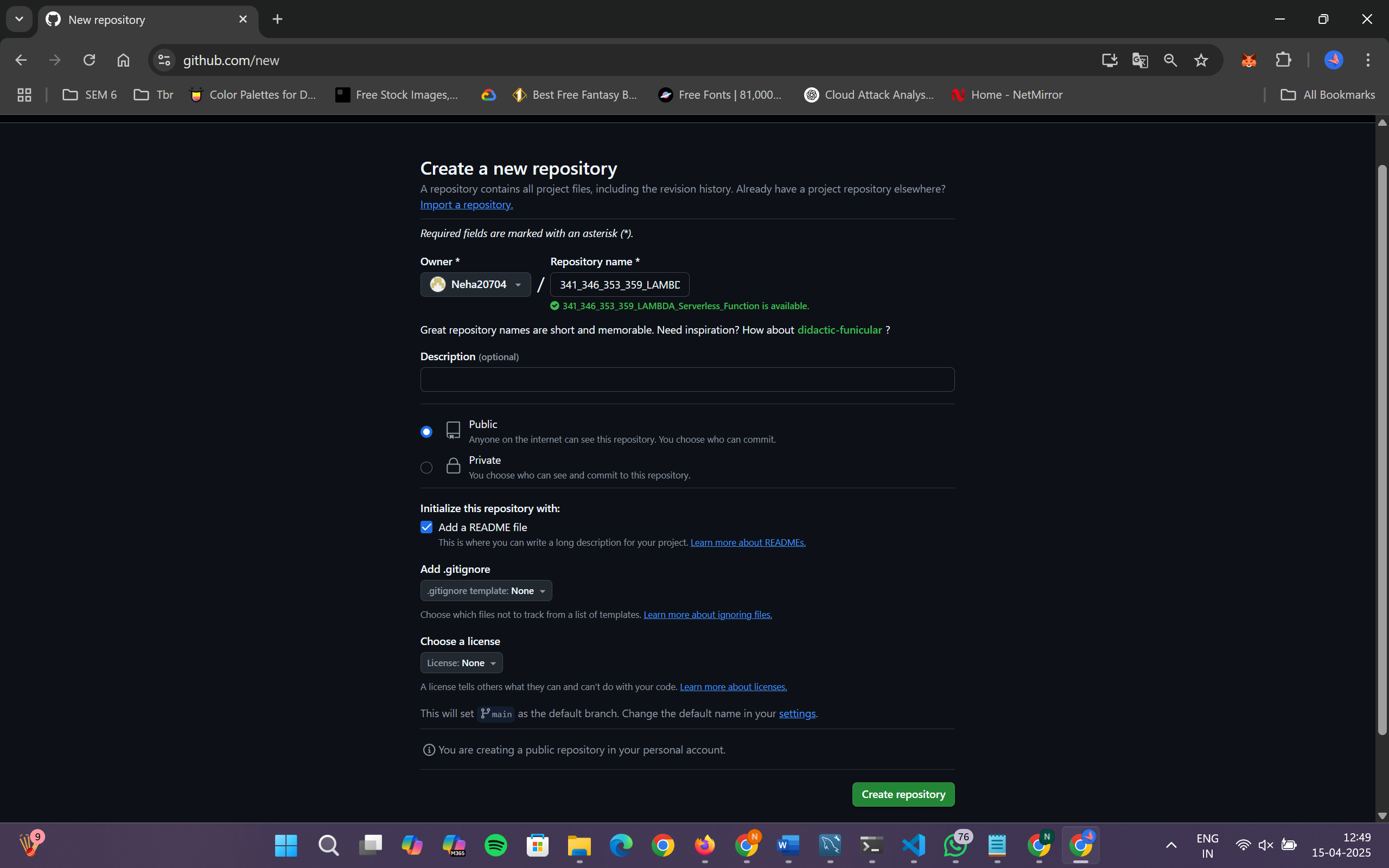
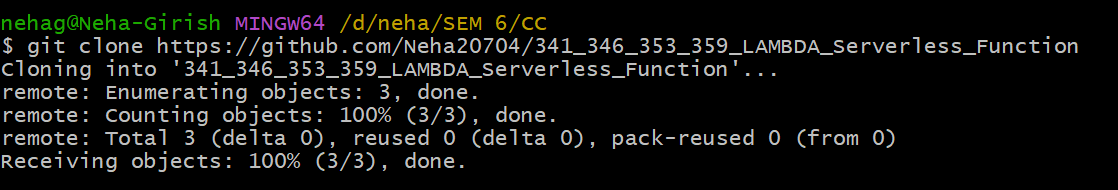
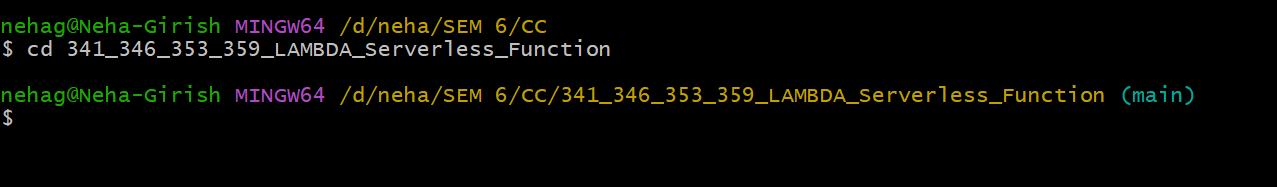
Creating a git repo and initializing it with a ReadME.md

