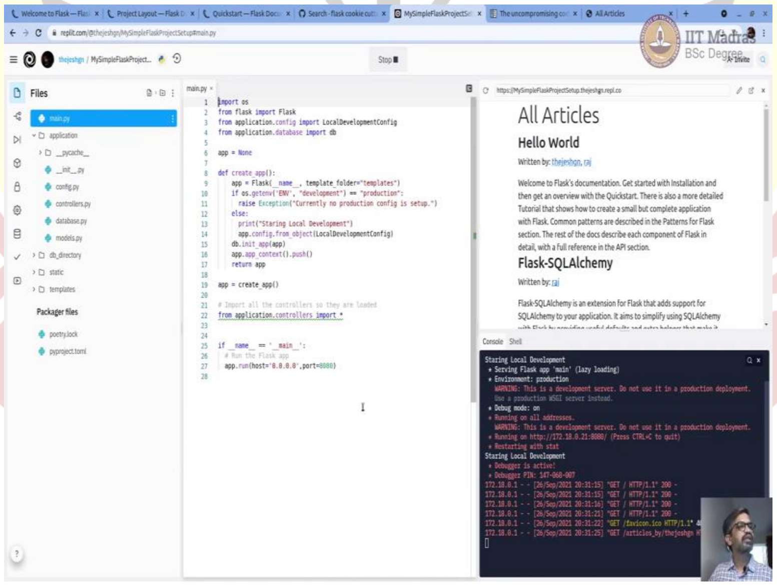
To run the application, use the **flask** command or **python -m flask**. Before you can do that you need to tell your terminal the application to work with by exporting the `FLASK_APP` environment variable:

```
$ export FLASK_APP=hello
$ flask run
 * Running on http://127.0.0.1:5000/
```

Application Discovery Behavior:

So, flask also has some ways there is some recommendation how to set it up. So, you can go to flask, project and they have a project layout, you can check it out how they set up, it is not very different, but they have their own, some suggestions which are interesting that you can pick it up and use it. And you can also see how they set up the application and run. It is not a it is not a bad way as well. So, you want to follow that should also be fine.

(Refer Slide Time: 26:28)



```
import os
from flask import Flask
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)
    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)
```

All Articles

Hello World

Written by: thejeshgn, raj

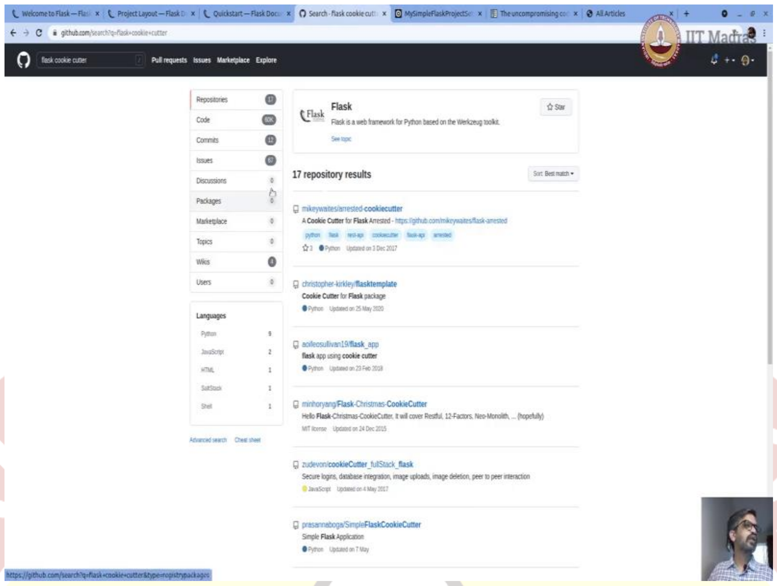
Welcome to Flask's documentation. Get started with Installation and then get an overview with the Quickstart. There is also a more detailed Tutorial that shows how to create a small but complete application with Flask. Common patterns are described in the Patterns for Flask section. The rest of the docs 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

```
Staring Local Development
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.18.0.21:8080/ (Press CTRL+C to quit)
 * Restarting with stat
Staring Local Development
 * Debugger is active!
 * Debugger PIN: 147-068-007
172.18.0.1 - - [26/Sep/2021 20:31:15] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:15] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:16] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:21] "GET / HTTP/1.1" 200 -
172.18.0.1 - - [26/Sep/2021 20:31:22] "GET /favicon.ico HTTP/1.1" 4
172.18.0.1 - - [26/Sep/2021 20:31:25] "GET /articles_by/thejeshgn H
```


github.com/search?q=flask+cookie+cutter

flask cookie cutter

Flask

Flask is a web framework for Python based on the Werkzeug toolkit.

17 repository results

mikeywaites/arrested-cookiecutter
A Cookie Cutter for Flask Arrested - https://github.com/mikeywaites/flask-arrested
python flask rest-api cookiecutter flask-api arrested
3 Python Updated on 3 Dec 2017

christopher-kirkley/flasktemplate
Cookie Cutter for Flask package
Python Updated on 25 May 2020

aoifeosullivan19/flask_app
flask app using cookie cutter
Python Updated on 23 Feb 2018

minhoryang/Flask-Christmas-CookieCutter
Hello Flask-Christmas-CookieCutter, It will cover Restful, 12-Factors, Neo-Monolith, ... (hopefully)
MIT license Updated on 24 Dec 2015

zudevon/cookieCutter_fullStack_flask
Secure logins, database integration, image uploads, image deletion, peer to peer interaction
JavaScript Updated on 4 May 2017

prasannaboga/SimpleFlaskCookieCutter
Simple Flask Application
Python Updated on 7 May

https://github.com/search?q=flask+cookie+cutter&type=registrypackages

But just make sure that if you are trying on replit I think it requires a file called main dot py to run. It is probably just a way a replit runs. So, that is why probably forking this and trying it out will be an interesting way to learn it. And you can also go to GitHub or any code repo and search for flask cookie cutter project. And you can get, get to see many projects and you can check how they have organized the code and what is good or bad about it.

So, that you can learn actually that is it from this screencast about flask Project setup hope you will get to set up and run the project. Thank you so much. Bye.