Hands-on Practice: MVC and Clean Architecture with Python (60-Minute Session)[1]

Objective: Build a simple backend API using **Flask** to implement both **MVC** and **Clean Architecture**.

Agenda (60 Minutes)

Time	Topic	Key Points
0 - 10 min	Setup Project Environment	Install Flask, initialize project
10 - 30 min	Implement MVC Pattern	Build a simple API with MVC
30 - 50 min	Implement Clean Architecture	Refactor the MVC API into Clean Architecture
50 - 60 min	Compare, Test, and Conclusion	Observe benefits and trade-offs

---

0 - 10 min: Setup Project Environment

Prerequisites:

- Install Python (3.x)
- Install required packages

```
pip install Flask Flask-SQLAlchemy
```

- Create a project folder and structure

```
mkdir backend_architecture && cd backend_architecture
mkdir models controllers routes database use case repository domain
touch app.py database/db.py
```

---

[1] Didik Maryono: Head of Engineering at Ivosights, Co-Founder Dentist Care Indonesia

10 - 30 min: Implementing MVC in Python (Flask)

1. Create Model (User Model)

📂 `models/user.py`

```
from database.db import db

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)
```

---

2. Create Controller

📂 `controllers/user_controller.py`

```
from flask import request, jsonify
from models.user import User
from database.db import db

def get_users():
    users = User.query.all()
    return jsonify([{"id": user.id, "name": user.name, "email": user.email} for user in users])

def create_user():
    data = request.get_json()
    new_user = User(name=data['name'], email=data['email'])
    db.session.add(new_user)
    db.session.commit()
    return jsonify({"message": "User created successfully"}), 201
```

---

3. Setup Routes

📂 `routes/routes.py`

```
from flask import Blueprint
from controllers import user_controller

user_routes = Blueprint('user_routes', __name__)
```

```
user_routes.route('/users', methods=['GET'])(user_controller.get_users)
user_routes.route('/users', methods=['POST'])(user_controller.create_user)
```

---

4. Initialize Database and Server

📂 `database/db.py`

```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()
```

📂 `app.py`

```
from flask import Flask
from database.db import db
from routes.routes import user_routes

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///test.db'
db.init_app(app)

with app.app_context():
    db.create_all()

app.register_blueprint(user_routes)

if __name__ == '__main__':
    app.run(debug=True)
```

Test the API

Start the server:

```
python app.py
```

Create a user:

```
curl -X POST http://127.0.0.1:5000/users -H "Content-Type: application/json" -d '{"name": "John Doe", "email": "john@example.com"}'
```

Fetch users:

```
curl -X GET http://127.0.0.1:5000/users
```

---

30 - 50 min: Refactoring into Clean Architecture

Now, we'll **refactor** our MVC implementation into **Clean Architecture** by introducing separate layers.

---

1. Create Domain (Entities)

📂 `domain/user.py`

```python
class UserEntity:
    def __init__(self, id, name, email):
        self.id = id
        self.name = name
        self.email = email
```

---

2. Define Repository Interface

📂 `repository/user_repository.py`

```python
from abc import ABC, abstractmethod
from domain.user import UserEntity

class UserRepository(ABC):
    @abstractmethod
    def get_all(self):
        pass

    @abstractmethod
    def create(self, user: UserEntity):
        pass
```

---

3. Implement Repository (Database Access)

📂 `repository/user_repository_impl.py`

```
from repository.user_repository import UserRepository
from models.user import User
from database.db import db

class UserRepositoryImpl(UserRepository):
    def get_all(self):
        users = User.query.all()
        return [UserEntity(user.id, user.name, user.email) for user in users]

    def create(self, user_entity: UserEntity):
        user = User(name=user_entity.name, email=user_entity.email)
        db.session.add(user)
        db.session.commit()
```

---

4. Implement Use Case (Business Logic)

📂 `usecase/user_usecase.py`

```
class UserUsecase:
    def __init__(self, user_repository):
        self.user_repository = user_repository

    def get_all_users(self):
        return self.user_repository.get_all()

    def create_user(self, user):
        return self.user_repository.create(user)
```

---

5. Implement Controller

📂 `controllers/user_controller.py`

```
from flask import request, jsonify
from usecase.user_usecase import UserUsecase
from repository.user_repository_impl import UserRepositoryImpl
from domain.user import UserEntity
```

```
user_repository = UserRepositoryImpl()
user_usecase = UserUsecase(user_repository)

def get_users():
    users = user_usecase.get_all_users()
    return jsonify([{"id": user.id, "name": user.name, "email": user.email} for user in users])

def create_user():
    data = request.get_json()
    new_user = UserEntity(None, data['name'], data['email'])
    user_usecase.create_user(new_user)
    return jsonify({"message": "User created successfully"}), 201
```

---

6. Update Router

📂 `routes/routes.py`

```
from flask import Blueprint
from controllers import user_controller

user_routes = Blueprint('user_routes', __name__)

user_routes.route('/users', methods=['GET'])(user_controller.get_users)
user_routes.route('/users', methods=['POST'])(user_controller.create_user)
```

---

50 - 60 min: Compare, Test, and Conclusion

Comparison of Both Approaches

Feature	MVC	Clean Architecture
Code Structure	Simple	Modular
Scalability	Limited	High
Business Logic Placement	Controllers	Use Case Layer
Testability	Hard	Easy

Final Thought:

- **MVC is great for simple projects** with less complexity.
- **Clean Architecture is ideal for large-scale applications** needing separation of concerns and maintainability.

This **60-minute hands-on session** provides a **practical comparison** of MVC vs Clean Architecture in Python.