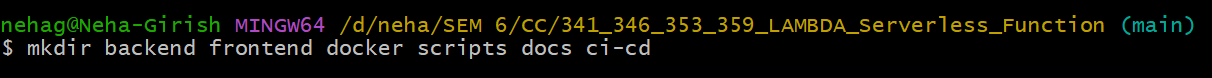


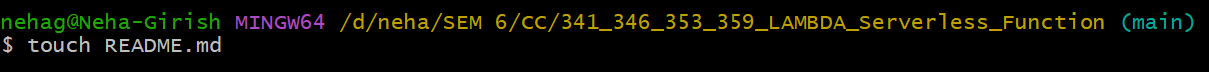
0.2: Clone and Initialize Project Structure

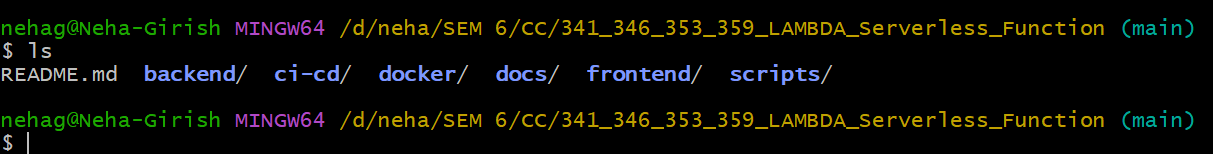
Run the following commands:



