PROJECT TITLE – 4
LAMBDA - Serverless Function

General Guidelines:

- Maintain a **Version Control** system with frequent commits to the GitHub repository for easier tracking of progress.
- Write clear documentation for each task completed, including design decisions, issues faced, and their solutions.
- Regularly update the CI/CD pipeline to ensure that the system is well-tested and deployable.
- Weekly_progress according to the assigned problem statement must be pushed to the repo, which will be considered during the evaluation.
- Your Team's weekly progress should be shown to the teacher on request.
- Name the GitHub repository with the four SRNs in order and the project title.

Problem Statement:

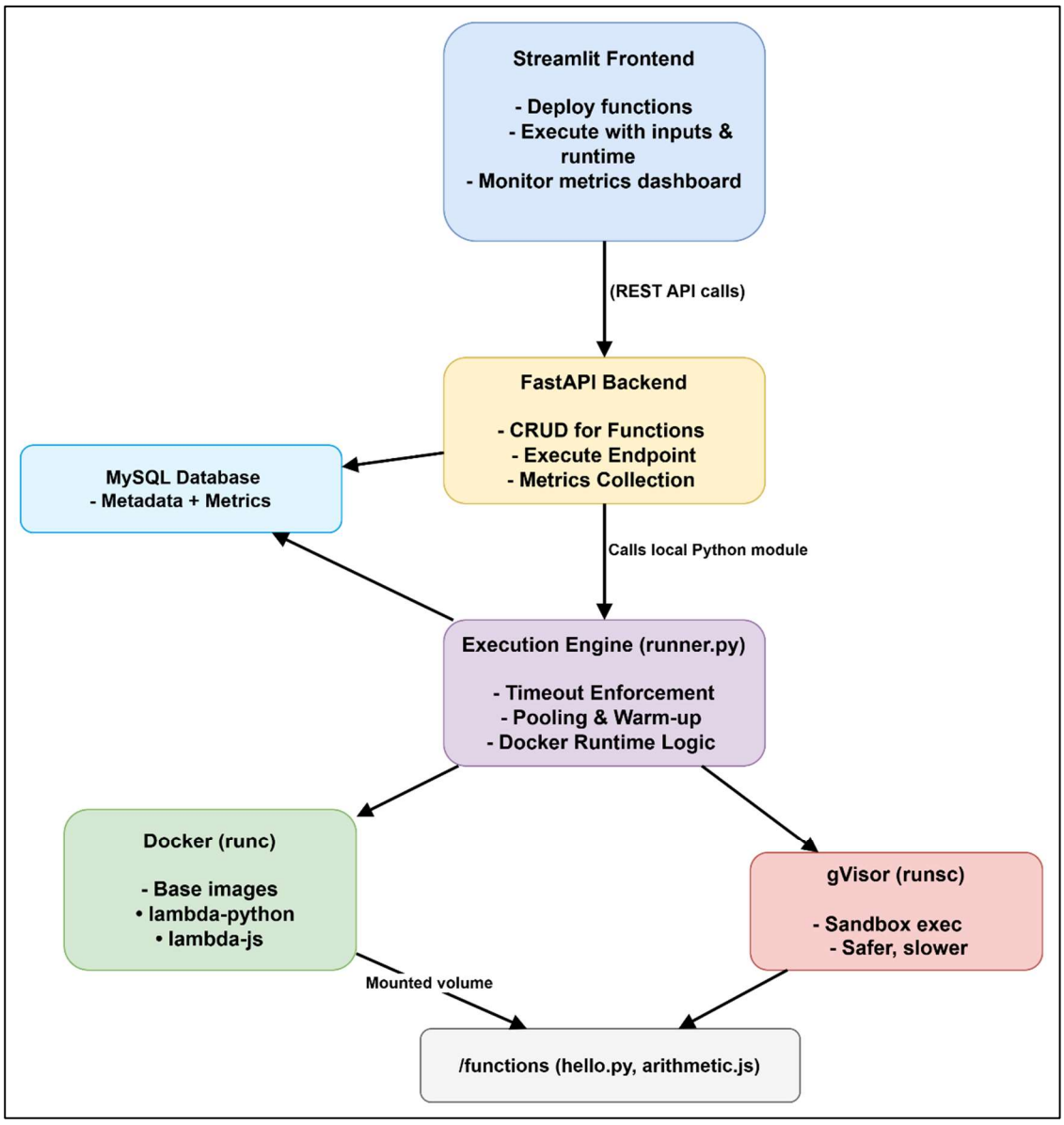
The objective of this project is to design and implement a serverless function execution platform, similar to AWS Lambda, that enables users to deploy and execute functions on-demand via HTTP requests. The system will support multiple programming languages (Python and JavaScript) and enforce execution constraints such as time limits and resource usage restrictions. To optimize execution, the platform will integrate at least two virtualization technologies, such as Firecracker MicroVMs, Nanos Unikernel, or Docker Containers, leveraging techniques like pre-warmed execution environments and request batching for improved performance. A key feature of the system is a web-based monitoring dashboard that provides real-time insights into function execution metrics, including request volume, response times, error rates, and resource utilization. The backend will handle function deployment, execution, and logging, while a frontend interface will allow users to manage and monitor their functions.

Week 1: Project Setup and Core Infrastructure (10 M)

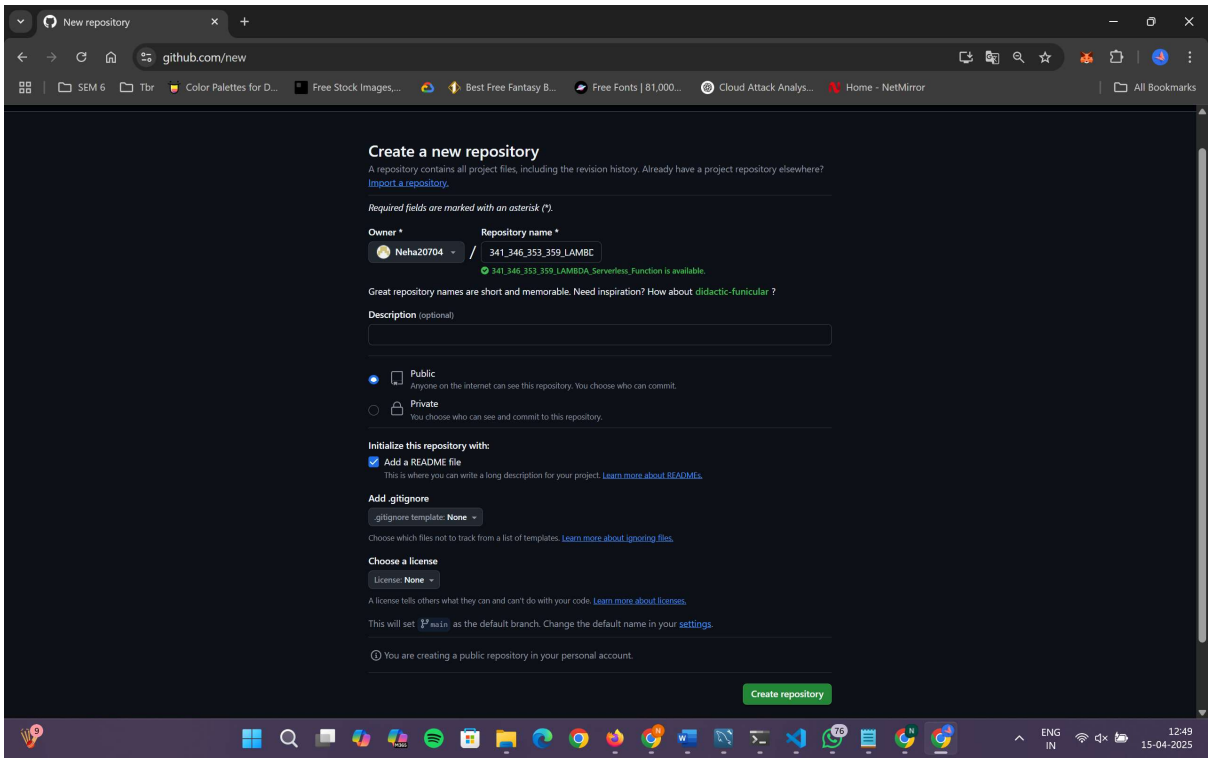
Objective: : Simple function execution via API in Docker container.

- **Task 1:** Project Planning and Environment Setup.
  - Define project architecture and create system design diagrams
  - Set up development environment (Git repository, CI/CD pipeline)
  - Choose and install required dependencies
  - Create project folder structure

Project Architecture:



Creating a git repo and initializing it with a ReadME.md



Clone and Initialize Project Structure

Run the following commands:

```
nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC
$ git clone https://github.com/Neha20704/341_346_353_359_LAMBDA_Serverless_Function
Cloning into '341_346_353_359_LAMBDA_Serverless_Function'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
```

```
nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC
$ cd 341_346_353_359_LAMBDA_Serverless_Function

nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function (main)
$
```

```
nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function (main)
$ mkdir backend frontend docker scripts docs ci-cd
```

```
nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function (main)
$ touch README.md
```

```
nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function (main)
$ ls
README.md  backend/  ci-cd/  docker/  docs/  frontend/  scripts/

nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function (main)
$
```

Initialize Git and Make First Commit

```
nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function (main)
$ git add .

nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function (main)
$ git commit -m "Initial project setup with folder structure"
[main 694c5b8] Initial project setup with folder structure
 1 file changed, 0 insertions(+), 0 deletions(-)

nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 85.89 KiB | 6.13 MiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/Neha20704/341_346_353_359_LAMBDA_Serverless_Function
   d43f6cd..694c5b8  main -> main

nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function (main)
$
```

Set Up CI/CD

```
nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function (main)
$ mkdir -p .github/workflows

nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function (main)
$ touch .github/workflows/main.yml

nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function (main)
$
```

```
D:\neha\SEM 6\CC\341_346_353_359_LAMBDA_Serverless_Function>tree /f
Folder PATH listing for volume Data
Volume serial number is E8AE-784F
D:.
    Creating a git repo and initializing it with a ReadME.docx
    README.md

├───.github
    └───workflows
            main.yml

├───backend
├───ci-cd
├───docker
├───docs
├───frontend
└───scripts

D:\neha\SEM 6\CC\341_346_353_359_LAMBDA_Serverless_Function>
```

Document in /docs/setup-log.md

```
nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/docs (main)
$ touch setup-log.md

nehag@Neha-Girish MINGW64 /d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/docs (main)
$
```

Backend Folder Structure

```
D:\neha\SEM 6\CC\341_346_353_359_LAMBDA_Serverless_Function\backend\app>tree /f
Folder PATH listing for volume Data
Volume serial number is E8AE-784F
D:.
    crud.py
    database.py
    main.py
    models.py
    schemas.py
    __init__.py

No subfolders exist


D:\neha\SEM 6\CC\341_346_353_359_LAMBDA_Serverless_Function\backend\app>
```

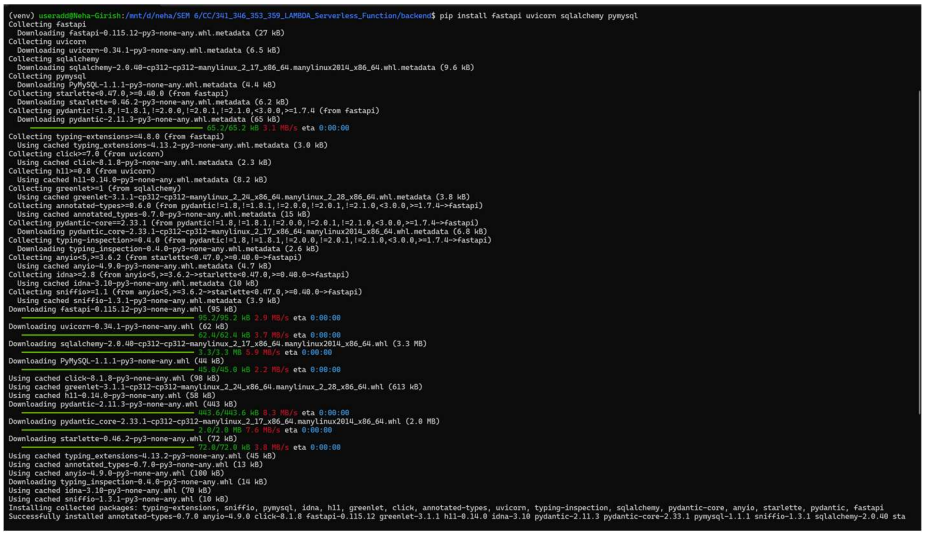
- **Task 2:** Backend API Foundation
    - Implement basic API server (Express/FastAPI)
    - Create database schema for function storage
    - Implement function metadata storage (name, route, language, timeout
    - settings)
    - Create basic CRUD endpoints for function management

Set Up FastAPI Environment

Create a Python environment:

```
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/backend$ python3 -m venv venv
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/backend$ source venv/bin/activate
(venv) useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/backend$
```

Install dependencies:



Database schema for function storage:

```
mysql> CREATE DATABASE lambda_functions;
Query OK, 1 row affected (0.03 sec)

mysql>
```

app/database.py

```python
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

# Replace these with your actual MySQL credentials
DB_USER = "root"
DB_PASSWORD = "Neha@2004"
DB_HOST = "localhost"
DB_NAME = "lambda_functions"

SQLALCHEMY_DATABASE_URL =
f"mysql+pymysql://{DB_USER}:{DB_PASSWORD}@{DB_HOST}/{DB_NAME}"

engine = create_engine(SQLALCHEMY_DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()
```

app/models.py

```python
from sqlalchemy import Column, Integer, String
from .database import Base

class Function(Base):
    __tablename__ = "functions"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), index=True)
    language = Column(String(50))
    route = Column(String(100), unique=True)
    timeout = Column(Integer)
```

app/schemas.py

```python
from pydantic import BaseModel

class FunctionCreate(BaseModel):
    name: str
    language: str
    route: str
    timeout: int

class FunctionOut(FunctionCreate):
    id: int

    class Config:
        orm_mode = True
```

app/crud.py

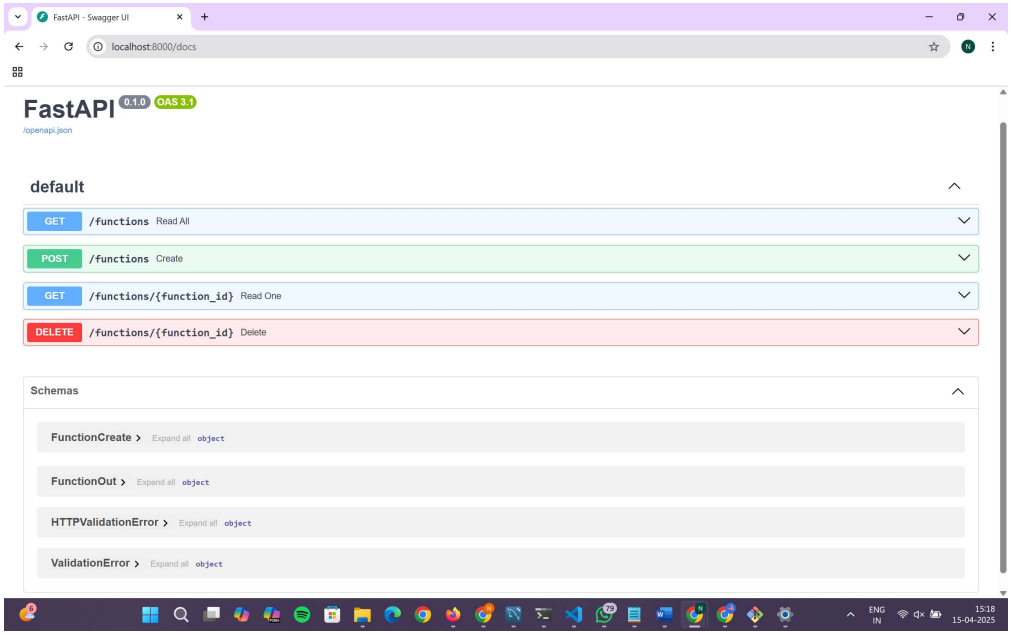
```python
from sqlalchemy.orm import Session
from . import models, schemas

def get_functions(db: Session):
    return db.query(models.Function).all()

def get_function(db: Session, function_id: int):
    return db.query(models.Function).filter(models.Function.id ==
function_id).first()

def create_function(db: Session, function: schemas.FunctionCreate):
    db_func = models.Function(**function.dict())
    db.add(db_func)
    db.commit()
    db.refresh(db_func)
    return db_func

def delete_function(db: Session, function_id: int):
    func = db.query(models.Function).filter(models.Function.id ==
function_id).first()
    if func:
        db.delete(func)
        db.commit()
        return True
    return False
```

app/main.py

```python
from fastapi import FastAPI, Depends, HTTPException
from sqlalchemy.orm import Session
from . import models, schemas, crud, database

models.Base.metadata.create_all(bind=database.engine)

app = FastAPI()

def get_db():
    db = database.SessionLocal()
    try:
        yield db
    finally:
        db.close()

@app.post("/functions", response_model=schemas.FunctionOut)
def create(function: schemas.FunctionCreate, db: Session = Depends(get_db)):
    return crud.create_function(db, function)

@app.get("/functions", response_model=list[schemas.FunctionOut])
def read_all(db: Session = Depends(get_db)):
    return crud.get_functions(db)

@app.get("/functions/{function_id}", response_model=schemas.FunctionOut)
def read_one(function_id: int, db: Session = Depends(get_db)):
    func = crud.get_function(db, function_id)
    if not func:
        raise HTTPException(status_code=404, detail="Function not found")
    return func

@app.delete("/functions/{function_id}")
def delete(function_id: int, db: Session = Depends(get_db)):
    success = crud.delete_function(db, function_id)
    if not success:
        raise HTTPException(status_code=404, detail="Function not found")
    return {"deleted": True}
```

Backend server running:

```
cd backend
uvicorn app.main:app –reload
```



- **Task 3:** Your First Virtualization Technology.
    - Set up Docker as the first virtualization technology
    - Create base container images for Python and JavaScript functions
    - Implement function packaging mechanism
    - Build basic execution engine that can run a function inside Docker
    - Implement timeout enforcement make sure this is clearly thought through

Deliverable: A simple working prototype using docker.

docker/base_images/python/Dockerfile

```
FROM python:3.12-slim
WORKDIR /app
COPY handler.py .
CMD ["python", "handler.py"]
```

docker/base_images/python/handler.py

```
import importlib.util
import os

func_file = os.environ.get("FUNCTION_FILE", "hello.py")
```

```
spec = importlib.util.spec_from_file_location("user_func",
f"/functions/{func_file}")
module = importlib.util.module_from_spec(spec)
spec.loader.exec_module(module)

if hasattr(module, 'main'):
    print(module.main())
else:
    print("No main() found")
```

docker/base_images/javascript/Dockerfile

```
FROM node:18-alpine
WORKDIR /app
COPY handler.js .
CMD ["node", "handler.js"]
```

docker/base_images/javascript/handler.js

```
const { exec } = require('child_process');

const file = process.env.FUNCTION_FILE || 'hello.js';
exec(`node /functions/${file}`, (err, stdout, stderr) => {
    if (err) {
        console.error('Execution error:', err);
        return;
    }
    console.log(stdout);
});
```

docker/functions/python/hello.py

```
import sys

def main(name="World"):
    return f"Hello, {name}!"

if __name__ == "__main__":
    # Get the name from CLI args
    arg = sys.argv[1] if len(sys.argv) > 1 else "World"
    print(main(arg))
```

docker/functions/python/arithmetic.py

```python
import sys

def add(a, b):
    return a + b

if __name__ == "__main__":
    if len(sys.argv) < 3:
        print("Usage: python arithmetic.py <num1> <num2>")
    else:
        num1 = float(sys.argv[1])
        num2 = float(sys.argv[2])
        print(f"Sum: {add(num1, num2)}")
```

docker/functions/javascript/hello.js

```javascript
const args = process.argv.slice(2);
const inputName = args[0] || "World";
console.log(`Hello, ${inputName}!`);
```

docker/functions/javascript/arithmetic.js

```javascript
const a = parseFloat(process.argv[2]);
const b = parseFloat(process.argv[3]);

if (isNaN(a) || isNaN(b)) {
    console.log("Usage: node arithmetic.js <num1> <num2>");
} else {
    console.log(`Sum: ${a + b}`);
}
```