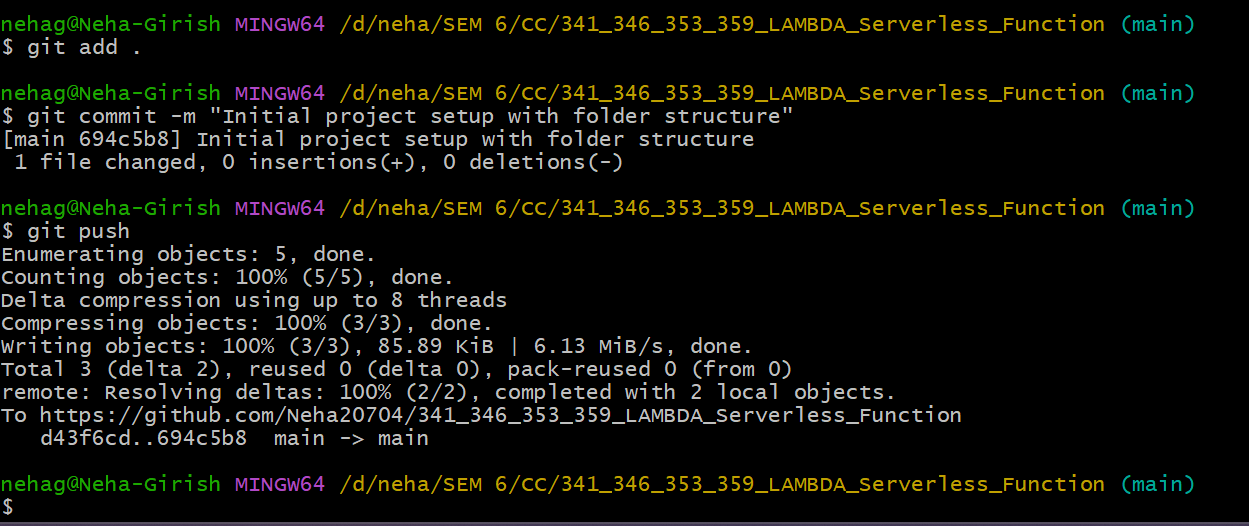




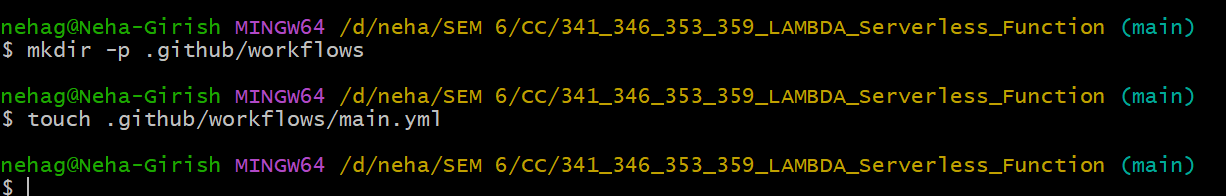


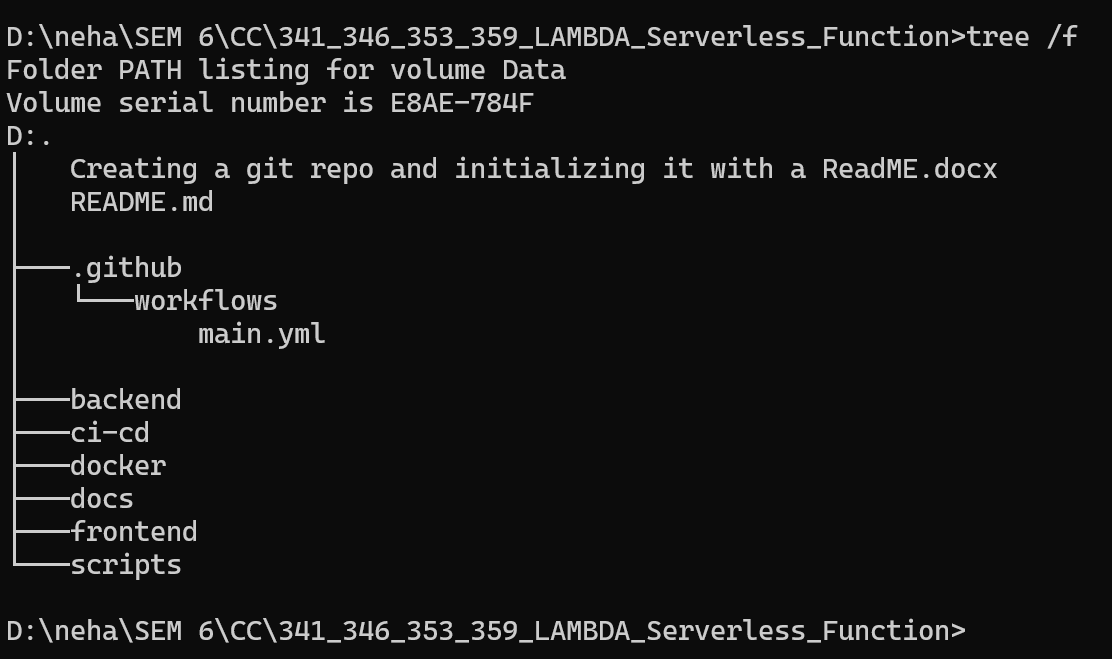


0.3: Initialize Git and Make First Commit

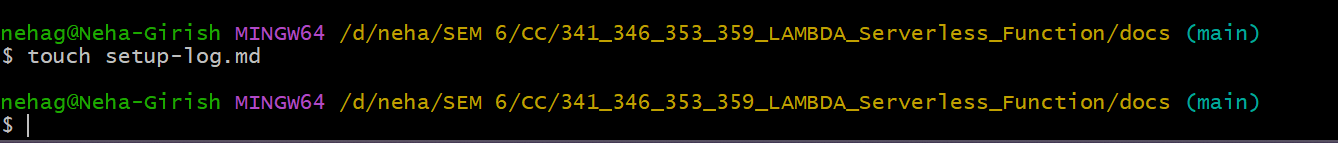


0.4: Set Up CI/CD (Optional for Now)