Build Base Images

Commands:
cd docker/base_images/python

docker build -t lambda-python .

cd ../javascript

docker build -t lambda-javascript .

```
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$ cd docker/base_images/python
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/docker/base_images/python$ docker build -t lambda-python .
[+] Building 4.5s (8/8) FINISHED                                                                                                docker:default
 => [internal] load build definition from Dockerfile                                                                                      0.1s
 => => transferring dockerfile: 123B                                                                                                      0.0s
 => [internal] load metadata for docker.io/library/python:3.12-slim                                                                       1.0s
 => [internal] load .dockerignore                                                                                                         0.1s
 => => transferring context: 2B                                                                                                           0.0s
 => [1/3] FROM docker.io/library/python:3.12-slim@sha256:85824326bc4ae27a1abb5bc0dd9e08847aa5fe73d8afb593b1b45b7cb4180f57                 2.9s
 => => resolve docker.io/library/python:3.12-slim@sha256:85824326bc4ae27a1abb5bc0dd9e08847aa5fe73d8afb593b1b45b7cb4180f57                 0.0s
 => => sha256:6cca951a45d16183402c0ea0d763285dff4024af76a75119fb405268bc35c359 3.51MB / 3.51MB                                           17.9s
 => => sha256:b3b4f30c6749b72e9b25139714a21ee8c3c5689b213a023c7b42be2634cfc320 13.66MB / 13.66MB                                          1.8s
 => => sha256:e9ddbe7a005f6950194f9ec89ccfff2c867294919O1f767d8b03f8062de3daf5 249B / 249B                                                18.0s
 => => sha256:85824326bc4ae27a1abb5bc0dd9e08847aa5fe73d8afb593b1b45b7cb4180f57 9.13kB / 9.13kB                                            0.0s
 => => sha256:a08295c4819f91ab6bbf02a62f2d70b87a3b76ac79fecf5260228f41dc4aa1c4 1.75kB / 1.75kB                                            0.0s
 => => sha256:3d1d8f39b99d1a90422205ebfa5044475db03aabe1ae4f937fea5d4ed1076ad8d 5.51kB / 5.51kB                                            0.0s
 => => extracting sha256:6cca951a45d16183402c0ea0d763285dff4024af76a75119fb405268bc35c359                                                 0.0s
 => => extracting sha256:b3b4f30c6749b72e9b25139714a21ee8c3c5689b213a023c7b42be2634cfc320                                                 0.8s
 => => extracting sha256:e9ddbe7a005f6950194f9ec89ccfff2c867294919O1f767d8b03f8062de3daf5                                                 0.0s
 => [internal] load build context                                                                                                         0.1s
 => => transferring context: 396B                                                                                                         0.0s
 => [2/3] WORKDIR /app                                                                                                                    0.2s
 => [3/3] COPY handler.py .                                                                                                               0.1s
 => exporting to image                                                                                                                    0.1s
 => => exporting layers                                                                                                                   0.1s
 => => writing image sha256:e7a3a78b9a5ef617cffe3a269be775b434335117fc2ddc639da4932ee03fe193                                              0.0s
 => => naming to docker.io/library/lambda-python                                                                                          0.0s
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/docker/base_images/python$
```

```
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/docker/base_images/javascript$ docker build -t lambda-javascript .
[+] Building -7.6s (9/9) FINISHED                                                                                                docker:default
 => [internal] load build definition from Dockerfile                                                                                      0.1s
 => => transferring dockerfile: 119B                                                                                                      0.0s
 => [internal] load metadata for docker.io/library/node:18-alpine                                                                         0.0s
 => [auth] library/node:pull token for registry-1.docker.io                                                                               0.0s
 => [internal] load .dockerignore                                                                                                         0.0s
 => => transferring context: 2B                                                                                                           0.0s
 => [1/3] FROM docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e                   5.8s
 => => resolve docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e                   0.0s
 => => sha256:1e5a4c89cee5c0826c540ab06d4b6b491c96eda01837f430bd47f0d26702d6e3 1.26MB / 1.26MB                                             0.6s
 => => sha256:25ff2da83641908f65c3a74d80409d6b1b62ccfaab220b9ea70b80df5a2e0549 446B / 446B                                                 0.6s
 => => sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e 7.67kB / 7.67kB                                            16.4s
 => => sha256:929b04d7c782f04f615cf785488fed452b6569f87c73ff666ad553a7554f0006 1.72kB / 1.72kB                                            16.7s
 => => sha256:ee77c6cd7c1886ecc802ad6cedef3a8ec1ea27d1fb96162bf03dd3710839b8da 6.18kB / 6.18kB                                            0.0s
 => => sha256:dd71dde834b5c203d162902e6b8994cb2309ae049a0eabc4efea161b2b5a3d0e 40.01MB / 40.01MB                                          3.4s
 => => extracting sha256:dd71dde834b5c203d162902e6b8994cb2309ae049a0eabc4efea161b2b5a3d0e                                                 1.9s
 => => extracting sha256:1e5a4c89cee5c0826c540ab06d4b6b491c96eda01837f430bd47f0d26702d6e3                                                 0.1s
 => => extracting sha256:25ff2da83641908f65c3a74d80409d6b1b62ccfaab220b9ea70b80df5a2e0549                                                 0.0s
 => [internal] load build context                                                                                                         0.1s
 => => transferring context: 320B                                                                                                         0.0s
 => [2/3] WORKDIR /app                                                                                                                    0.3s
 => [3/3] COPY handler.js .                                                                                                               0.1s
 => exporting to image                                                                                                                    0.1s
 => => exporting layers                                                                                                                   0.1s
 => => writing image sha256:24062053d0bcfb7b7847666e45648e6a50a03392a7845dc426528c39e3a4e6e9                                              0.0s
 => => naming to docker.io/library/lambda-javascript                                                                                      0.0s
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/docker/base_images/javascript$
```

Run Python & JavaScript Functions in Containers

Commands:

docker run --rm -v "$(pwd)/docker/functions/python:/app" lambda-python python hello.py ABC

docker run --rm -v "$(pwd)/docker/functions/python:/app" lambda-python python arithmetic.py 5 7

docker run --rm -v "$(pwd)/docker/functions/javascript:/app" lambda-javascript node hello.js

docker run --rm -v "$(pwd)/docker/functions/javascript:/app" lambda-javascript node arithmetic.js 5 10

hello.py:

```
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$ docker run --rm -v "$(pwd)/docker/functions/python:/app" lambda-python
python hello.py ABC
Hello, ABC!
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$
```

arithmetic.py:

```
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$ docker run --rm -v "$(pwd)/docker/functions/python:/app" lambda-python
python arithmetic.py
Usage: python arithmetic.py <num1> <num2>
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$ docker run --rm -v "$(pwd)/docker/functions/python:/app" lambda-python
python arithmetic.py 5 7
Sum: 12.0
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$
```

hello.js:

```
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$ docker run --rm -v "$(pwd)/docker/functions/javascript:/app" lambda-jav
ascript node hello.js
Hello, World!
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$
```

arithmetic.js:

```
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$ docker run --rm -v "$(pwd)/docker/functions/javascript:/app" lambda-jav
ascript node arithmetic.js 5 10
Sum: 15
```

docker directory structure:

```
D:\neha\SEM 6\CC\341_346_353_359_LAMBDA_Serverless_Function\docker>tree /f
Folder PATH listing for volume Data
Volume serial number is E8AE-784F
D:.
├───base_images
│   ├───javascript
│   │       Dockerfile
│   │       handler.js
│   │
│   └───python
│           Dockerfile
│           handler.py
│
└───functions
    ├───javascript
    │       arithmetic.js
    │       hello.js
    │
    └───python
            arithmetic.py
            hello.py
```

Building basic execution. Mechanism

executor/executor.py

```python
# executor/executor.py
import subprocess
import os

def run_function(language, filename, timeout=5):
```

```
    """
    Execute a function inside a Docker container with timeout enforcement.

    Args:
        language (str): 'python' or 'javascript'
        filename (str): Function file name (e.g., hello.py)
        timeout (int): Timeout in seconds

    Returns:
        stdout (str), stderr (str)
    """
    container = f"lambda-{language}"
    func_path = os.path.abspath(os.path.join(os.path.dirname(__file__),
"../docker/functions", language))
   # func_path = os.path.abspath(f"./docker/functions/{language}")

    cmd = [
        "docker", "run", "--rm",
        "-v", f"{func_path}:/functions",              # Mount host function
dir to /functions in container
        "-e", f"FUNCTION_FILE={filename}",            # Pass the file to be
run
        container                                     # This image runs
handler.py/js, which picks up FUNCTION_FILE
    ]

    try:
        print("Running:", " ".join(cmd))  # Debug
        result = subprocess.run(cmd, capture_output=True, text=True,
timeout=timeout)
        return result.stdout.strip(), result.stderr.strip()
    except subprocess.TimeoutExpired:
        return "", f"ERROR: Execution timed out after {timeout} seconds"
```

test file: run_tests.py

```
# executor/run_tests.py
from executor import run_function

# Test Python hello
print("== Python Hello ==")
out, err = run_function("python", "hello.py", args=["ChatGPT"])
print("Output:", out)
print("Error:", err)
```

```
# Test JS hello
print("== JavaScript Hello ==")
out, err = run_function("javascript", "hello.js", args=["ChatGPT"])
print("Output:", out)
print("Error:", err)

# Test timeout (Python infinite loop)
print("== Timeout Test ==")
out, err = run_function("python", "infinite.py", timeout=3)
print("Output:", out)
print("Error:", err)
```

Testing timeout mechanism:

docker/functions/python/infinite.py:

```
# infinite.py
import time

while True:
    time.sleep(1)
```

showing timing out:

```
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/executor$ python3 run_tests.py
== Python Hello ==
Running: docker run --rm -v /mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/docker/functions/python:/functions -e FUNCTION_FILE=hello.py lam
bda-python
Output: Hello, World!
Error:
== JS Hello ==
Running: docker run --rm -v /mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/docker/functions/javascript:/functions -e FUNCTION_FILE=hello.js
 lambda-javascript
Output: Hello, World!
Error:
== Timeout Test ==
Running: docker run --rm -v /mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/docker/functions/python:/functions -e FUNCTION_FILE=infinite.py
lambda-python
Output:
Error: ERROR: Execution timed out after 3 seconds
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/executor$
```

Week 2: Enhanced Execution and Second Virtualization Technology (15 M)

Objective: The goal for Week 2 is to enhance the function execution capabilities of the platform by improving the execution engine with more advanced routing, warm-up mechanisms, and error handling. Additionally, this week will introduce a second virtualization technology alongside Docker, either **Firecracker MicroVMs**, **Nanos Unikernel**, or **gVisor**. This is crucial for comparing performance across different virtualization approaches and choosing the most effective one for handling serverless function executions at scale.

The second virtualization technology's integration will allow the system to support a broader range of deployment environments, which is necessary for scalability and performance optimization. We will also collect and aggregate execution metrics to monitor the function's performance in terms of response times, error rates, and resource utilization.

- **Task 1:** Execution Engine Improvements:
    - Implement request routing to appropriate function containers
    - Add request/response handling and error management
    - Implement function warm-up mechanism , i.e dummy caching and function
    - Create a container pool for improved performance.

Updated executor.py with Dynamic routing to the correct container (Python/JS), Timeout enforcement , Basic container pooling support, Error handling

```
#executor/executor.py
import subprocess
import os
import time
import threading
from queue import Queue

# Container pool setup
container_pool = Queue()

# Max pool size (adjust as needed)
POOL_SIZE = 5

# Populate the pool with containers
def init_container_pool():
    for _ in range(POOL_SIZE):
        container_pool.put(None)  # Placeholder for container

# Function to run code inside containers
def run_function(language, filename, timeout=5):
    container = f"lambda-{language}"
    func_path = os.path.abspath(os.path.join(os.path.dirname(__file__),
"../docker/functions", language))
    # Get a container from the pool
    if container_pool.empty():
        print("No containers available, creating a new one...")
        container_id = None  # Or use subprocess to run a new container if
necessary
    else:
        container_id = container_pool.get()
```

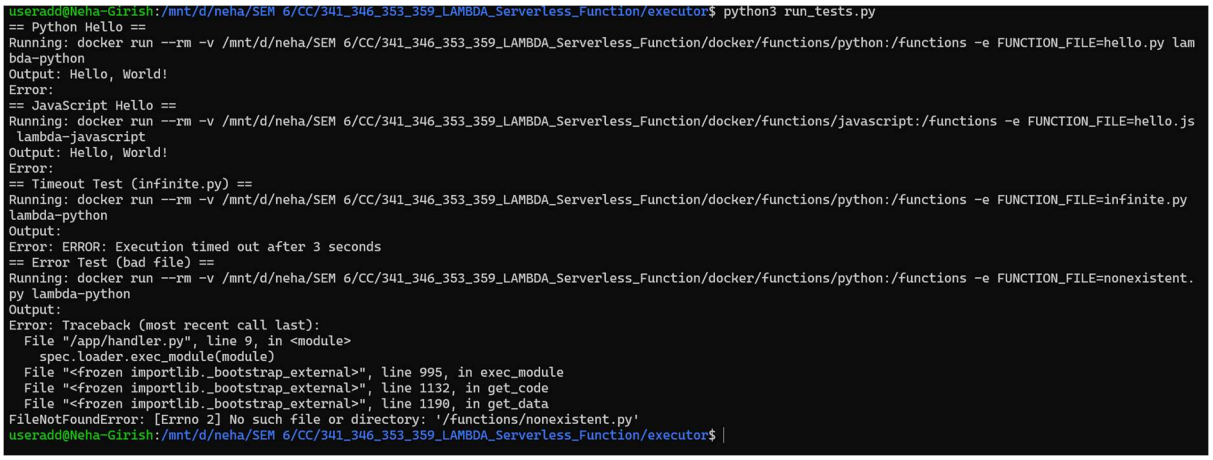
```
    cmd = [
        "docker", "run", "--rm",
        "-v", f"{func_path}:/functions",
        "-e", f"FUNCTION_FILE={filename}",
        container
    ]

    try:

        print("Running:", " ".join(cmd))
        result = subprocess.run(cmd, capture_output=True, text=True,
timeout=timeout)
        return result.stdout.strip(), result.stderr.strip()
    except subprocess.TimeoutExpired:
        return "", f"ERROR: Execution timed out after {timeout} seconds"
    except Exception as e:
        return "", f"ERROR: {str(e)}"
    finally:
        # Put container back into pool (or cleanup)
        if container_id:
            container_pool.put(container_id)

# Initialize container pool
init_container_pool()
```

updated run_tests.py:

```
# executor/run_tests.py
from executor import run_function

print("== Python Hello ==")
out, err = run_function("python", "hello.py", timeout=5)
print("Output:", out)
print("Error:", err)

print("== JavaScript Hello ==")
out, err = run_function("javascript", "hello.js", timeout=5)
print("Output:", out)
print("Error:", err)

print("== Timeout Test (infinite.py) ==")
out, err = run_function("python", "infinite.py", timeout=3)
print("Output:", out)
print("Error:", err)

print("== Error Test (bad file) ==")
```

```
out, err = run_function("python", "nonexistent.py", timeout=3)
print("Output:", out)
print("Error:", err)
```

Output Screenshot:



__Update executor.py ->runner.py to avoid wsl clashes__**

Updated code of runner.py:

```
import subprocess
import os
from queue import Queue

# Container pool setup
container_pool = Queue()
POOL_SIZE = 5

# List of functions to pre-warm
WARMED_FUNCTIONS = {
    "python": ["hello.py", "arithmetic.py", "infinite.py"],
    "javascript": ["hello.js", "arithmetic.js"]
}

def init_container_pool():
    for _ in range(POOL_SIZE):
        container_pool.put(None)  # Placeholder if you implement persistent containers

def run_function(language, filename, args=None, timeout=5):
    container = f"lambda-{language}"
    func_path = os.path.abspath(os.path.join(os.path.dirname(__file__), "../docker/functions", language))
```

```
    # Build the base command
    cmd = [
        "docker", "run", "--rm",
        "-v", f"{func_path}:/functions",
        "-e", f"FUNCTION_FILE={filename}",
    ]

    if args:
        import json
        cmd += ["-e", f"ARGS={json.dumps(args)}"]

    # Add image name last
    cmd.append(container)

    try:
        print("Running:", " ".join(cmd))
        result = subprocess.run(cmd, capture_output=True, text=True,
timeout=timeout)
        return result.stdout.strip(), result.stderr.strip()
    except subprocess.TimeoutExpired:
        return "", f"ERROR: Execution timed out after {timeout} seconds"
    except Exception as e:
        return "", f"ERROR: {str(e)}"
    finally:
        container_pool.put(None)

def warm_up_functions():
    print("Warming up frequently used functions...")
    for lang in WARMED_FUNCTIONS:
        for fname in WARMED_FUNCTIONS[lang]:
            try:
                print(f" → {lang}/{fname}")
                run_function(lang, fname, timeout=3)
            except Exception as e:
                print(f"Warm-up failed for {lang}/{fname}: {str(e)}")

# Init on import
init_container_pool()
warm_up_functions()
```

Updated main.py:

```
from fastapi import FastAPI, Depends, HTTPException, Request

from sqlalchemy.orm import Session
from . import models, schemas, crud, database
```

```
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
"../../executor")))
import runner  # ☑ this imports runner.py from the executor/ folder

models.Base.metadata.create_all(bind=database.engine)

app = FastAPI()

def get_db():
    db = database.SessionLocal()
    try:
        yield db
    finally:
        db.close()

@app.post("/functions", response_model=schemas.FunctionOut)
def create(function: schemas.FunctionCreate, db: Session = Depends(get_db)):
    try:
        return crud.create_function(db, function)
    except Exception as e:
        print(f"Error occurred: {str(e)}")  # Log the error for debugging
        raise HTTPException(status_code=500, detail="Internal Server Error")


@app.get("/functions", response_model=list[schemas.FunctionOut])
def read_all(db: Session = Depends(get_db)):
    return crud.get_functions(db)

@app.get("/functions/{function_id}", response_model=schemas.FunctionOut)
def read_one(function_id: int, db: Session = Depends(get_db)):
    func = crud.get_function(db, function_id)
    if not func:
        raise HTTPException(status_code=404, detail="Function not found")
    return func

@app.delete("/functions/{function_id}")
def delete(function_id: int, db: Session = Depends(get_db)):
    success = crud.delete_function(db, function_id)
    if not success:
        raise HTTPException(status_code=404, detail="Function not found")
    return {"deleted": True}

@app.post("/functions/execute")
async def execute_function(request: schemas.ExecuteFunctionRequest, db:
Session = Depends(get_db)):
    function_id = request.id
```

```
    args = request.args or []  # Default to empty list if None

    # Get metadata from DB using function_id
    func = db.query(models.Function).filter(models.Function.id ==
function_id).first()
    if not func:
        raise HTTPException(status_code=404, detail="Function not found")

    # Run the function with args
    output, error = runner.run_function(func.language, func.route, args=args,
timeout=func.timeout)
    return {"output": output, "error": error}
```

update handler.py

```
import importlib.util

import os
import json

func_file = os.environ.get("FUNCTION_FILE", "hello.py")
func_path = f"/functions/{func_file}"
raw_args = os.environ.get("ARGS", "[]")

try:
    args = json.loads(raw_args)
except:
    args = []

spec = importlib.util.spec_from_file_location("user_func", func_path)
module = importlib.util.module_from_spec(spec)
spec.loader.exec_module(module)

if hasattr(module, 'main'):
    print(module.main(*args))
else:
    print("No main() found")
```

update handler.js

```
const { exec } = require('child_process');

const file = process.env.FUNCTION_FILE || 'hello.js';
let args = [];

try {
```