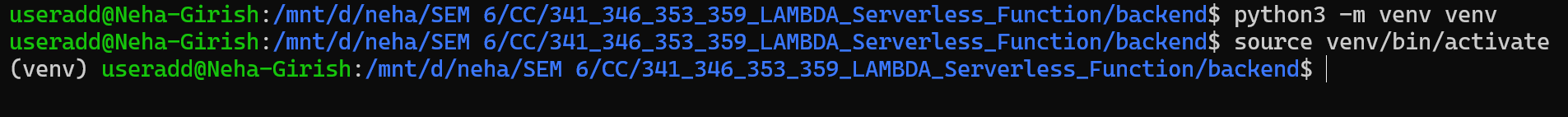


0.5: Document in /docs/setup-log.md

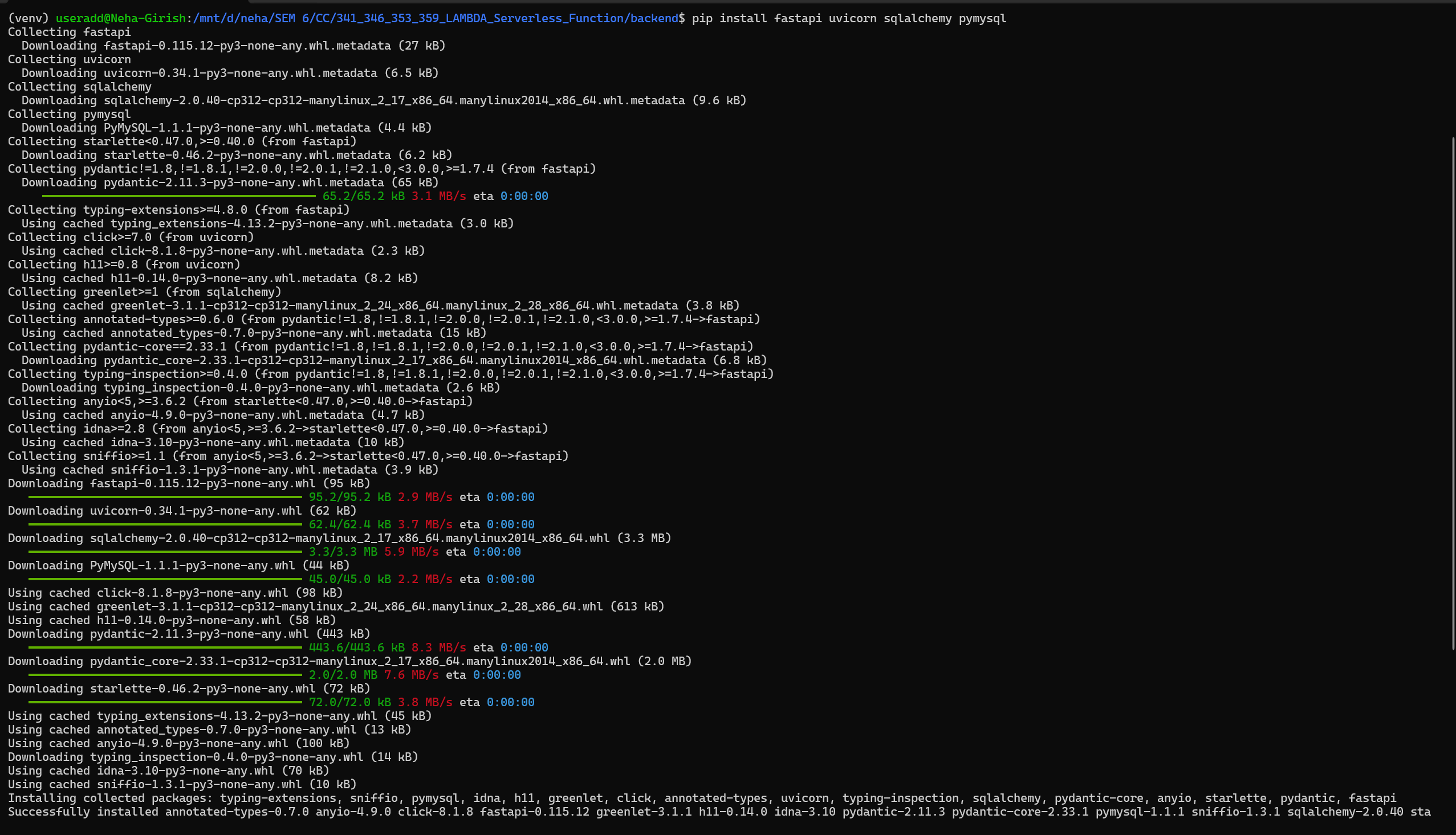


1.1: Set Up FastAPI Environment

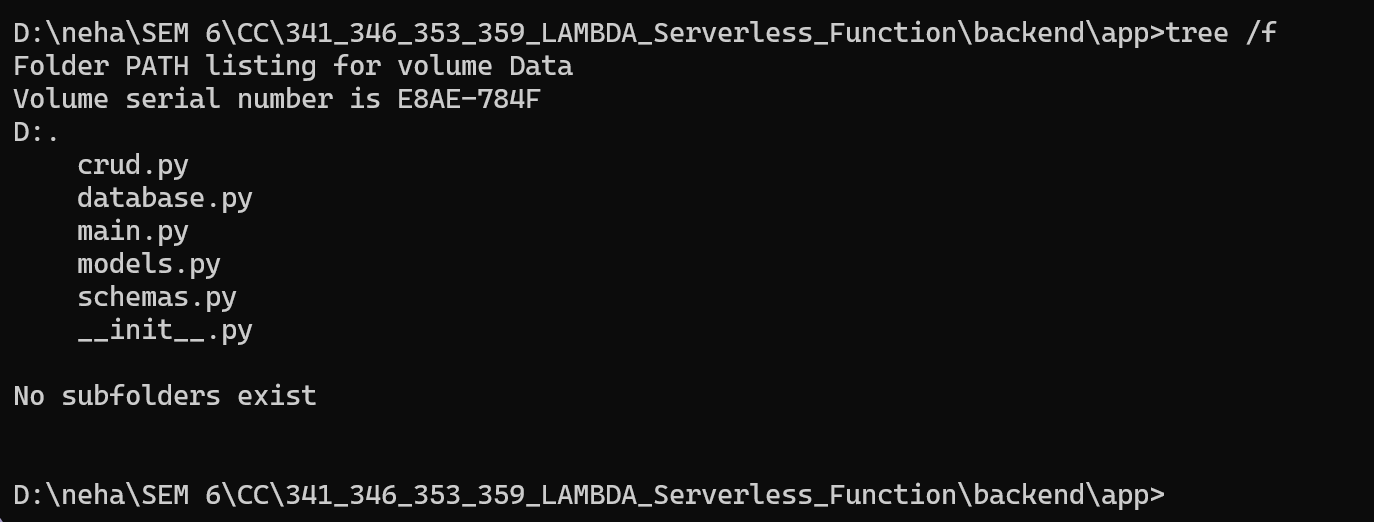
Create a Python environment:

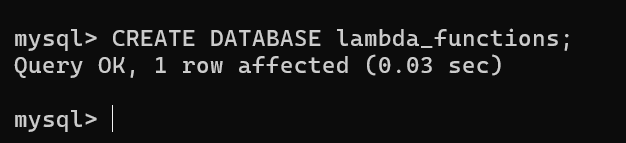


Install dependencies:



1.2: Backend Folder Structure





app/database.py

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

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

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

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

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}")

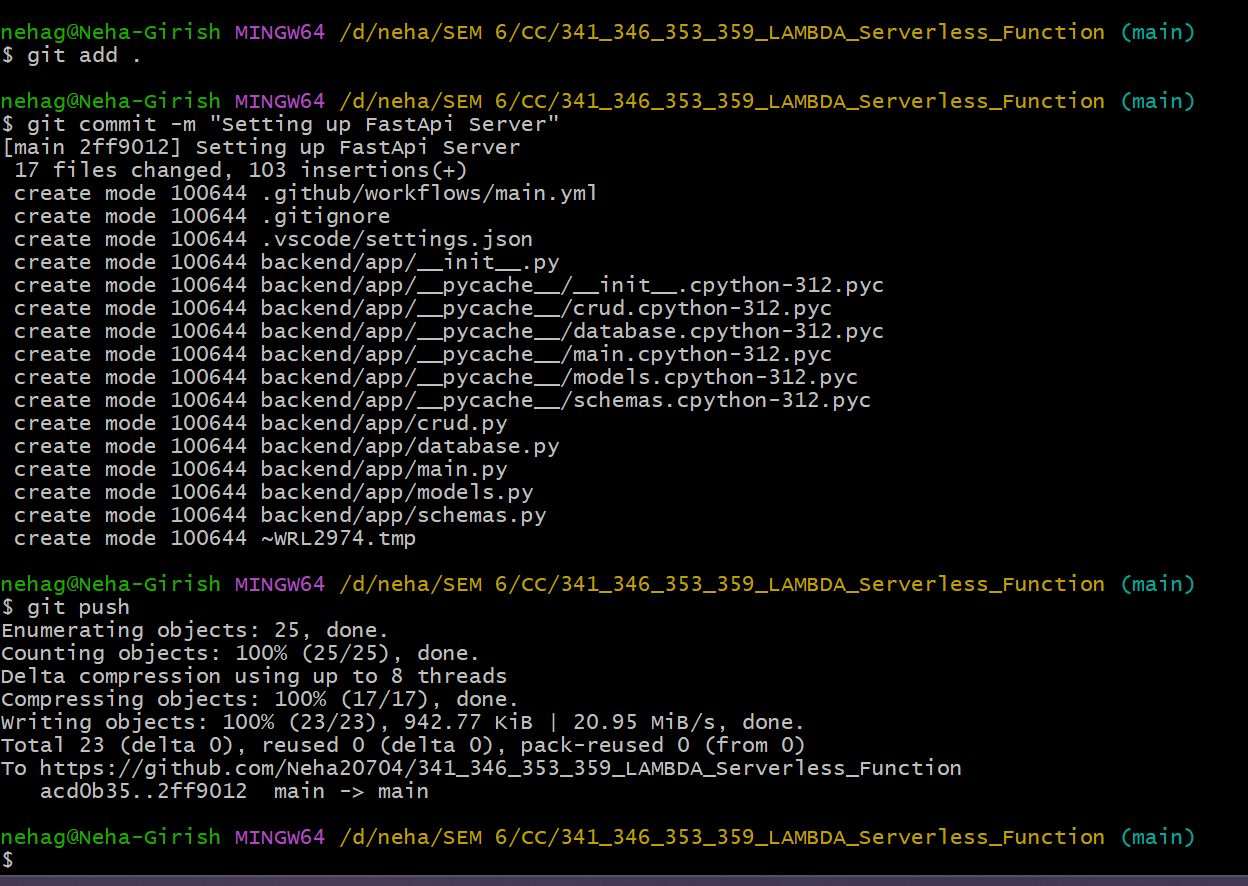
def delete(function\_id: int, db: Session = Depends(get\_db)):