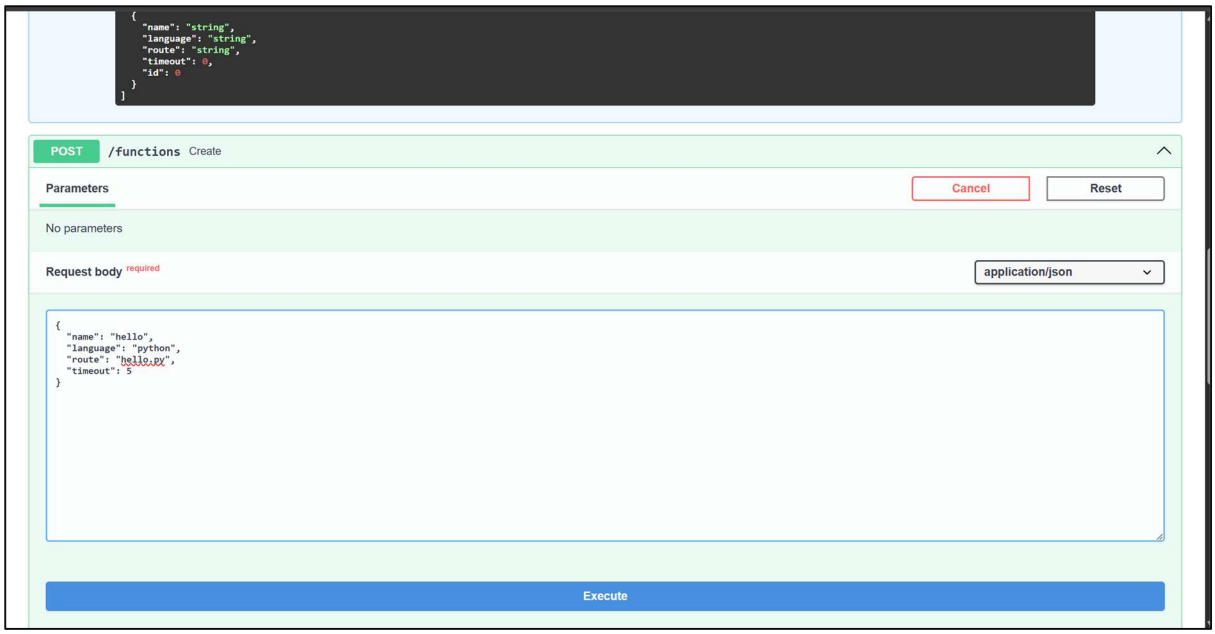
```
  args = JSON.parse(process.env.ARGS || '[]');
} catch (e) {
  console.error('Failed to parse ARGS:', e);
}

const command = `node /functions/${file} ${args.join(' ')}`;

exec(command, (err, stdout, stderr) => {
  if (err) {
    console.error('Execution error:', err);
    console.error(stderr);
    return;
  }
  console.log(stdout);
});
```

Rebuild docker containers again after making changes

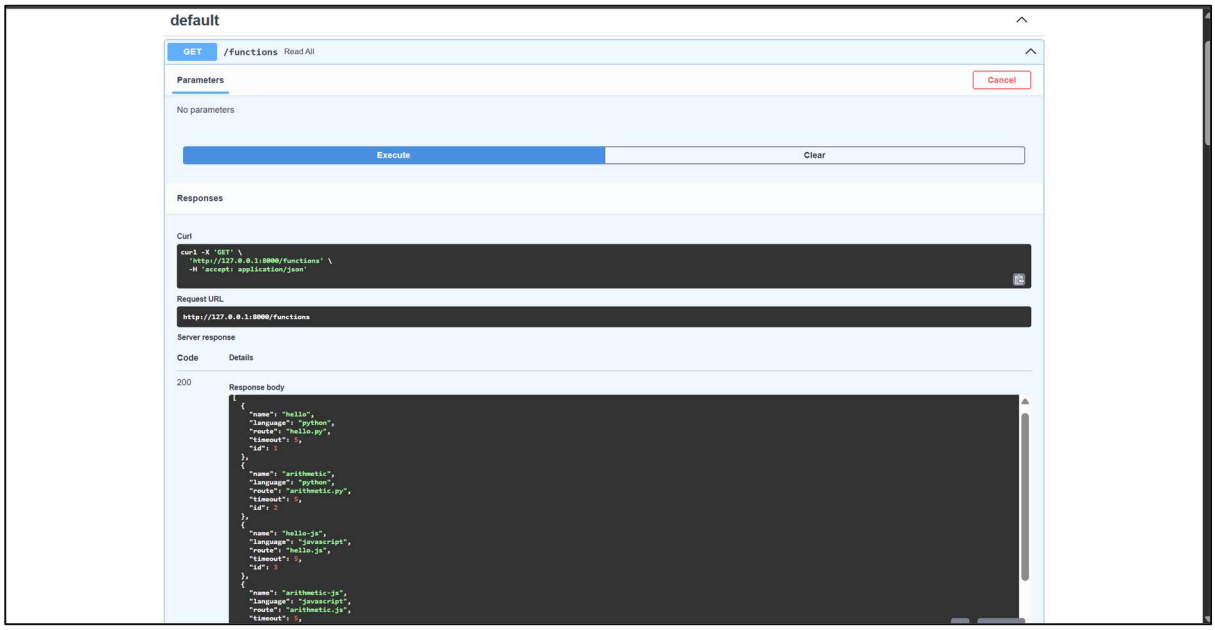
run api server:

cd backend
uvicorn app.main:app –reload

Posting Functions using POST:



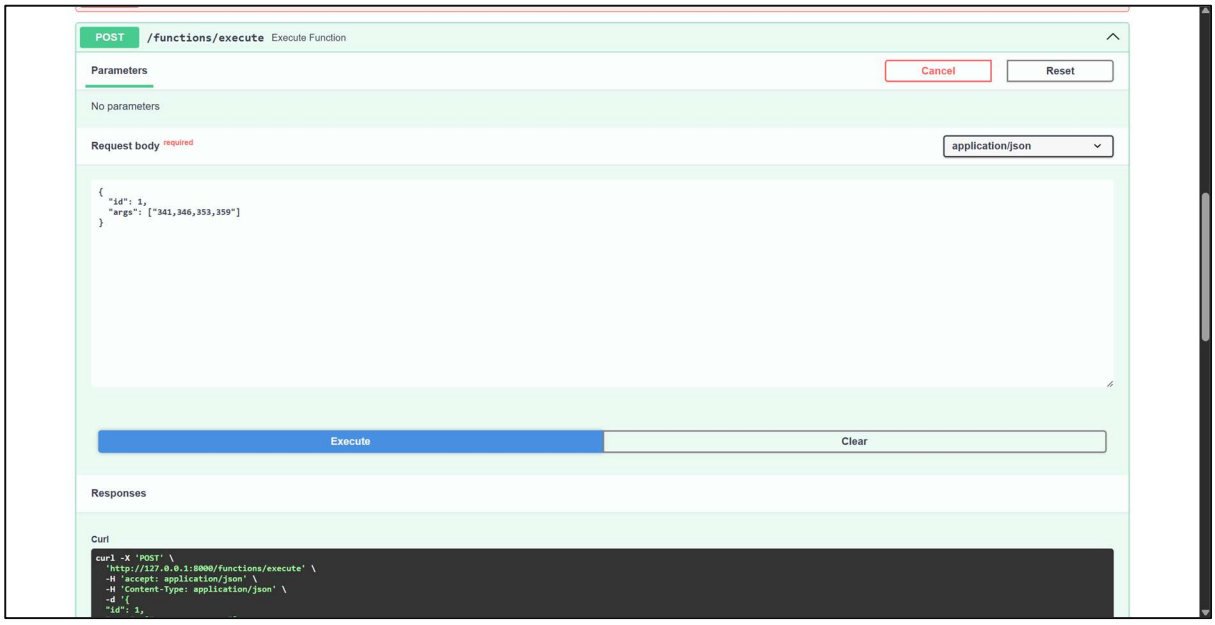
Using GET function to view function list:

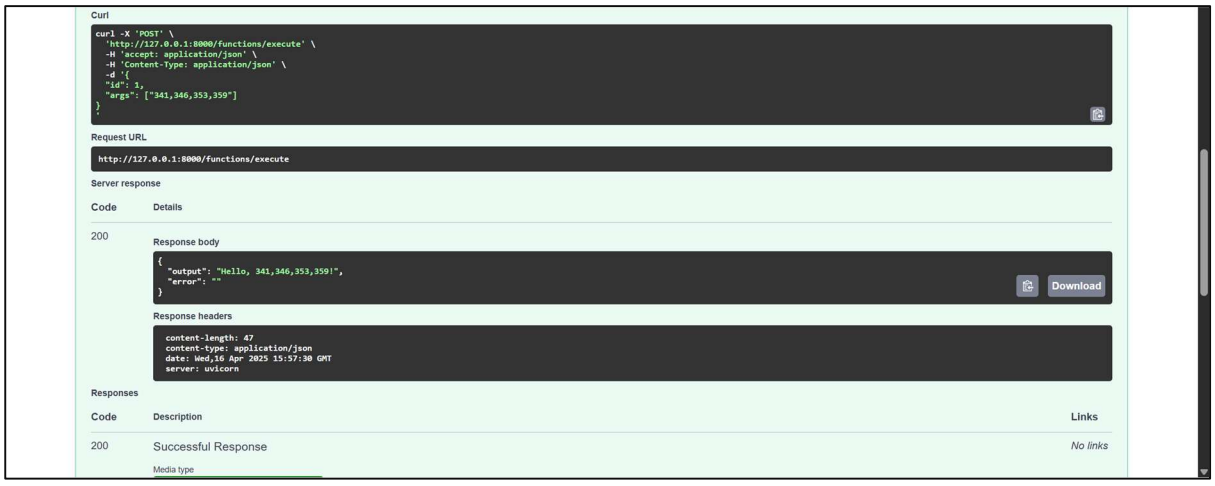


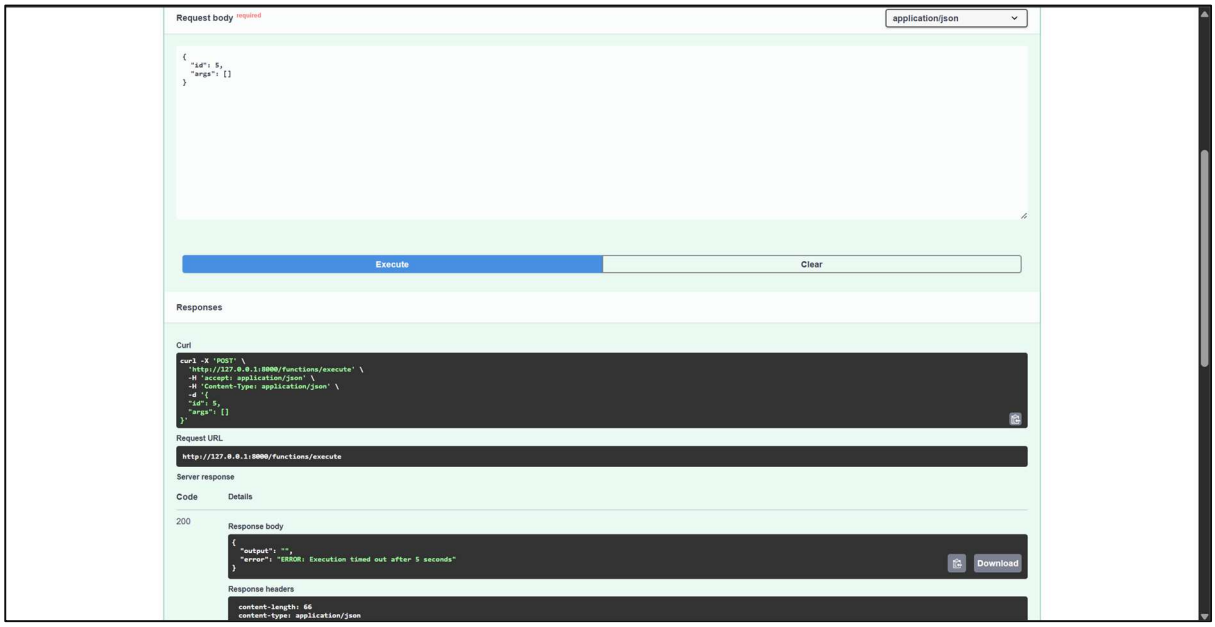
SQL database is also updated automatically:

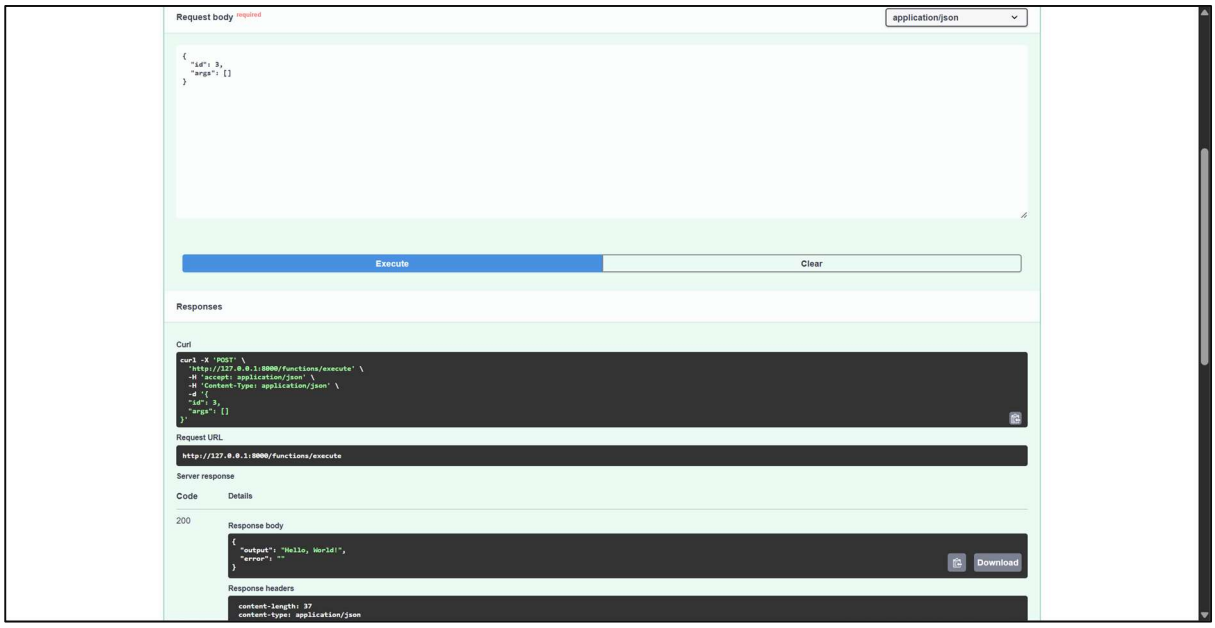
```
mysql> select * from functions;
+----+---------------+------------+---------------+---------+
| id | name          | language   | route         | timeout |
+----+---------------+------------+---------------+---------+
|  1 | hello         | python     | hello.py      |       5 |
|  2 | arithmetic    | python     | arithmetic.py |       5 |
|  3 | hello-js      | javascript | hello.js      |       5 |
|  4 | arithmetic-js | javascript | arithmetic.js |       5 |
|  5 | infinite      | python     | infinite.py   |       5 |
+----+---------------+------------+---------------+---------+
5 rows in set (0.00 sec)

mysql>
```

Executing with custom inputs using function/execute









- **Task 2:** Second Virtualization Technology.
    - Set up second virtualization technology (Firecracker MicroVMs or Nanos Unikernel),ps: if you find them hard to set up try using gvisor.
    - Create packaging mechanism for the second technology
    - Implement execution engine support for the second technology
    - Compare performance between the two virtualization approaches

Setting up gvisor, i.e, runtime = runsc

```
useradd@Neha-Girish:~$ sudo usermod -aG docker $USER
useradd@Neha-Girish:~$ newgrp docker
useradd@Neha-Girish:~$ curl -fsSL https://gvisor.dev/archive.key | sudo gpg --dearmor -o /usr/share/keyrings/gvisor-archive-keyring.gpg
File '/usr/share/keyrings/gvisor-archive-keyring.gpg' exists. Overwrite? (y/N) y
useradd@Neha-Girish:~$ echo "deb [signed-by=/usr/share/keyrings/gvisor-archive-keyring.gpg] https://storage.googleapis.com/gvisor/releases release main" | s
udo tee /etc/apt/sources.list.d/gvisor.list > /dev/null
useradd@Neha-Girish:~$ sudo apt update
Hit:1 https://storage.googleapis.com/gvisor/releases release InRelease
Hit:2 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:3 http://archive.ubuntu.com/ubuntu noble InRelease
Get:4 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Hit:5 http://archive.ubuntu.com/ubuntu noble-backports InRelease
Fetched 126 kB in 1s (92.5 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
useradd@Neha-Girish:~$ sudo apt install -y runsc
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
runsc is already the newest version (20250407.0).
The following package was automatically installed and is no longer required:
  libllvm17t64
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
useradd@Neha-Girish:~$ sudo nano /etc/docker/daemon.json
```

Editing docker daemon.