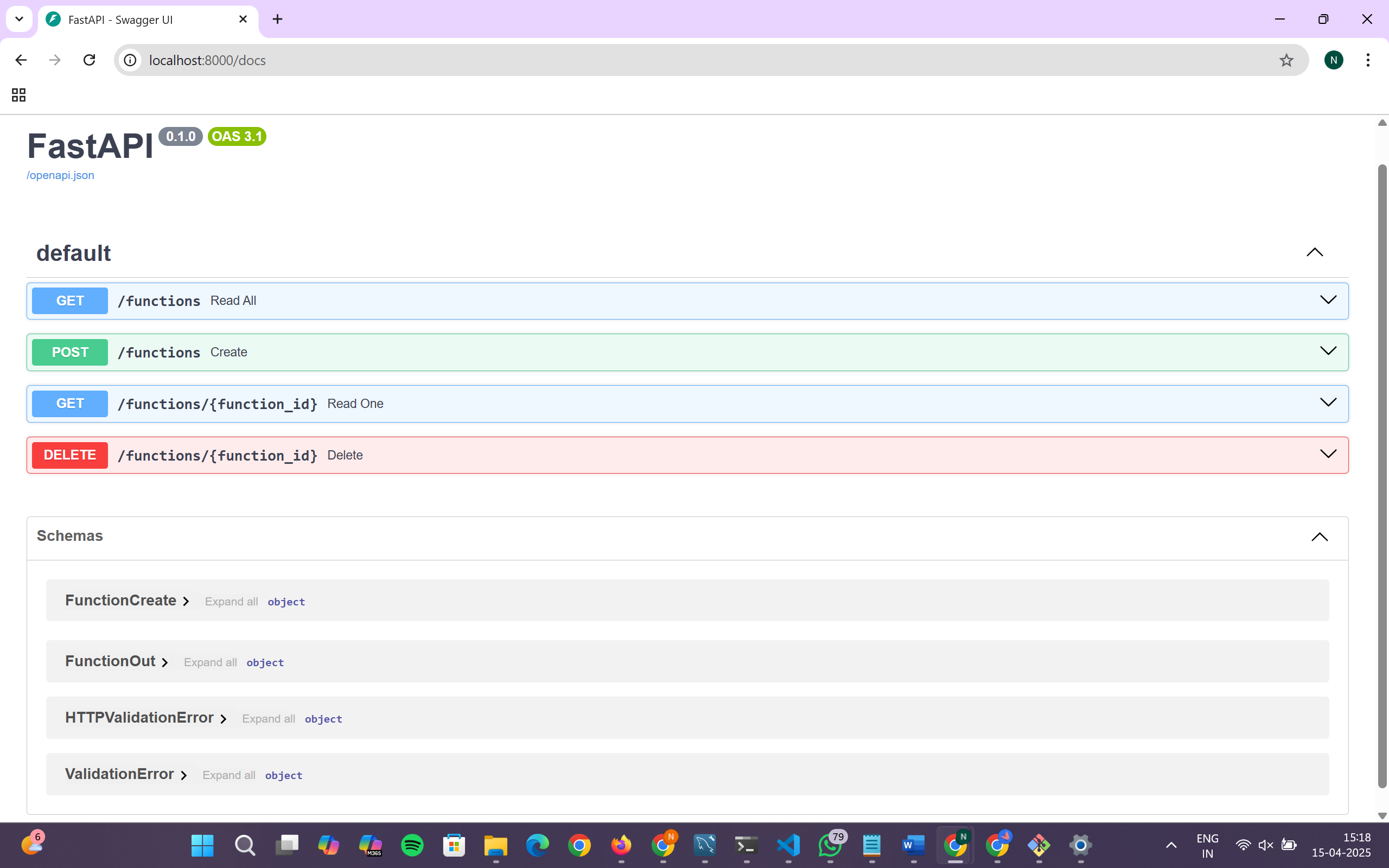
    success = crud.delete\_function(db, function\_id)