Testing if gvisor is installed:

```
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/docker/base_images/python$ docker run --rm --runtime=runsc hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Create Packaging Mechanism for gVisor

Done this dynamically — no need to create new Dockerfiles/images.

We reuse same images (lambda-python, lambda-javascript)

When use_gvisor=True, runner.py adds --runtime=runsc

Update runner.py with execution engine support

```
import subprocess

import os
from queue import Queue

# Container pool setup
container_pool = Queue()
POOL_SIZE = 5

# List of functions to pre-warm
WARMED_FUNCTIONS = {
    "python": ["hello.py", "arithmetic.py", "infinite.py"],
    "javascript": ["hello.js", "arithmetic.js"]
}

def init_container_pool():
    for _ in range(POOL_SIZE):
        container_pool.put(None)  # Placeholder if you implement persistent
containers

def run_function(language, filename, args=None, timeout=5):
    container = f"lambda-{language}"
    func_path = os.path.abspath(os.path.join(os.path.dirname(__file__),
"../docker/functions", language))

    # Build the base command
    cmd = [
        "docker", "run", "--rm",
        "-v", f"{func_path}:/functions",
        "-e", f"FUNCTION_FILE={filename}",
    ]

    if args:
        import json
        cmd += ["-e", f"ARGS={json.dumps(args)}"]

    # Add gVisor runtime flag if needed
    if use_gvisor:
        cmd += ["--runtime=runsc"]


    # Add image name last
    cmd.append(container)
```

```
    try:
        print("Running:", " ".join(cmd))
        result = subprocess.run(cmd, capture_output=True, text=True,
timeout=timeout)
        return result.stdout.strip(), result.stderr.strip()
    except subprocess.TimeoutExpired:
        return "", f"ERROR: Execution timed out after {timeout} seconds"
    except Exception as e:
        return "", f"ERROR: {str(e)}"
    finally:
        container_pool.put(None)

def warm_up_functions():
    print("Warming up frequently used functions...")
    for lang in WARMED_FUNCTIONS:
        for fname in WARMED_FUNCTIONS[lang]:
            try:
                print(f" → {lang}/{fname}")
                run_function(lang, fname, timeout=3)
            except Exception as e:
                print(f"Warm-up failed for {lang}/{fname}: {str(e)}")

# Init on import
init_container_pool()
warm_up_functions()
```

Update schemas.py

```
from pydantic import BaseModel
from typing import List, Optional

class FunctionCreate(BaseModel):
    name: str
    language: str
    route: str
    timeout: int

class FunctionOut(FunctionCreate):
    id: int

    class Config:
        orm_mode = True

# Create a model for the /functions/execute request body
class ExecuteFunctionRequest(BaseModel):
```

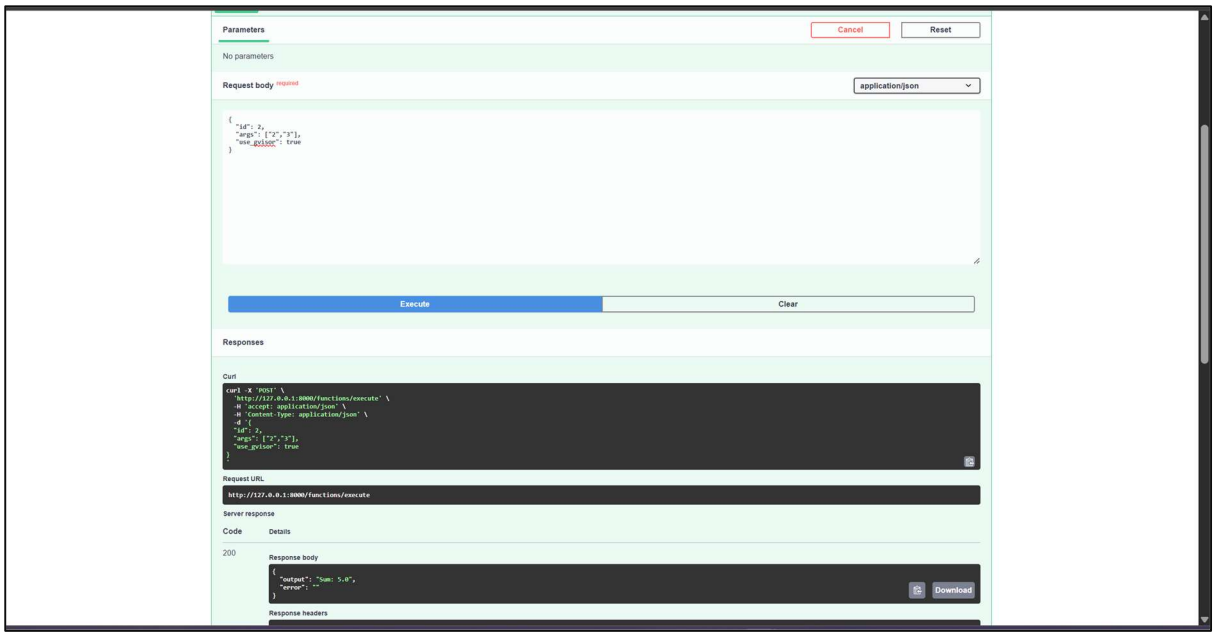
```
    id: int  # Expecting an 'id' field
    args: Optional[List[str]] = None
    use_gvisor: Optional[bool] = False
```

Update main.py

```
from fastapi import FastAPI, Depends, HTTPException, Request
from sqlalchemy.orm import Session
from . import models, schemas, crud, database
from .schemas import ExecuteFunctionRequest

import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
"../../executor")))
import runner  # this imports runner.py from the executor/ folder

models.Base.metadata.create_all(bind=database.engine)

app = FastAPI()

def get_db():
    db = database.SessionLocal()
    try:
        yield db
    finally:
        db.close()

@app.post("/functions", response_model=schemas.FunctionOut)
def create(function: schemas.FunctionCreate, db: Session = Depends(get_db)):
    try:
        return crud.create_function(db, function)
    except Exception as e:
        print(f"Error occurred: {str(e)}")  # Log the error for debugging
        raise HTTPException(status_code=500, detail="Internal Server Error")


@app.get("/functions", response_model=list[schemas.FunctionOut])
def read_all(db: Session = Depends(get_db)):
    return crud.get_functions(db)

@app.get("/functions/{function_id}", response_model=schemas.FunctionOut)
def read_one(function_id: int, db: Session = Depends(get_db)):
    func = crud.get_function(db, function_id)
    if not func:
        raise HTTPException(status_code=404, detail="Function not found")
```

```
    return func

@app.delete("/functions/{function_id}")
def delete(function_id: int, db: Session = Depends(get_db)):
    success = crud.delete_function(db, function_id)
    if not success:
        raise HTTPException(status_code=404, detail="Function not found")
    return {"deleted": True}

@app.post("/functions/execute")
def execute_function(request: ExecuteFunctionRequest, db: Session =
Depends(get_db)):
    function_id = request.id
    args = request.args or []
    use_gvisor = request.use_gvisor

    func = db.query(models.Function).filter(models.Function.id ==
function_id).first()
    if not func:
        raise HTTPException(status_code=404, detail="Function not found")

    output, error = runner.run_function(
        func.language, func.route,
        args=args,
        timeout=func.timeout,
        use_gvisor=use_gvisor
    )
    return {"output": output, "error": error}
```

Implementing using gvisor runtime:



Performance comparison using time command:

runc runtimes:

```
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$ time docker run --rm --runtime=runc -v "/mnt/d/neha/SEM 6/CC/341_346_35
3_359_LAMBDA_Serverless_Function/docker/functions:/functions" -e FUNCTION_FILE=python/hello.py lambda-python
Hello, World!

real    0m0.749s
user    0m0.025s
sys     0m0.000s
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$ time docker run --rm --runtime=runc -v "/mnt/d/neha/SEM 6/CC/341_346_35
3_359_LAMBDA_Serverless_Function/docker/functions:/functions" -e FUNCTION_FILE=python/arithmetic.py lambda-python
Usage: python arithmetic.py <num1> <num2>

real    0m0.772s
user    0m0.018s
sys     0m0.010s
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$ time docker run --rm --runtime=runc -v "/mnt/d/neha/SEM 6/CC/341_346_35
3_359_LAMBDA_Serverless_Function/docker/functions:/functions" -e FUNCTION_FILE=javascript/hello.js lambda-javascript
Hello, World!


real    0m0.899s
user    0m0.013s
sys     0m0.013s
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$ time docker run --rm --runtime=runc -v "/mnt/d/neha/SEM 6/CC/341_346_35
3_359_LAMBDA_Serverless_Function/docker/functions:/functions" -e FUNCTION_FILE=javascript/arithmetic.js lambda-javascript
Usage: node arithmetic.js <num1> <num2>


real    0m0.761s
user    0m0.016s
sys     0m0.008s
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$
```

runsc runtimes:

```
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$ time docker run --rm --runtime=runsc -v "/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/docker/functions:/functions" -e FUNCTION_FILE=python/hello.py lambda-python
Hello, World!

real    0m1.388s
user    0m0.026s
sys     0m0.000s
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$ time docker run --rm --runtime=runsc -v "/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/docker/functions:/functions" -e FUNCTION_FILE=python/arithmetic.py lambda-python
Usage: python arithmetic.py <num1> <num2>

real    0m1.439s
user    0m0.020s
sys     0m0.013s
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$ time docker run --rm --runtime=runsc -v "/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/docker/functions:/functions" -e FUNCTION_FILE=javascript/hello.js lambda-javascript
Hello, World!


real    0m1.721s
user    0m0.013s
sys     0m0.013s
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$ time docker run --rm --runtime=runsc -v "/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/docker/functions:/functions" -e FUNCTION_FILE=javascript/arithmetic.js lambda-javascript
Usage: node arithmetic.js <num1> <num2>


real    0m1.573s
user    0m0.010s
sys     0m0.015s
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$
```

Runtime comparisons:

Function File	Runtime	Real Time	User Time	Sys Time
python/hello.py	runc	0m0.749s	0m0.025s	0m0.000s
python/arithmetic.py	runc	0m0.772s	0m0.018s	0m0.010s
javascript/hello.js	runc	0m0.899s	0m0.013s	0m0.013s
javascript/arithmetic.js	runc	0m0.761s	0m0.016s	0m0.008s
python/hello.py	runsc	0m1.388s	0m0.026s	0m0.000s
python/arithmetic.py	runsc	0m1.439s	0m0.020s	0m0.013s
javascript/hello.js	runsc	0m1.721s	0m0.013s	0m0.013s
javascript/arithmetic.js	runsc	0m1.573s	0m0.010s	0m0.015s

- **Task 3:** Metrics Collection.
    - Implement metrics collection for function execution (response time, errors, resources)
    - Create storage mechanism for metrics
    - Implement basic aggregation of metrics

Deliverable: Almost a working system, with metric monitoring and 2 ways of virtualization.

update models.py

```python
from sqlalchemy import Column, Integer, Float, Boolean, String, DateTime, ForeignKey
from datetime import datetime
from .database import Base

class Function(Base):
    __tablename__ = "functions"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), index=True)
    language = Column(String(50))
```

```
    route = Column(String(100), unique=True)
    timeout = Column(Integer)


class ExecutionMetric(Base):
    __tablename__ = "metrics"
    id = Column(Integer, primary_key=True, index=True)
    function_id = Column(Integer, ForeignKey("functions.id"))
    timestamp = Column(DateTime, default=datetime.utcnow)
    execution_time = Column(Float)
    was_error = Column(Boolean)
    error_message = Column(String(255), nullable=True)
```

update schemas.py

```
from pydantic import BaseModel
from typing import List, Optional
from datetime import datetime


class FunctionCreate(BaseModel):
    name: str
    language: str
    route: str
    timeout: int

class FunctionOut(FunctionCreate):
    id: int

    class Config:
        orm_mode = True

# Create a model for the /functions/execute request body
class ExecuteFunctionRequest(BaseModel):
    id: int  # Expecting an 'id' field
    args: Optional[List[str]] = None
    use_gvisor: Optional[bool] = False

class MetricOut(BaseModel):
    function_id: int
    timestamp: datetime
    execution_time: float
    was_error: bool
    error_message: str | None = None

    class Config:
        orm_mode = True
```

Adding a crud function in crud.py

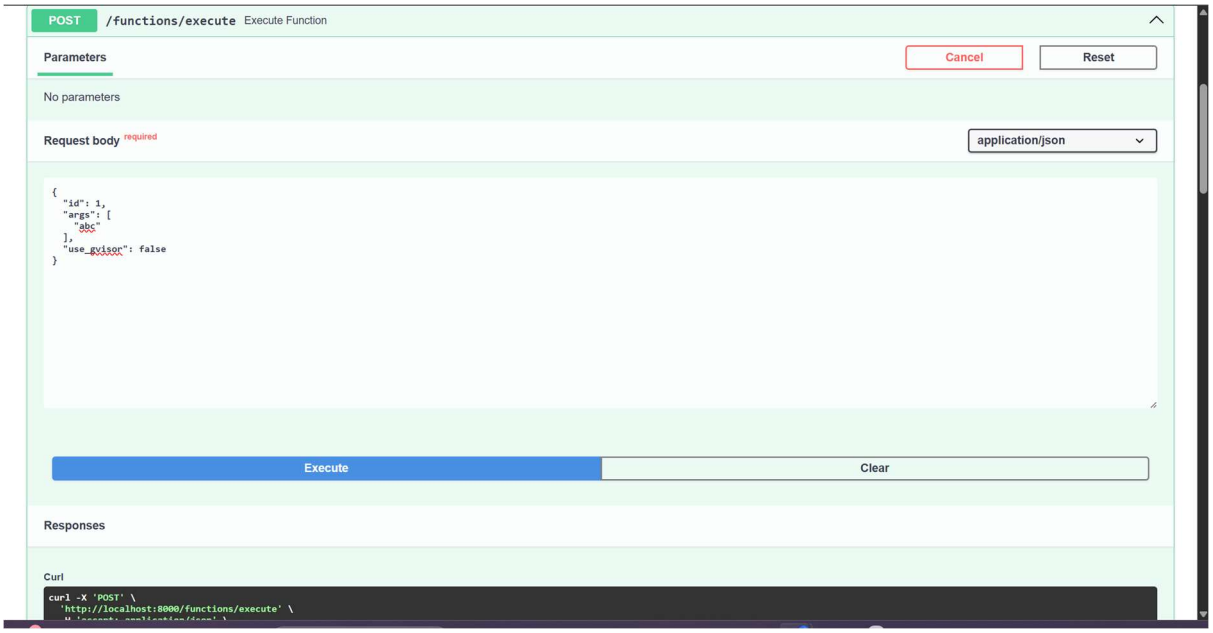
```
from sqlalchemy.orm import Session
from . import models, schemas
from .models import ExecutionMetric


def get_functions(db: Session):
    return db.query(models.Function).all()

def get_function(db: Session, function_id: int):
    return db.query(models.Function).filter(models.Function.id ==
function_id).first()

def create_function(db: Session, function: schemas.FunctionCreate):
    db_func = models.Function(**function.dict())
    db.add(db_func)
    db.commit()
    db.refresh(db_func)
    return db_func

def delete_function(db: Session, function_id: int):
    func = db.query(models.Function).filter(models.Function.id ==
function_id).first()
    if func:
        db.delete(func)
        db.commit()
        return True
    return False


def log_metric(db: Session, function_id: int, time_taken: float, was_error:
bool, error_message: str = None):
    metric = ExecutionMetric(
        function_id=function_id,
        execution_time=time_taken,
        was_error=was_error,
        error_message=error_message
    )
    db.add(metric)
    db.commit()
    db.refresh(metric)
    return metric
```

Update functions/execute and endpoint to view metrics in main.py:

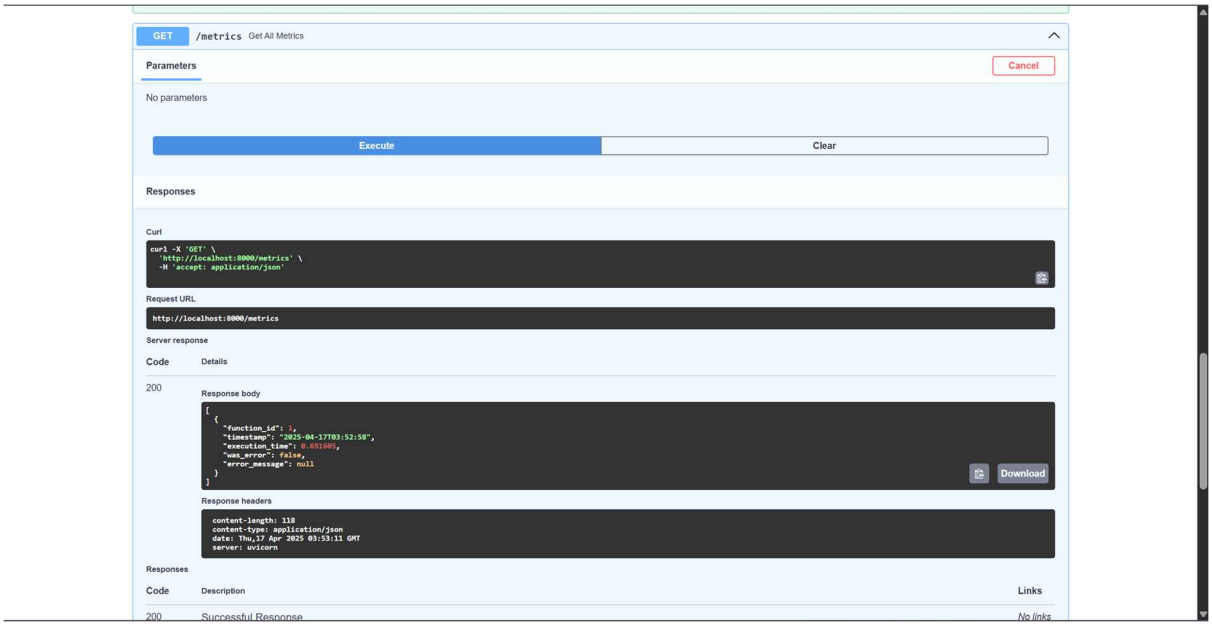
```
from fastapi import FastAPI, Depends, HTTPException, Request
from sqlalchemy.orm import Session
from . import models, schemas, crud, database
from .schemas import ExecuteFunctionRequest
import time
from .crud import log_metric


import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
"../../executor")))
import runner  #  this imports runner.py from the executor/ folder

models.Base.metadata.create_all(bind=database.engine)

app = FastAPI()

def get_db():
    db = database.SessionLocal()
    try:
        yield db
    finally:
        db.close()

@app.post("/functions", response_model=schemas.FunctionOut)
def create(function: schemas.FunctionCreate, db: Session = Depends(get_db)):
    try:
        return crud.create_function(db, function)
    except Exception as e:
        print(f"Error occurred: {str(e)}")  # Log the error for debugging
        raise HTTPException(status_code=500, detail="Internal Server Error")


@app.get("/functions", response_model=list[schemas.FunctionOut])
def read_all(db: Session = Depends(get_db)):
    return crud.get_functions(db)

@app.get("/functions/{function_id}", response_model=schemas.FunctionOut)
def read_one(function_id: int, db: Session = Depends(get_db)):
    func = crud.get_function(db, function_id)
    if not func:
        raise HTTPException(status_code=404, detail="Function not found")
    return func

@app.delete("/functions/{function_id}")
```

```
def delete(function_id: int, db: Session = Depends(get_db)):
    success = crud.delete_function(db, function_id)
    if not success:
        raise HTTPException(status_code=404, detail="Function not found")
    return {"deleted": True}

@app.post("/functions/execute")
def execute_function(request: ExecuteFunctionRequest, db: Session =
Depends(get_db)):
    function_id = request.id
    args = request.args or []
    use_gvisor = request.use_gvisor

    func = db.query(models.Function).filter(models.Function.id ==
function_id).first()
    if not func:
        raise HTTPException(status_code=404, detail="Function not found")

    start = time.time()
    output, error = runner.run_function(
        func.language, func.route,
        args=args,
        timeout=func.timeout,
        use_gvisor=use_gvisor
    )
    duration = time.time() - start
    was_error = bool(error)

    log_metric(db, function_id=func.id, time_taken=duration,
was_error=was_error, error_message=error if was_error else None)

    return {"output": output, "error": error}

#endpoint to view metrics
@app.get("/metrics", response_model=list[schemas.MetricOut])
def get_all_metrics(db: Session = Depends(get_db)):
    return db.query(models.ExecutionMetric).all()
```

Executing a function:



Checking GetMetrics:



Storage mechanism for metrics:

```
mysql> select * from metrics;
+----+-------------+---------------------+----------------+-----------+---------------+
| id | function_id | timestamp           | execution_time | was_error | error_message |
+----+-------------+---------------------+----------------+-----------+---------------+
|  1 |           1 | 2025-04-17 03:52:58 |       0.691605 |         0 | NULL          |
+----+-------------+---------------------+----------------+-----------+---------------+
1 row in set (0.00 sec)

mysql>
```

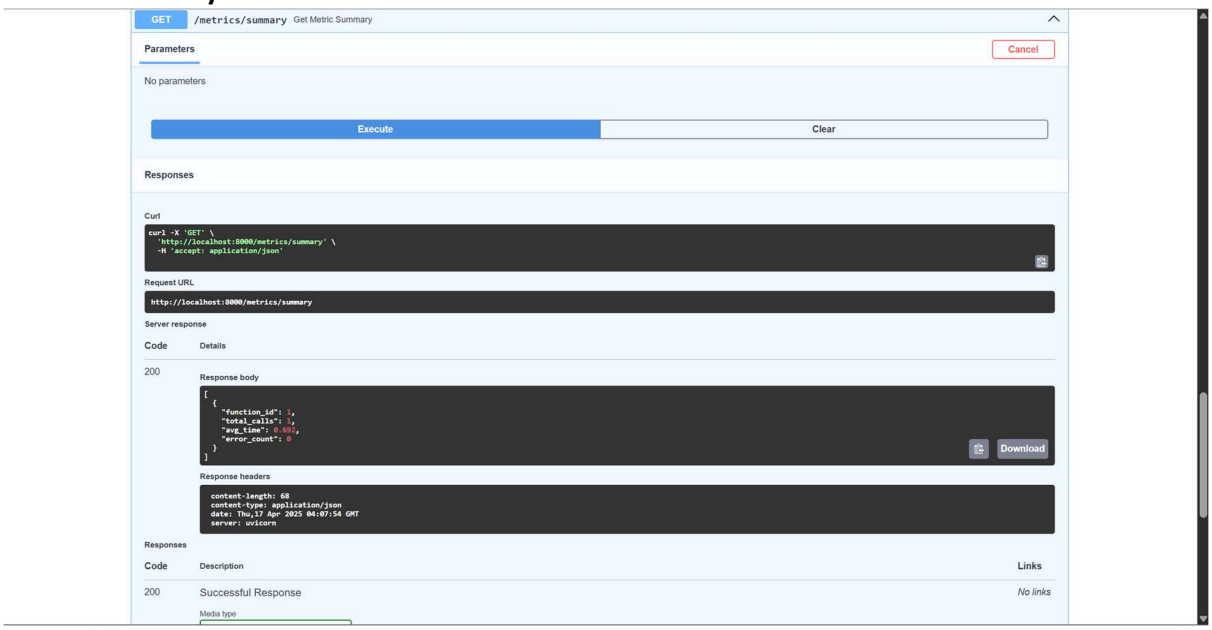
Metrics aggregation:
update main.py for metrics summary:

```
from fastapi import FastAPI, Depends, HTTPException, Request
from fastapi.responses import JSONResponse
from sqlalchemy.orm import Session
from sqlalchemy import func
#from fastapi.responses import JSONResponse
from . import models, schemas, crud, database
from .schemas import ExecuteFunctionRequest
import time
from .crud import log_metric


import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
"../../executor")))
import runner  #this imports runner.py from the executor/ folder
```

```
models.Base.metadata.create_all(bind=database.engine)

app = FastAPI()

def get_db():
    db = database.SessionLocal()
    try:
        yield db
    finally:
        db.close()

@app.post("/functions", response_model=schemas.FunctionOut)
def create(function: schemas.FunctionCreate, db: Session = Depends(get_db)):
    try:
        return crud.create_function(db, function)
    except Exception as e:
        print(f"Error occurred: {str(e)}")  # Log the error for debugging
        raise HTTPException(status_code=500, detail="Internal Server Error")


@app.get("/functions", response_model=list[schemas.FunctionOut])
def read_all(db: Session = Depends(get_db)):
    return crud.get_functions(db)

@app.get("/functions/{function_id}", response_model=schemas.FunctionOut)
def read_one(function_id: int, db: Session = Depends(get_db)):
    func = crud.get_function(db, function_id)
    if not func:
        raise HTTPException(status_code=404, detail="Function not found")
    return func

@app.delete("/functions/{function_id}")
def delete(function_id: int, db: Session = Depends(get_db)):
    success = crud.delete_function(db, function_id)
    if not success:
        raise HTTPException(status_code=404, detail="Function not found")
    return {"deleted": True}

@app.post("/functions/execute")
def execute_function(request: ExecuteFunctionRequest, db: Session =
Depends(get_db)):
    function_id = request.id
    args = request.args or []
    use_gvisor = request.use_gvisor

    func = db.query(models.Function).filter(models.Function.id ==
function_id).first()
```

```
    if not func:
        raise HTTPException(status_code=404, detail="Function not found")

    start = time.time()
    output, error = runner.run_function(
        func.language, func.route,
        args=args,
        timeout=func.timeout,
        use_gvisor=use_gvisor
    )
    duration = time.time() - start
    was_error = bool(error)

    log_metric(db, function_id=func.id, time_taken=duration,
was_error=was_error, error_message=error if was_error else None)

    return {"output": output, "error": error}

#endpoint to view metrics
@app.get("/metrics", response_model=list[schemas.MetricOut])
def get_all_metrics(db: Session = Depends(get_db)):
    return db.query(models.ExecutionMetric).all()

#Metrics aggregation
@app.get("/metrics/summary")
def get_metric_summary(db: Session = Depends(get_db)):
    summary = db.query(
        models.ExecutionMetric.function_id,
        func.count().label("total_calls"),
        func.avg(models.ExecutionMetric.execution_time).label("avg_time"),
        func.sum(func.if_(models.ExecutionMetric.was_error == True, 1,
0)).label("error_count")
    ).group_by(models.ExecutionMetric.function_id).all()

    results = [
        {
            "function_id": s.function_id,
            "total_calls": s.total_calls,
            "avg_time": round(s.avg_time, 3),
            "error_count": int(s.error_count)
        } for s in summary
    ]
    return JSONResponse(content=results)
```

Metrics Summary:



Week 3: Frontend, Monitoring Dashboard, and Integration (15 M)

Objective: Make the system scalable and fault-tolerant.

- **Task 1:** Basic Frontend
  - Create frontend application structure (Streamlit or similar)
  - Implement function deployment interface
  - Create function management views (list, create, update, delete)

Creating app.py, creating the virtual environment and installing the necessary dependencies

```
useradd@Neha-Girish:~$ cd "/mnt/d/neha/SEM 6/CC"
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC$  cd "341_346_353_359_LAMBDA_Serverless_Function"
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function$ cd frontend
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/frontend$ touch app.py
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/frontend$ ls
app.py
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/frontend$ python3 -m venv venv
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/frontend$ source venv/bin/activate
(venv) useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/frontend$ pip install streamlit requests
Collecting streamlit
  Downloading streamlit-1.44.1-py3-none-any.whl.metadata (8.9 kB)
Collecting requests
  Using cached requests-2.32.3-py3-none-any.whl.metadata (4.6 kB)
Collecting altair<6,>=4.0 (from streamlit)
  Downloading altair-5.5.0-py3-none-any.whl.metadata (11 kB)
Collecting blinker<2,>=1.0.0 (from streamlit)
  Using cached blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Collecting cachetools<6,>=4.0 (from streamlit)
  Downloading cachetools-5.5.2-py3-none-any.whl.metadata (5.4 kB)
Collecting click<9,>=7.0 (from streamlit)
  Using cached click-8.1.8-py3-none-any.whl.metadata (2.3 kB)
Collecting numpy<3,>=1.23 (from streamlit)
  Downloading numpy-2.2.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 1.9 MB/s eta 0:00:00
Collecting packaging<25,>=20 (from streamlit)
  Downloading packaging-24.2-py3-none-any.whl.metadata (3.2 kB)
Collecting pandas<3,>=1.4.0 (from streamlit)
  Downloading pandas-2.2.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (89 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 89.9/89.9 kB 2.7 MB/s eta 0:00:00
Collecting pillow<12,>=7.1.0 (from streamlit)
  Downloading pillow-11.2.1-cp312-cp312-manylinux_2_28_x86_64.whl.metadata (8.9 kB)
Collecting protobuf<6,>=3.20 (from streamlit)
  Downloading protobuf-5.29.4-cp38-abi3-manylinux2014_x86_64.whl.metadata (592 bytes)
Collecting pyarrow>=7.0 (from streamlit)
  Downloading pyarrow-19.0.1-cp312-cp312-manylinux_2_28_x86_64.whl.metadata (3.3 kB)
Collecting tenacity<10,>=8.1.0 (from streamlit)
  Downloading tenacity-9.1.2-py3-none-any.whl.metadata (1.2 kB)
Collecting toml<2,>=0.10.1 (from streamlit)
  Downloading toml-0.10.2-py2.py3-none-any.whl.metadata (7.1 kB)
Collecting typing-extensions<5,>=4.4.0 (from streamlit)
  Using cached typing_extensions-4.13.2-py3-none-any.whl.metadata (3.0 kB)
Collecting watchdog<7,>=2.1.5 (from streamlit)
  Downloading watchdog-6.0.0-py3-none-manylinux2014_x86_64.whl.metadata (44 kB)
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.1/42.1 MB 13.5 MB/s eta 0:00:00
Downloading pydeck-0.9.1-py2.py3-none-any.whl (6.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 14.9 MB/s eta 0:00:00
Downloading tenacity-9.1.2-py3-none-any.whl (28 kB)
Downloading toml-0.10.2-py2.py3-none-any.whl (16 kB)
Downloading tornado-6.4.2-cp38-abi3-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (437 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 437.2/437.2 kB 13.1 MB/s eta 0:00:00
Using cached typing_extensions-4.13.2-py3-none-any.whl (45 kB)
Downloading urllib3-2.4.0-py3-none-any.whl (128 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 128.7/128.7 kB 5.6 MB/s eta 0:00:00
Downloading watchdog-6.0.0-py3-none-manylinux2014_x86_64.whl (79 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.1/79.1 kB 5.3 MB/s eta 0:00:00
Downloading gitdb-4.0.12-py3-none-any.whl (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.8/62.8 kB 4.4 MB/s eta 0:00:00
Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.9/134.9 kB 7.7 MB/s eta 0:00:00
Downloading jsonschema-4.23.0-py3-none-any.whl (88 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 88.5/88.5 kB 6.4 MB/s eta 0:00:00
Downloading narwhals-1.35.0-py3-none-any.whl (325 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 325.7/325.7 kB 10.1 MB/s eta 0:00:00
Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 229.9/229.9 kB 8.6 MB/s eta 0:00:00
Downloading pytz-2025.2-py2.py3-none-any.whl (509 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 509.2/509.2 kB 11.5 MB/s eta 0:00:00
Downloading tzdata-2025.2-py2.py3-none-any.whl (347 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 347.8/347.8 kB 11.5 MB/s eta 0:00:00
Downloading attrs-25.3.0-py3-none-any.whl (63 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.8/63.8 kB 4.7 MB/s eta 0:00:00
Downloading jsonschema_specifications-2024.10.1-py3-none-any.whl (18 kB)
Using cached MarkupSafe-3.0.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (23 kB)
Downloading referencing-0.36.2-py3-none-any.whl (26 kB)
Downloading rpds_py-0.24.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (393 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 393.7/393.7 kB 10.4 MB/s eta 0:00:00
Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
Downloading smmap-5.0.2-py3-none-any.whl (24 kB)
Installing collected packages: pytz, watchdog, urllib3, tzdata, typing-extensions, tornado, toml, tenacity, smmap, six, rpds-py, pyarrow, protobuf, pillow,
packaging, numpy, narwhals, MarkupSafe, idna, click, charset-normalizer, certifi, cachetools, blinker, attrs, requests, referencing, python-dateutil, jinja2
, gitdb, pydeck, pandas, jsonschema-specifications, gitpython, jsonschema, altair, streamlit
Successfully installed MarkupSafe-3.0.2 altair-5.5.0 attrs-25.3.0 blinker-1.9.0 cachetools-5.5.2 certifi-2025.1.31 charset-normalizer-3.4.1 click-8.1.8 gitd
b-4.0.12 gitpython-3.1.44 idna-3.10 jinja2-3.1.6 jsonschema-4.23.0 jsonschema-specifications-2024.10.1 narwhals-1.35.0 numpy-2.2.4 packaging-24.2 pandas-2.2
.3 pillow-11.2.1 protobuf-5.29.4 pyarrow-19.0.1 pydeck-0.9.1 python-dateutil-2.9.0.post0 pytz-2025.2 referencing-0.36.2 requests-2.32.3 rpds-py-0.24.0 six-1
.17.0 smmap-5.0.2 streamlit-1.44.1 tenacity-9.1.2 toml-0.10.2 tornado-6.4.2 typing-extensions-4.13.2 tzdata-2025.2 urllib3-2.4.0 watchdog-6.0.0
(venv) useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/frontend$
```

```
(venv) useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/frontend$ streamlit --version
Streamlit, version 1.44.1
```

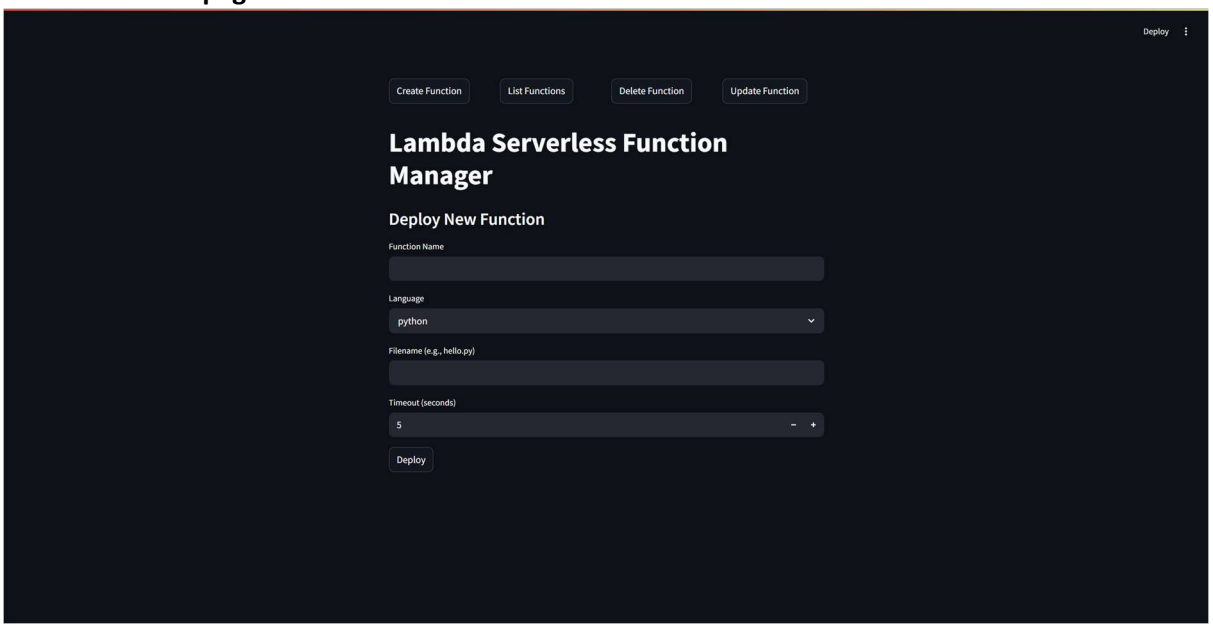
frontend/app.py:

```
import streamlit as st
import requests

API_URL = "http://127.0.0.1:8000"

# Initialize session state
if "page" not in st.session_state:
    st.session_state.page = "Create Function"

# Menu bar with buttons
st.markdown("""
    <style>
        .menu {
            display: flex;
            gap: 20px;
            padding: 10px 0;
            border-bottom: 1px solid #ccc;
            margin-bottom: 20px;
        }
        .menu-button {
            background-color: #f0f2f6;
            border: none;
            color: #333;
            padding: 10px 20px;
            cursor: pointer;
            font-weight: bold;
        }
        .menu-button:hover {
            background-color: #e0e2e6;
        }
    </style>
""", unsafe_allow_html=True)

col1, col2, col3, col4 = st.columns(4)

with col1:
    if st.button("Create Function"):
        st.session_state.page = "Create Function"
with col2:
    if st.button("List Functions"):
        st.session_state.page = "List Functions"
with col3:
    if st.button("Delete Function"):
        st.session_state.page = "Delete Function"
with col4:
    if st.button("Update Function"):
```

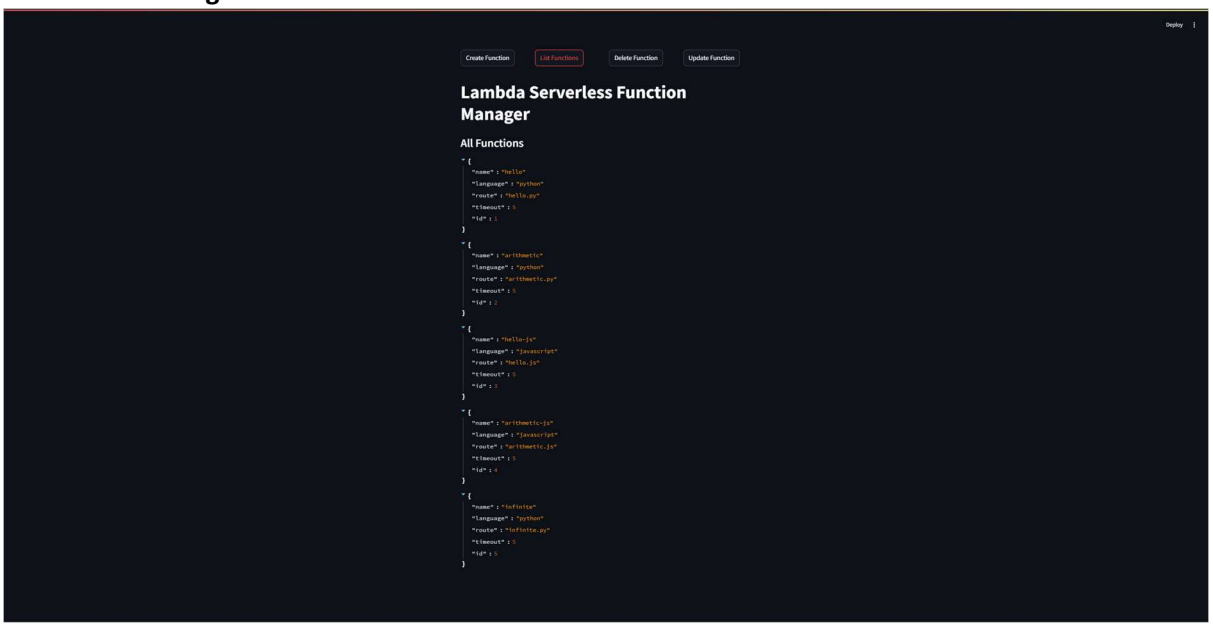
```
        st.session_state.page = "Update Function"
# Main content
st.title("Lambda Serverless Function Manager")

# Create Function Page
if st.session_state.page == "Create Function":
    st.subheader("Deploy New Function")

    name = st.text_input("Function Name")
    language = st.selectbox("Language", ["python", "javascript"])
    route = st.text_input("Filename (e.g., hello.py)")
    timeout = st.number_input("Timeout (seconds)", min_value=1, max_value=30,
value=5)

    if st.button("Deploy"):
        payload = {
            "name": name,
            "language": language,
            "route": route,
            "timeout": timeout
        }
        response = requests.post(f"{API_URL}/functions", json=payload)
        if response.status_code == 200:
            st.success("Function deployed successfully!")
        else:
            st.error(f"Failed: {response.text}")

# List Functions Page
elif st.session_state.page == "List Functions":
    st.subheader("All Functions")
    response = requests.get(f"{API_URL}/functions")
    if response.ok:
        for func in response.json():
            st.json(func)
    else:
        st.error("Could not fetch functions.")

# Delete Function Page
elif st.session_state.page == "Delete Function":
    st.subheader("Delete Function")
    response = requests.get(f"{API_URL}/functions")
    if response.ok:
        functions = response.json()
        choices = {f"{f['name']} (ID: {f['id']})": f['id'] for f in functions}
        selected = st.selectbox("Select Function to Delete",
list(choices.keys()))
        if st.button("Delete"):
```

```
            fid = choices[selected]
            del_res = requests.delete(f"{API_URL}/functions/{fid}")
            if del_res.status_code == 200:
                st.success("Deleted successfully!")
            else:
                st.error("Delete failed.")
    else:
        st.error("Could not load functions.")
#Update Function Page
# Update Function Page
elif st.session_state.page == "Update Function":
    st.subheader("✏️ Update Existing Function")

    response = requests.get(f"{API_URL}/functions")
    if response.ok:
        functions = response.json()
        choices = {f"{f['name']} (ID: {f['id']})": f for f in functions}
        selected = st.selectbox("Select Function to Update",
list(choices.keys()))

        func_data = choices[selected]
        updated_name = st.text_input("New Name", value=func_data["name"])
        updated_lang = st.selectbox("Language", ["python", "javascript"],
index=["python", "javascript"].index(func_data["language"]))
        updated_route = st.text_input("Filename", value=func_data["route"])
        updated_timeout = st.number_input("Timeout", min_value=1,
max_value=30, value=func_data["timeout"])

        if st.button("Update"):
            payload = {
                "name": updated_name,
                "language": updated_lang,
                "route": updated_route,
                "timeout": updated_timeout
            }
            res = requests.put(f"{API_URL}/functions/{func_data['id']}",
json=payload)
            if res.status_code == 200:
                st.success("Function updated successfully!")
            else:
                st.error(f"Update failed: {res.text}")
    else:
        st.error("Could not fetch functions.")
```

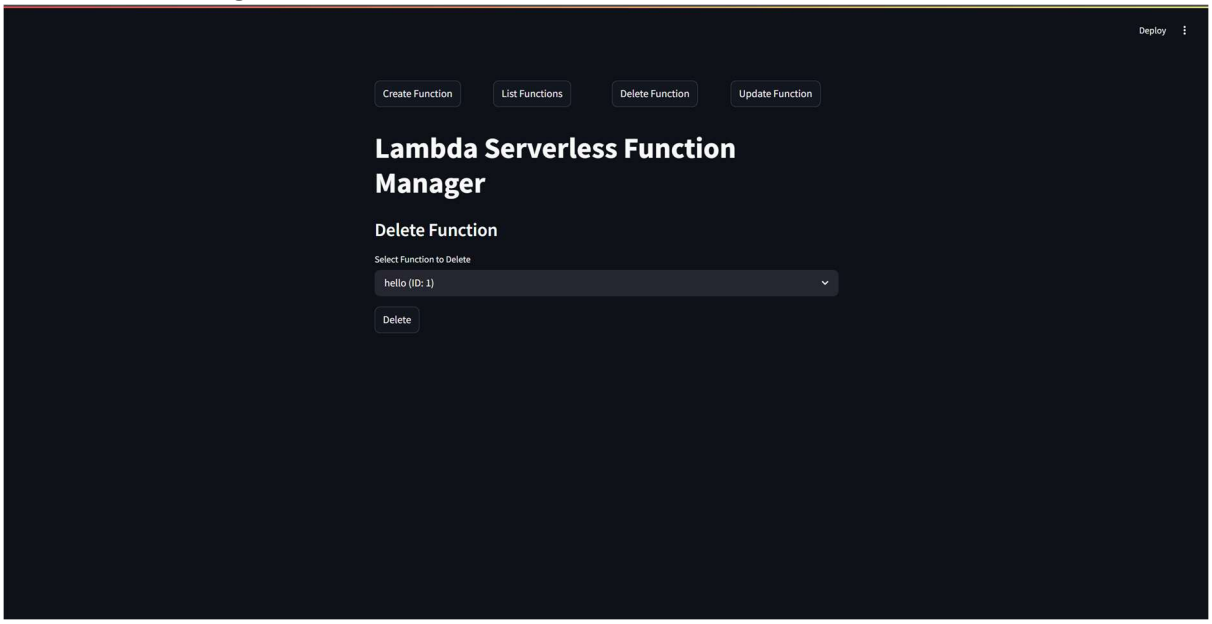
Create Function page:



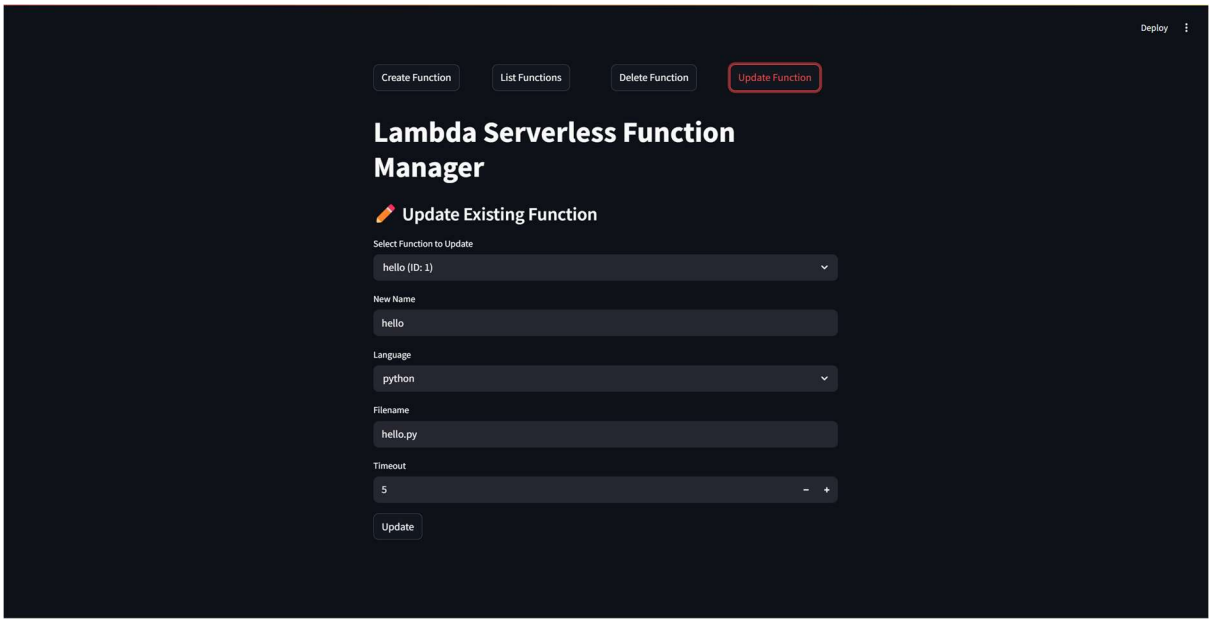
List Functions Page:



Delete Function Page:



Update Function



Update app.py for function execution

```
import streamlit as st

import requests

API_URL = "http://127.0.0.1:8000"

# Initialize session state
if "page" not in st.session_state:
    st.session_state.page = "Create Function"
```

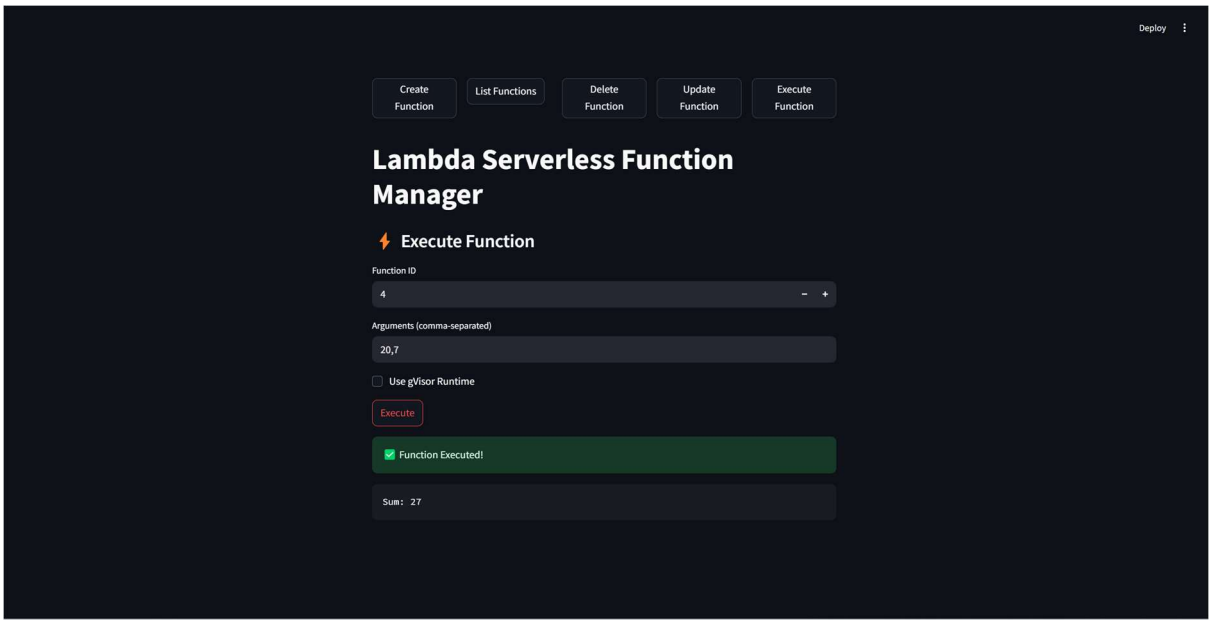
```
# Menu bar with buttons
st.markdown("""
    <style>
        .menu {
            display: flex;
            gap: 20px;
            padding: 10px 0;
            border-bottom: 1px solid #ccc;
            margin-bottom: 20px;
        }
        .menu-button {
            background-color: #f0f2f6;
            border: none;
            color: #333;
            padding: 10px 20px;
            cursor: pointer;
            font-weight: bold;
        }
        .menu-button:hover {
            background-color: #e0e2e6;
        }
    </style>
""", unsafe_allow_html=True)

col1, col2, col3, col4, col5 = st.columns(5)

with col1:
    if st.button("Create Function"):
        st.session_state.page = "Create Function"
with col2:
    if st.button("List Functions"):
        st.session_state.page = "List Functions"
with col3:
    if st.button("Delete Function"):
        st.session_state.page = "Delete Function"
with col4:
    if st.button("Update Function"):
        st.session_state.page = "Update Function"
with col5:
    if st.button("Execute Function"):
        st.session_state.page = "Execute Function"
# Main content
st.title("Lambda Serverless Function Manager")

# Create Function Page
if st.session_state.page == "Create Function":
```

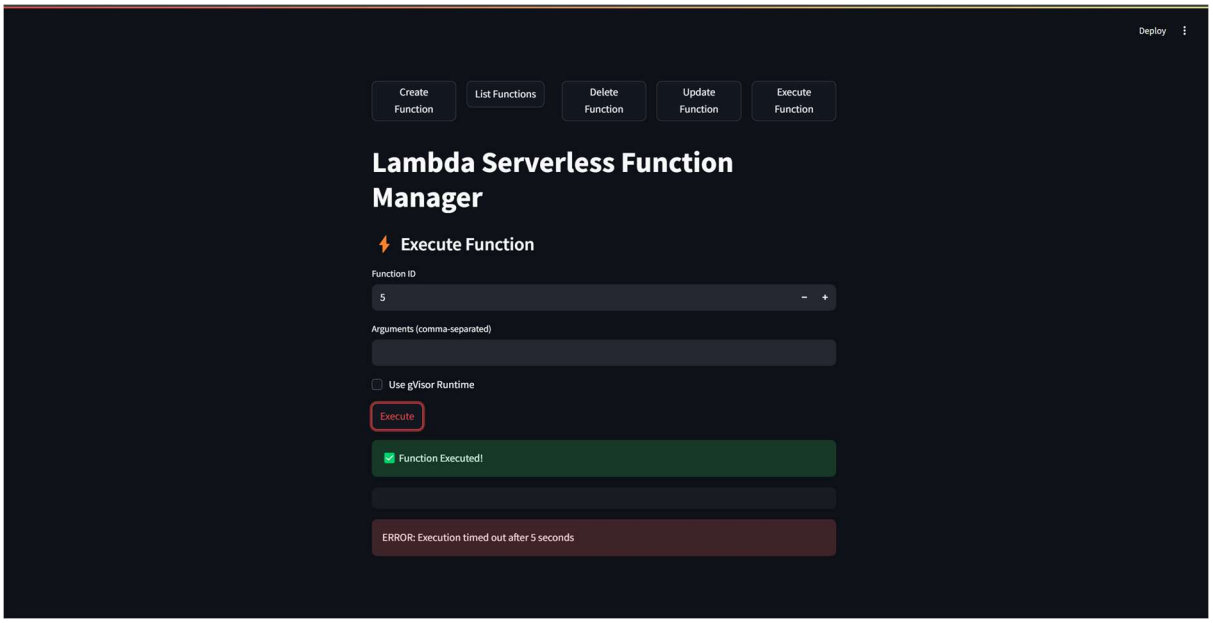
```
    st.subheader("Deploy New Function")

    name = st.text_input("Function Name")
    language = st.selectbox("Language", ["python", "javascript"])
    route = st.text_input("Filename (e.g., hello.py)")
    timeout = st.number_input("Timeout (seconds)", min_value=1, max_value=30,
value=5)

    if st.button("Deploy"):
        payload = {
            "name": name,
            "language": language,
            "route": route,
            "timeout": timeout
        }
        response = requests.post(f"{API_URL}/functions", json=payload)
        if response.status_code == 200:
            st.success("Function deployed successfully!")
        else:
            st.error(f"Failed: {response.text}")

# List Functions Page
elif st.session_state.page == "List Functions":
    st.subheader("All Functions")
    response = requests.get(f"{API_URL}/functions")
    if response.ok:
        for func in response.json():
            st.json(func)
    else:
        st.error("Could not fetch functions.")

# Delete Function Page
elif st.session_state.page == "Delete Function":
    st.subheader("Delete Function")
    response = requests.get(f"{API_URL}/functions")
    if response.ok:
        functions = response.json()
        choices = {f"{f['name']} (ID: {f['id']})": f['id'] for f in functions}
        selected = st.selectbox("Select Function to Delete",
list(choices.keys()))
        if st.button("Delete"):
            fid = choices[selected]
            del_res = requests.delete(f"{API_URL}/functions/{fid}")
            if del_res.status_code == 200:
                st.success("Deleted successfully!")
            else:
                st.error("Delete failed.")
```

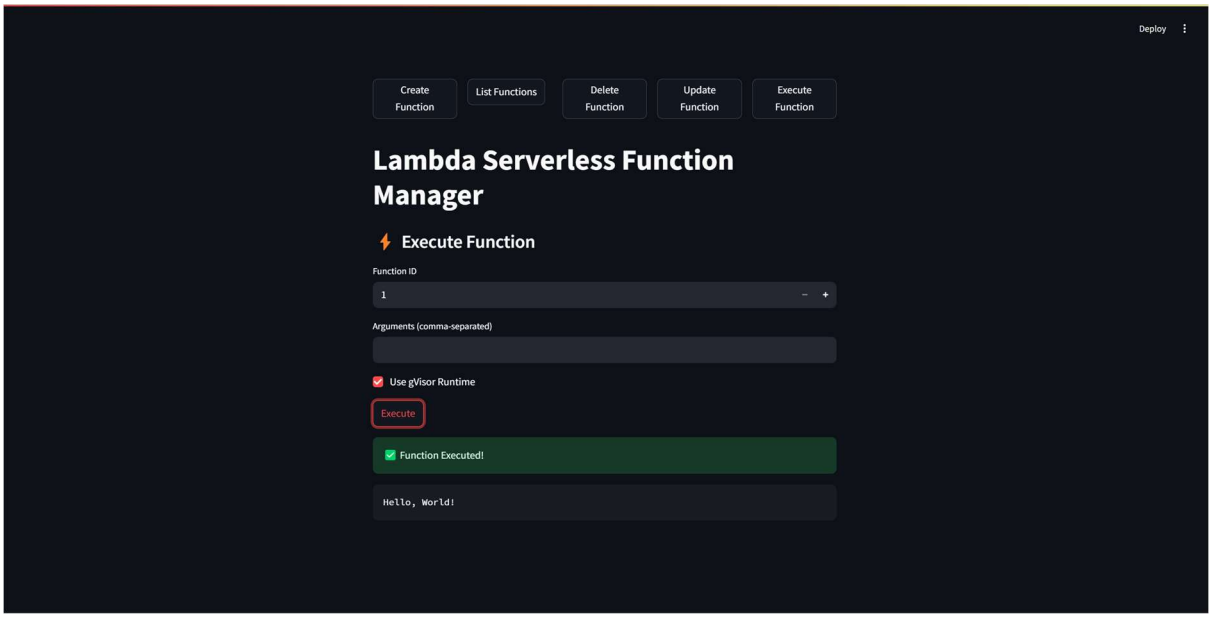
```
    else:
        st.error("Could not load functions.")