    if not success:

        raise HTTPException(status\_code=404, detail="Function not found")

    return {"deleted": True}





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

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

const args = process.argv.slice(2);

const inputName = args[0] || "World";

console.log(`Hello, ${inputName}!`);

docker/functions/javascript/arithmetic.js

const a = parseFloat(process.argv[2]);

const b = parseFloat(process.argv[3]);

if (isNaN(a) || isNaN(b)) {

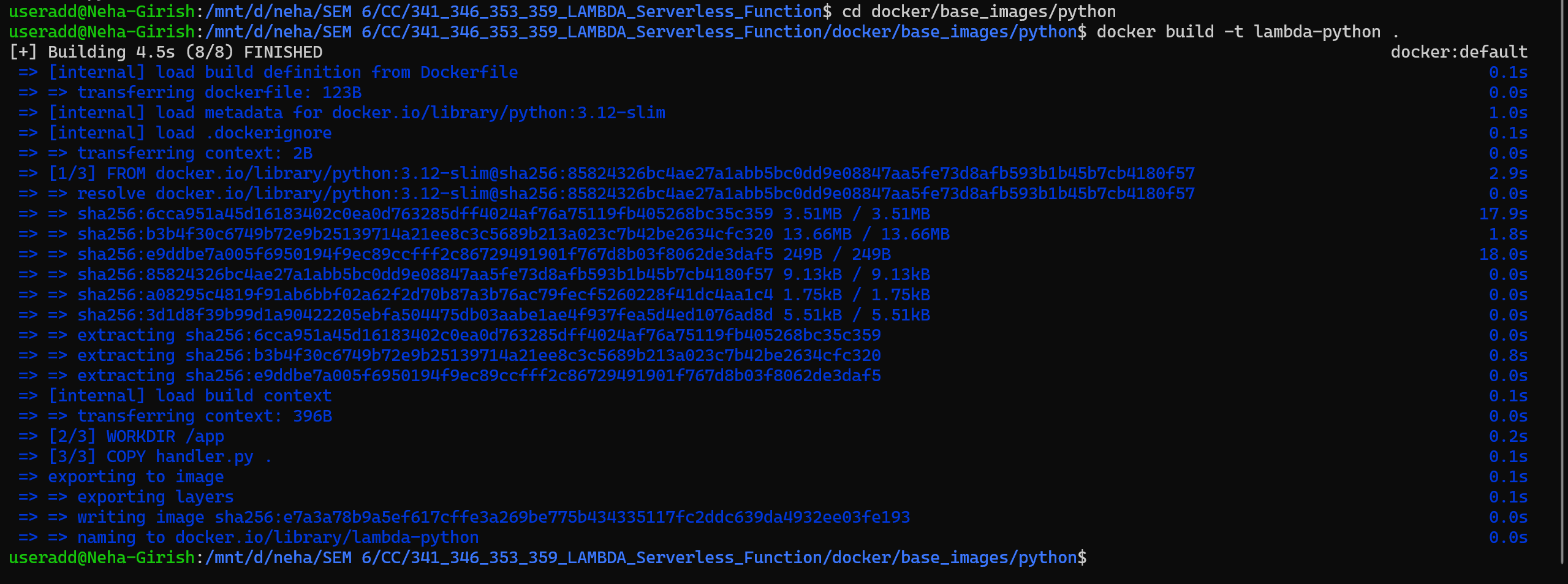
    console.log("Usage: node arithmetic.js <num1> <num2>");