# Update Function Page
elif st.session_state.page == "Update Function":
    st.subheader("✏️ Update Existing Function")

    response = requests.get(f"{API_URL}/functions")
    if response.ok:
        functions = response.json()
        choices = {f"{f['name']} (ID: {f['id']})": f for f in functions}
        selected = st.selectbox("Select Function to Update",
list(choices.keys()))

        func_data = choices[selected]
        updated_name = st.text_input("New Name", value=func_data["name"])
        updated_lang = st.selectbox("Language", ["python", "javascript"],
index=["python", "javascript"].index(func_data["language"]))
        updated_route = st.text_input("Filename", value=func_data["route"])
        updated_timeout = st.number_input("Timeout", min_value=1,
max_value=30, value=func_data["timeout"])

        if st.button("Update"):
            payload = {
                "name": updated_name,
                "language": updated_lang,
                "route": updated_route,
                "timeout": updated_timeout
            }
            res = requests.put(f"{API_URL}/functions/{func_data['id']}",
json=payload)
            if res.status_code == 200:
                st.success("Function updated successfully!")
            else:
                st.error(f"Update failed: {res.text}")
    else:
        st.error("Could not fetch functions.")
# Execute Function Page
elif st.session_state.page == "Execute Function":
    st.subheader("⚡ Execute Function")

    function_id = st.number_input("Function ID", min_value=1, step=1)
    raw_args = st.text_input("Arguments (comma-separated)", value="")
    use_gvisor = st.checkbox("Use gVisor Runtime", value=False)

    if st.button("Execute"):
        args = [arg.strip() for arg in raw_args.split(",") if arg.strip()]
```

```
        payload = {
            "id": int(function_id),
            "args": args,
            "use_gvisor": use_gvisor
        }
        response = requests.post(f"{API_URL}/functions/execute", json=payload)
        if response.status_code == 200:
            data = response.json()
            st.success("Function Executed!")
            st.code(data.get("output", ""), language="text")
            if data.get("error"):
                st.error(data["error"])
        else:
            st.error("Function execution failed.")
```

Execute Function page:







- **Task 2:** Monitoring Dashboard
  - Implement metrics visualization components
  - Create dashboard views for individual function performance
  - Implement system-wide statistics view

pip install pandas first

```
useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/frontend$ source venv/bin/activate
(venv) useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/frontend$ pip install pandas
Requirement already satisfied: pandas in ./venv/lib/python3.12/site-packages (2.2.3)
Requirement already satisfied: numpy>=1.26.0 in ./venv/lib/python3.12/site-packages (from pandas) (2.2.4)
Requirement already satisfied: python-dateutil>=2.8.2 in ./venv/lib/python3.12/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in ./venv/lib/python3.12/site-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in ./venv/lib/python3.12/site-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in ./venv/lib/python3.12/site-packages (from python-dateutil>=2.8.2->pandas) (1.17.0
(venv) useradd@Neha-Girish:/mnt/d/neha/SEM 6/CC/341_346_353_359_LAMBDA_Serverless_Function/frontend$
```

Update app.py in frontend:

```
import streamlit as st
import requests
import pandas as pd

API_URL = "http://127.0.0.1:8000"

# Initialize session state
if "page" not in st.session_state:
    st.session_state.page = "Create Function"

# Menu bar with buttons
st.markdown("""
    <style>
        .menu {
            display: flex;
            gap: 20px;
            padding: 10px 0;
            border-bottom: 1px solid #ccc;
            margin-bottom: 20px;
        }
        .menu-button {
            background-color: #f0f2f6;
            border: none;
            color: #333;
            padding: 10px 20px;
            cursor: pointer;
            font-weight: bold;
        }
        .menu-button:hover {
            background-color: #e0e2e6;
        }
    </style>
""", unsafe_allow_html=True)

col1, col2, col3, col4, col5, col6 = st.columns(6)

with col1:
    if st.button("Create Function"):
```

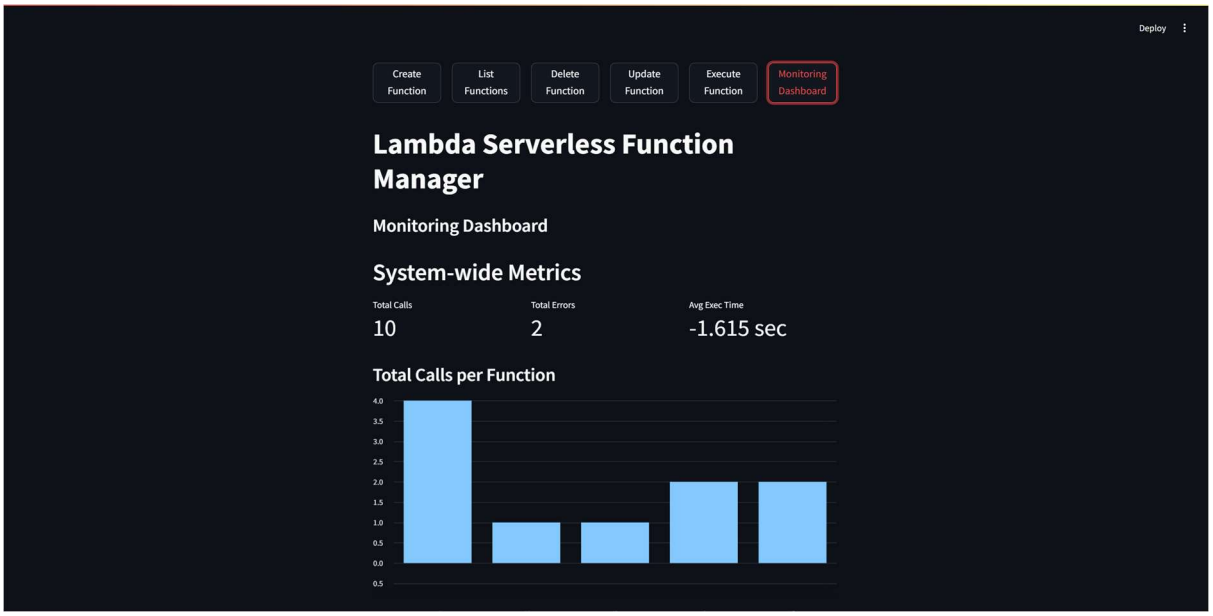
```
        st.session_state.page = "Create Function"
with col2:
    if st.button("List Functions"):
        st.session_state.page = "List Functions"
with col3:
    if st.button("Delete Function"):
        st.session_state.page = "Delete Function"
with col4:
    if st.button("Update Function"):
        st.session_state.page = "Update Function"
with col5:
    if st.button("Execute Function"):
        st.session_state.page = "Execute Function"
with col6:
    if st.button("Monitoring Dashboard"):
        st.session_state.page = "Monitoring Dashboard"

# Main content
st.title("Lambda Serverless Function Manager")

# Create Function Page
if st.session_state.page == "Create Function":
    st.subheader("Deploy New Function")

    name = st.text_input("Function Name")
    language = st.selectbox("Language", ["python", "javascript"])
    route = st.text_input("Filename (e.g., hello.py)")
    timeout = st.number_input("Timeout (seconds)", min_value=1, max_value=30,
value=5)

    if st.button("Deploy"):
        payload = {
            "name": name,
            "language": language,
            "route": route,
            "timeout": timeout
        }
        response = requests.post(f"{API_URL}/functions", json=payload)
        if response.status_code == 200:
            st.success("Function deployed successfully!")
        else:
            st.error(f"Failed: {response.text}")

# List Functions Page
elif st.session_state.page == "List Functions":
    st.subheader("All Functions")
    response = requests.get(f"{API_URL}/functions")
```

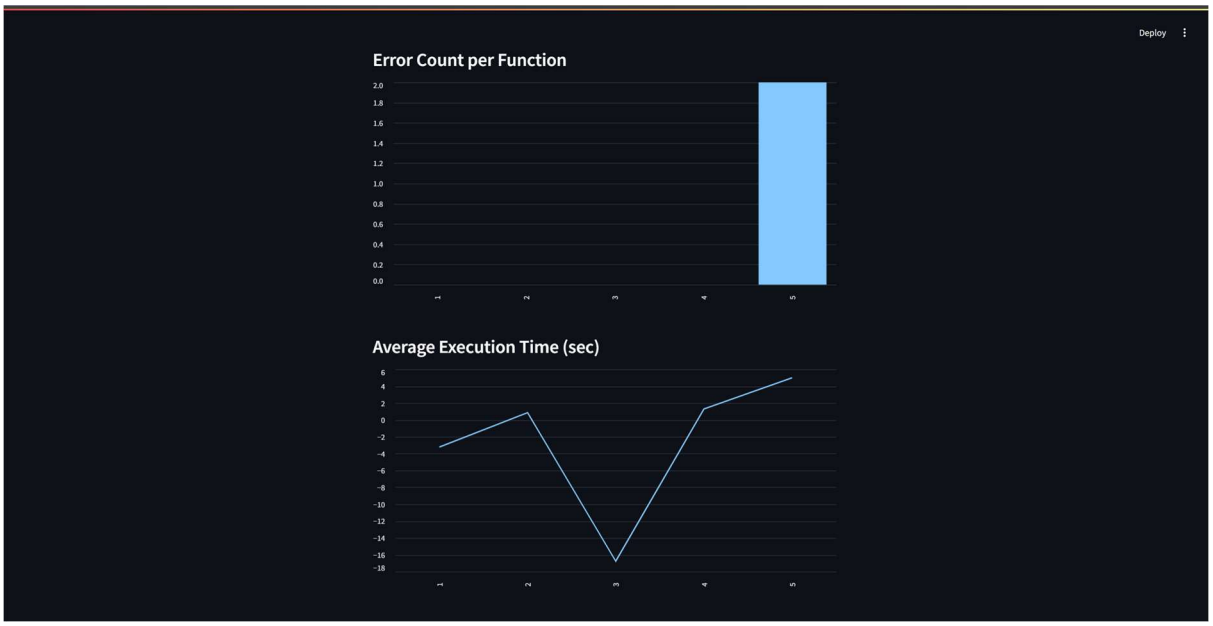
```
    if response.ok:
        for func in response.json():
            st.json(func)
    else:
        st.error("Could not fetch functions.")

# Delete Function Page
elif st.session_state.page == "Delete Function":
    st.subheader("Delete Function")
    response = requests.get(f"{API_URL}/functions")
    if response.ok:
        functions = response.json()
        choices = {f"{f['name']} (ID: {f['id']})": f['id'] for f in functions}
        selected = st.selectbox("Select Function to Delete",
list(choices.keys()))
        if st.button("Delete"):
            fid = choices[selected]
            del_res = requests.delete(f"{API_URL}/functions/{fid}")
            if del_res.status_code == 200:
                st.success("Deleted successfully!")
            else:
                st.error("Delete failed.")
    else:
        st.error("Could not load functions.")

# Update Function Page
elif st.session_state.page == "Update Function":
    st.subheader("✏️ Update Existing Function")

    response = requests.get(f"{API_URL}/functions")
    if response.ok:
        functions = response.json()
        choices = {f"{f['name']} (ID: {f['id']})": f for f in functions}
        selected = st.selectbox("Select Function to Update",
list(choices.keys()))

        func_data = choices[selected]
        updated_name = st.text_input("New Name", value=func_data["name"])
        updated_lang = st.selectbox("Language", ["python", "javascript"],
index=["python", "javascript"].index(func_data["language"]))
        updated_route = st.text_input("Filename", value=func_data["route"])
        updated_timeout = st.number_input("Timeout", min_value=1,
max_value=30, value=func_data["timeout"])

        if st.button("Update"):
            payload = {
                "name": updated_name,
```

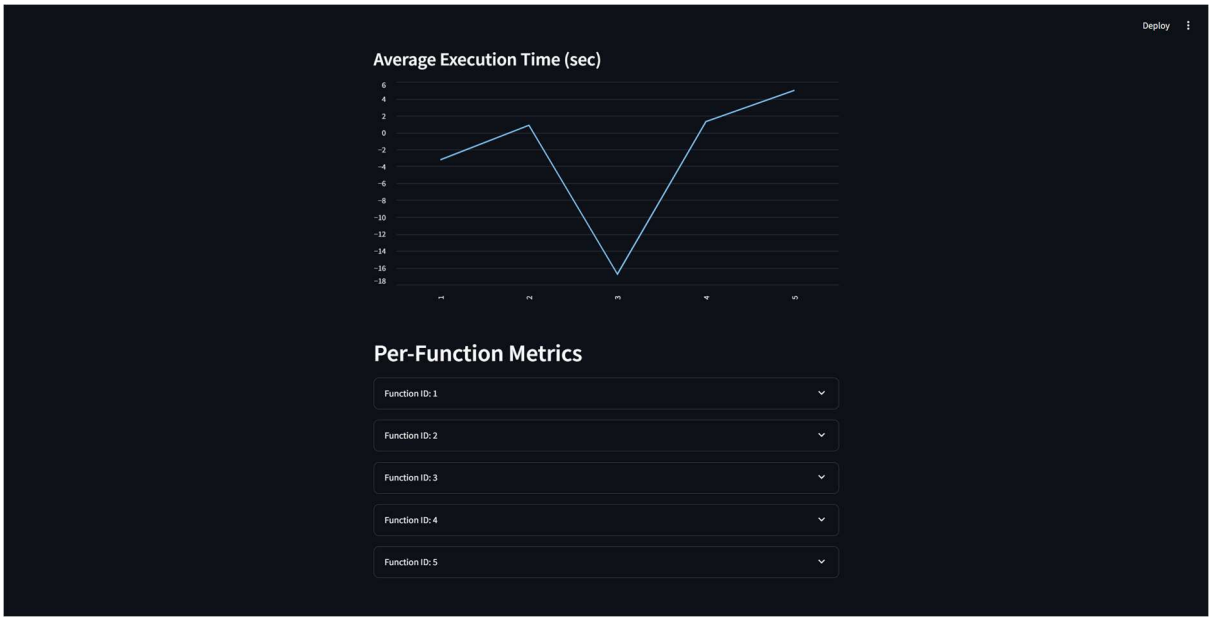
```
                "language": updated_lang,
                "route": updated_route,
                "timeout": updated_timeout
            }
            res = requests.put(f"{API_URL}/functions/{func_data['id']}",
json=payload)
            if res.status_code == 200:
                st.success("Function updated successfully!")
            else:
                st.error(f"Update failed: {res.text}")
    else:
        st.error("Could not fetch functions.")
# Execute Function Page
elif st.session_state.page == "Execute Function":
    st.subheader(" ⚡ Execute Function")

    function_id = st.number_input("Function ID", min_value=1, step=1)
    raw_args = st.text_input("Arguments (comma-separated)", value="")
    use_gvisor = st.checkbox("Use gVisor Runtime", value=False)

    if st.button("Execute"):
        args = [arg.strip() for arg in raw_args.split(",") if arg.strip()]
        payload = {
            "id": int(function_id),
            "args": args,
            "use_gvisor": use_gvisor
        }
        response = requests.post(f"{API_URL}/functions/execute", json=payload)
        if response.status_code == 200:
            data = response.json()
            st.success("☑ Function Executed!")
            st.code(data.get("output", ""), language="text")
            if data.get("error"):
                st.error(data["error"])
        else:
            st.error("Function execution failed.")

# Dashboard Page
elif st.session_state.page == "Monitoring Dashboard":
    st.subheader(" Monitoring Dashboard")

    try:
        response = requests.get(f"{API_URL}/metrics/summary")
        if response.status_code != 200:
            st.warning("Could not load metrics.")
        else:
            summary = response.json()
```

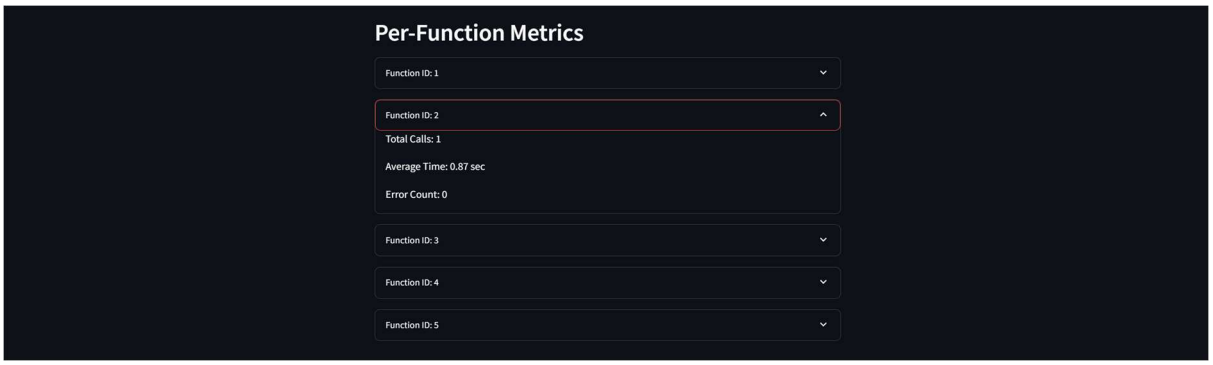
```
            if not summary:
                st.info("No metrics available yet.")
            else:
                df = pd.DataFrame(summary)
                df["function"] = df["function_id"].astype(str)

                # System-wide overview
                st.write("## System-wide Metrics")
                total_calls = df["total_calls"].sum()
                total_errors = df["error_count"].sum()
                avg_exec_time = round((df["avg_time"] *
df["total_calls"]).sum() / total_calls, 3) if total_calls else 0

                col1, col2, col3 = st.columns(3)
                col1.metric("Total Calls", total_calls)
                col2.metric("Total Errors", total_errors)
                col3.metric("Avg Exec Time", f"{avg_exec_time} sec")

                # Charts
                st.write("###  Total Calls per Function")
                st.bar_chart(df.set_index("function")["total_calls"])

                st.write("###  Error Count per Function")
                st.bar_chart(df.set_index("function")["error_count"])

                st.write("###  Average Execution Time (sec)")
                st.line_chart(df.set_index("function")["avg_time"])

                # Expanders for individual functions
                st.write("## Per-Function Metrics")
                for item in summary:
                    with st.expander(f"Function ID: {item['function_id']}"):
                        st.write(f"Total Calls: {item['total_calls']}")
                        st.write(f"Average Time: {item['avg_time']} sec")
                        st.write(f"Error Count: {item['error_count']}")

    except Exception as e:
        st.error(f"Failed to load dashboard: {e}")
```

Monitoring Dashboard:









Metrics are also auto-incrementally stored in mysql:

```
mysql> select * from metrics;
+----+-------------+---------------------+----------------+-----------+-----------------------------------------+
| id | function_id | timestamp           | execution_time | was_error | error_message                           |
+----+-------------+---------------------+----------------+-----------+-----------------------------------------+
|  1 |           1 | 2025-04-17 03:52:58 |       0.691605 |         0 | NULL                                    |
|  2 |           1 | 2025-04-17 10:01:53 |        0.83891 |         0 | NULL                                    |
|  3 |           1 | 2025-04-17 10:01:58 |         1.5456 |         0 | NULL                                    |
|  4 |           3 | 2025-04-17 10:02:10 |        -16.755 |         0 | NULL                                    |
|  5 |           5 | 2025-04-17 10:02:21 |        5.00378 |         1 | ERROR: Execution timed out after 5 seconds |
|  6 |           4 | 2025-04-17 10:02:33 |        1.77089 |         0 | NULL                                    |
|  7 |           4 | 2025-04-17 10:22:27 |       0.843738 |         0 | NULL                                    |
|  8 |           5 | 2025-04-17 10:23:06 |        4.99831 |         1 | ERROR: Execution timed out after 5 seconds |
|  9 |           1 | 2025-04-17 10:23:23 |       -15.9551 |         0 | NULL                                    |
| 10 |           2 | 2025-04-18 14:15:19 |       0.869897 |         0 | NULL                                    |
+----+-------------+---------------------+----------------+-----------+-----------------------------------------+
10 rows in set (0.00 sec)

mysql>
```

- **Task 3:** Integration and Polishing
    - Integrate all components (frontend, backend, execution engine)
    - Implement authentication/authorization (if time permits)
    - Conduct end-to-end testing

- Fix bugs and optimize performance
- Create documentation for the system

Deliverable: A complete system.

Adding special token to view metrics:

 # adding to backend/main.py:

```
from fastapi import FastAPI, Depends, HTTPException, Request, Header
from fastapi.responses import JSONResponse
from sqlalchemy.orm import Session
from sqlalchemy import func
#from fastapi.responses import JSONResponse
from . import models, schemas, crud, database
from .schemas import ExecuteFunctionRequest
import time
from .crud import log_metric
#from fastapi import Header

import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
"../../executor")))
import runner  # ☑ this imports runner.py from the executor/ folder

models.Base.metadata.create_all(bind=database.engine)