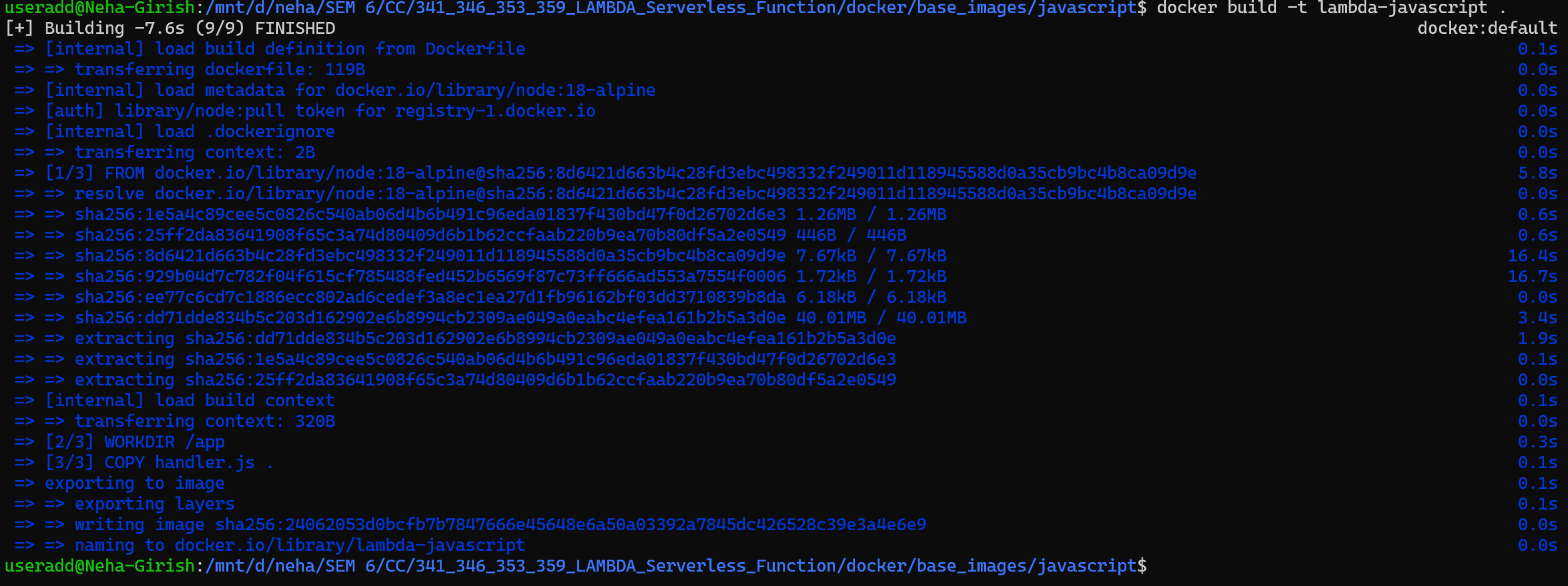
} else {

    console.log(`Sum: ${a + b}`);

}

Build Base Images





**Commands:  
cd docker/base\_images/python**

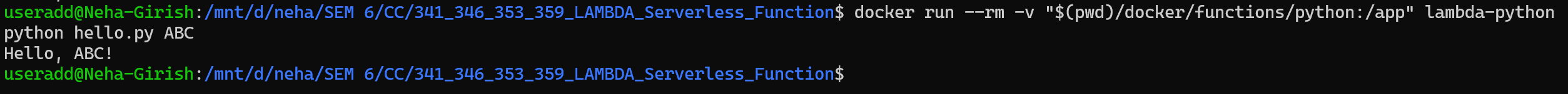
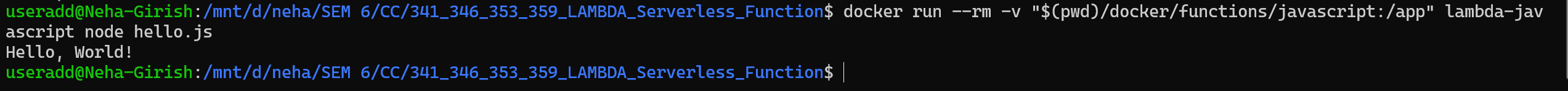
**docker build -t lambda-python .**

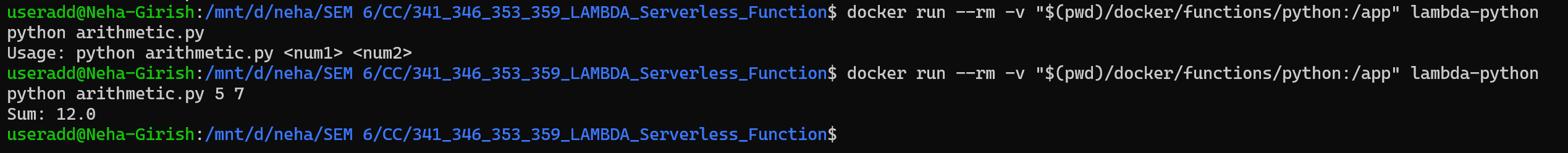
**cd ../javascript**

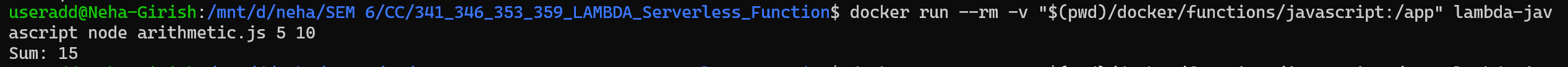
**docker build -t lambda-javascript .**

Step 3: Run Python & JavaScript Functions in Containers

Hello:  
python:

  
javascript:  
  
arithmetic:

Python:  


Javascript:  


**Commands:**

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

**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**

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

Implementing timeout:  
Rebuilding base images:  