API_TOKEN = "secret123"  #special token
def verify_token(authorization: str = Header(...)):
    if not authorization.startswith("Bearer "):
        raise HTTPException(status_code=403, detail="Invalid token format")

    token = authorization.split(" ")[1]
    if token != API_TOKEN:
        raise HTTPException(status_code=403, detail="Unauthorized")


app = FastAPI()

def get_db():
    db = database.SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

```python
@app.post("/functions", response_model=schemas.FunctionOut)
def create(function: schemas.FunctionCreate, db: Session = Depends(get_db)):
    try:
        return crud.create_function(db, function)
    except Exception as e:
        print(f"Error occurred: {str(e)}")  # Log the error for debugging
        raise HTTPException(status_code=500, detail="Internal Server Error")


@app.get("/functions", response_model=list[schemas.FunctionOut])
def read_all(db: Session = Depends(get_db)):
    return crud.get_functions(db)

@app.get("/functions/{function_id}", response_model=schemas.FunctionOut)
def read_one(function_id: int, db: Session = Depends(get_db)):
    func = crud.get_function(db, function_id)
    if not func:
        raise HTTPException(status_code=404, detail="Function not found")
    return func

@app.delete("/functions/{function_id}")
def delete(function_id: int, db: Session = Depends(get_db)):
    success = crud.delete_function(db, function_id)
    if not success:
        raise HTTPException(status_code=404, detail="Function not found")
    return {"deleted": True}

@app.post("/functions/execute")
def execute_function(request: ExecuteFunctionRequest, db: Session =
Depends(get_db)):
    function_id = request.id
    args = request.args or []
    use_gvisor = request.use_gvisor

    func = db.query(models.Function).filter(models.Function.id ==
function_id).first()
    if not func:
        raise HTTPException(status_code=404, detail="Function not found")

    start = time.time()
    output, error = runner.run_function(
        func.language, func.route,
        args=args,
        timeout=func.timeout,
        use_gvisor=use_gvisor
    )
    duration = time.time() - start
    was_error = bool(error)
```

```
    log_metric(db, function_id=func.id, time_taken=duration,
was_error=was_error, error_message=error if was_error else None)

    return {"output": output, "error": error}

#endpoint to view metrics
@app.get("/metrics", response_model=list[schemas.MetricOut])
def get_all_metrics(
    db: Session = Depends(get_db),
    auth: str = Depends(verify_token)
):
    return db.query(models.ExecutionMetric).all()

#Metrics aggregation
@app.get("/metrics/summary")
def get_metric_summary(
    db: Session = Depends(get_db),
    auth: str = Depends(verify_token)
):
    from sqlalchemy import func
    summary = db.query(
        models.ExecutionMetric.function_id,
        func.count().label("total_calls"),
        func.avg(models.ExecutionMetric.execution_time).label("avg_time"),
        func.sum(func.if_(models.ExecutionMetric.was_error == True, 1,
0)).label("error_count")
    ).group_by(models.ExecutionMetric.function_id).all()

    return [
        {
            "function_id": s.function_id,
            "total_calls": s.total_calls,
            "avg_time": round(s.avg_time, 3),
            "error_count": int(s.error_count)
        } for s in summary
    ]


#update function
@app.put("/functions/{function_id}", response_model=schemas.FunctionOut)
def update_function(function_id: int, function: schemas.FunctionCreate, db:
Session = Depends(get_db)):
    func = crud.get_function(db, function_id)
    if not func:
        raise HTTPException(status_code=404, detail="Function not found")

    # Update fields
```

```
    func.name = function.name
    func.language = function.language
    func.route = function.route
    func.timeout = function.timeout

    db.commit()
    db.refresh(func)
    return func
```

adding to frontend\app.py:

```
import streamlit as st
import requests
import pandas as pd

API_URL = "http://127.0.0.1:8000"
AUTH_TOKEN = "secret123"  # This must match FastAPI's internal token
HEADERS = {"Authorization": f"Bearer {AUTH_TOKEN}"}

# Initialize session state
if "page" not in st.session_state:
    st.session_state.page = "Create Function"

# Initialize auth flag
if "dashboard_authenticated" not in st.session_state:
    st.session_state.dashboard_authenticated = False

# Menu bar with buttons
st.markdown("""
    <style>
        .menu {
            display: flex;
            gap: 20px;
            padding: 10px 0;
            border-bottom: 1px solid #ccc;
            margin-bottom: 20px;
        }
        .menu-button {
            background-color: #f0f2f6;
            border: none;
            color: #333;
            padding: 10px 20px;
            cursor: pointer;
            font-weight: bold;
        }
        .menu-button:hover {
```

```
            background-color: #e0e2e6;
        }
    </style>
""", unsafe_allow_html=True)

col1, col2, col3, col4, col5, col6 = st.columns(6)

with col1:
    if st.button("Create Function"):
        st.session_state.page = "Create Function"
with col2:
    if st.button("List Functions"):
        st.session_state.page = "List Functions"
with col3:
    if st.button("Delete Function"):
        st.session_state.page = "Delete Function"
with col4:
    if st.button("Update Function"):
        st.session_state.page = "Update Function"
with col5:
    if st.button("Execute Function"):
        st.session_state.page = "Execute Function"
with col6:
    if st.button("Monitoring Dashboard"):
        st.session_state.page = "Monitoring Dashboard"

# Main title
st.title("Lambda Serverless Function Manager")

# Create Function Page
if st.session_state.page == "Create Function":
    st.subheader("Deploy New Function")

    name = st.text_input("Function Name")
    language = st.selectbox("Language", ["python", "javascript"])
    route = st.text_input("Filename (e.g., hello.py)")
    timeout = st.number_input("Timeout (seconds)", min_value=1, max_value=30,
value=5)

    if st.button("Deploy"):
        payload = {
            "name": name,
            "language": language,
            "route": route,
            "timeout": timeout
        }
        response = requests.post(f"{API_URL}/functions", json=payload)
```

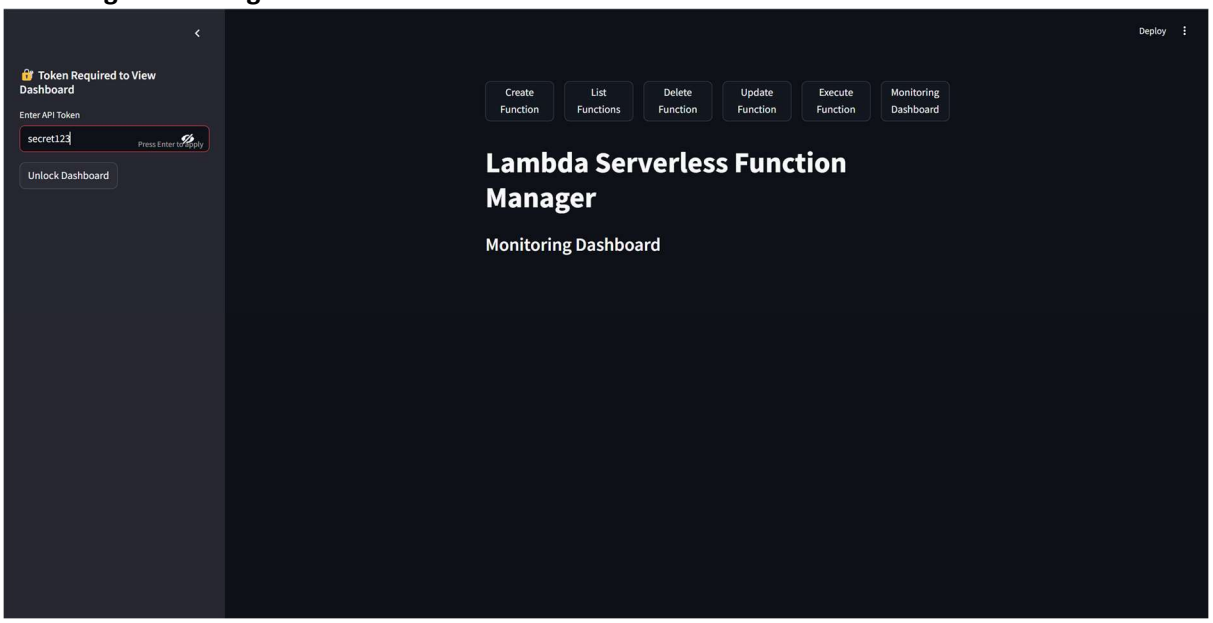
```
        if response.status_code == 200:
            st.success("Function deployed successfully!")
        else:
            st.error(f"Failed: {response.text}")

# List Functions Page
elif st.session_state.page == "List Functions":
    st.subheader("All Functions")
    response = requests.get(f"{API_URL}/functions")
    if response.ok:
        for func in response.json():
            st.json(func)
    else:
        st.error("Could not fetch functions.")

# Delete Function Page
elif st.session_state.page == "Delete Function":
    st.subheader("Delete Function")
    response = requests.get(f"{API_URL}/functions")
    if response.ok:
        functions = response.json()
        choices = {f"{f['name']} (ID: {f['id']})": f['id'] for f in functions}
        selected = st.selectbox("Select Function to Delete",
list(choices.keys()))
        if st.button("Delete"):
            fid = choices[selected]
            del_res = requests.delete(f"{API_URL}/functions/{fid}")
            if del_res.status_code == 200:
                st.success("Deleted successfully!")
            else:
                st.error("Delete failed.")
    else:
        st.error("Could not load functions.")

# Update Function Page
elif st.session_state.page == "Update Function":
    st.subheader("✏️ Update Existing Function")
    response = requests.get(f"{API_URL}/functions")
    if response.ok:
        functions = response.json()
        choices = {f"{f['name']} (ID: {f['id']})": f for f in functions}
        selected = st.selectbox("Select Function to Update",
list(choices.keys()))

        func_data = choices[selected]
        updated_name = st.text_input("New Name", value=func_data["name"])
```

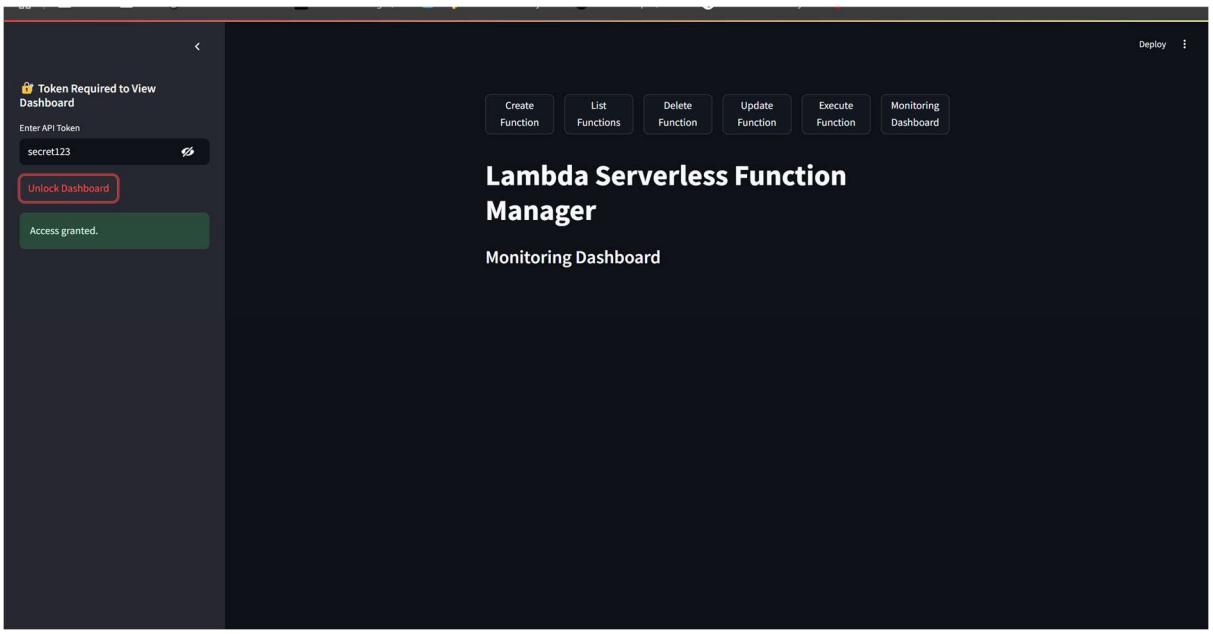
```
        updated_lang = st.selectbox("Language", ["python", "javascript"],
index=["python", "javascript"].index(func_data["language"]))
        updated_route = st.text_input("Filename", value=func_data["route"])
        updated_timeout = st.number_input("Timeout", min_value=1,
max_value=30, value=func_data["timeout"])

        if st.button("Update"):
            payload = {
                "name": updated_name,
                "language": updated_lang,
                "route": updated_route,
                "timeout": updated_timeout
            }
            res = requests.put(f"{API_URL}/functions/{func_data['id']}",
json=payload)
            if res.status_code == 200:
                st.success("Function updated successfully!")
            else:
                st.error(f"Update failed: {res.text}")
    else:
        st.error("Could not fetch functions.")

# Execute Function Page
elif st.session_state.page == "Execute Function":
    st.subheader("⚡ Execute Function")

    function_id = st.number_input("Function ID", min_value=1, step=1)
    raw_args = st.text_input("Arguments (comma-separated)", value="")
    use_gvisor = st.checkbox("Use gVisor Runtime", value=False)

    if st.button("Execute"):
        args = [arg.strip() for arg in raw_args.split(",") if arg.strip()]
        payload = {
            "id": int(function_id),
            "args": args,
            "use_gvisor": use_gvisor
        }
        response = requests.post(f"{API_URL}/functions/execute", json=payload)
        if response.status_code == 200:
            data = response.json()
            st.success("☑ Function Executed!")
            st.code(data.get("output", ""), language="text")
            if data.get("error"):
                st.error(data["error"])
        else:
            st.error("Function execution failed.")
```

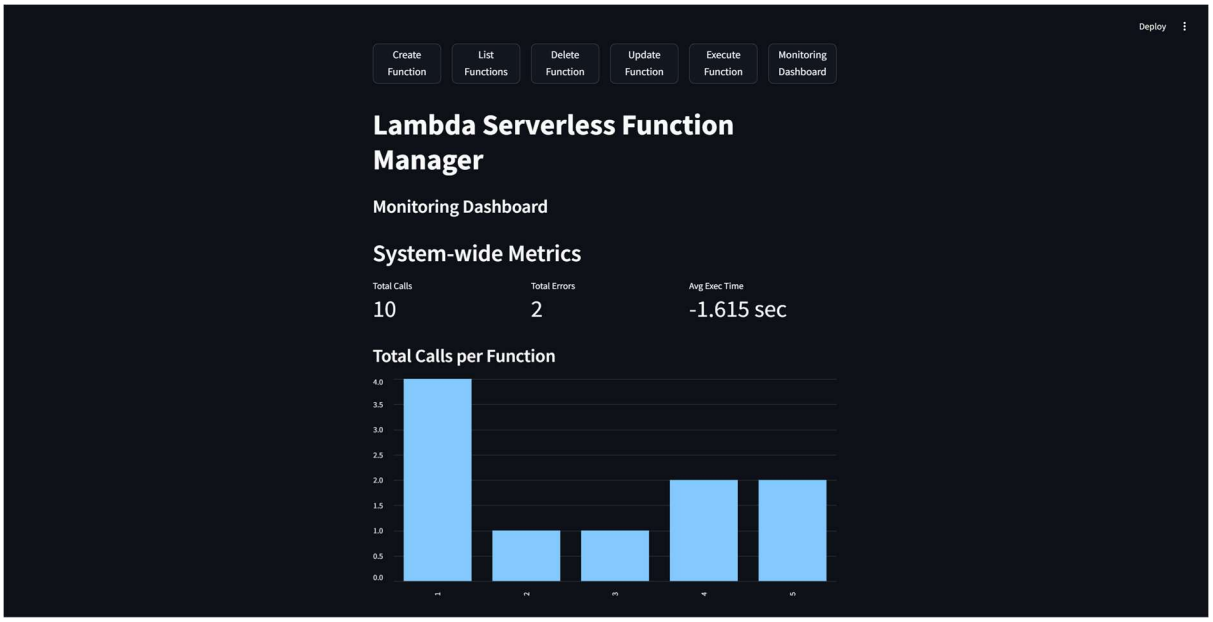
```
# Monitoring Dashboard (with token gate)
elif st.session_state.page == "Monitoring Dashboard":
    st.subheader(" Monitoring Dashboard")

    # Prompt for token if not yet authenticated
    if not st.session_state.dashboard_authenticated:
        with st.sidebar:
            st.markdown("### 🔐 Token Required to View Dashboard")
            token_input = st.text_input("Enter API Token", type="password")
            if st.button("Unlock Dashboard"):
                if token_input == AUTH_TOKEN:
                    st.session_state.dashboard_authenticated = True
                    st.success("Access granted.")
                else:
                    st.error("Invalid token.")
        st.stop()

    try:
        response = requests.get(f"{API_URL}/metrics/summary", headers=HEADERS)
        if response.status_code != 200:
            st.warning("Could not load metrics.")
        else:
            summary = response.json()
            if not summary:
                st.info("No metrics available yet.")
            else:
                df = pd.DataFrame(summary)
                df["function"] = df["function_id"].astype(str)

                st.write("## System-wide Metrics")
                total_calls = df["total_calls"].sum()
                total_errors = df["error_count"].sum()
                avg_exec_time = round((df["avg_time"] *
df["total_calls"]).sum() / total_calls, 3) if total_calls else 0

                col1, col2, col3 = st.columns(3)
                col1.metric("Total Calls", total_calls)
                col2.metric("Total Errors", total_errors)
                col3.metric("Avg Exec Time", f"{avg_exec_time} sec")

                st.write("###  Total Calls per Function")
                st.bar_chart(df.set_index("function")["total_calls"])

                st.write("###  Error Count per Function")
                st.bar_chart(df.set_index("function")["error_count"])

                st.write("### Average Execution Time (sec)")
```

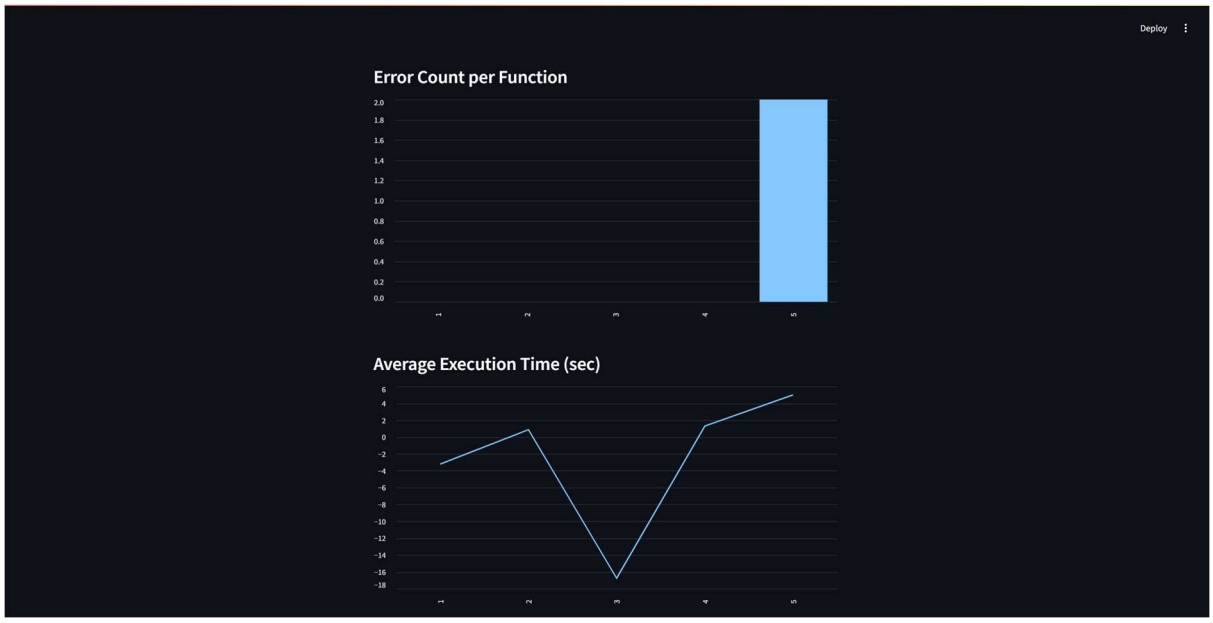
```
            st.line_chart(df.set_index("function")["avg_time"])

            st.write("## Per-Function Metrics")
            for item in summary:
                with st.expander(f"Function ID: {item['function_id']}"):
                    st.write(f"Total Calls: {item['total_calls']}")
                    st.write(f"Average Time: {item['avg_time']} sec")
                    st.write(f"Error Count: {item['error_count']}")

    except Exception as e:
        st.error(f"Failed to load dashboard: {e}")
```

accessing monitoring dashboard:









NAME	SRN
Naru Meghana	PES2UG22CS341
Neha Girish	PES2UG22CS346
Nida Fathima	PES2UG22CS353
Nikhita Gejjalli	PES2UG22CS359