**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC MỞ THÀNH PHỐ HỒ CHÍ MINH**

**PHẠM QUANG MINH**

**PHÁT TRIỂN CÔNG CỤ ONLINE COMPILER**

**ĐỒ ÁN NGÀNH**

**NGÀNH KHOA HỌC MÁY TÍNH**

**TP. HỒ CHÍ MINH, 2021BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC MỞ THÀNH PHỐ HỒ CHÍ MINH**

**PHẠM QUANG MINH**

**PHÁT TRIỂN CÔNG CỤ ONLINE COMPILER**

**Mã số sinh viên: 1851010075**

**ĐỒ ÁN NGÀNH**

**NGÀNH KHOA HỌC MÁY TÍNH**

**Giảng viên hướng dẫn: LÊ NGỌC HIẾU**

**TP. HỒ CHÍ MINH, 2021**

**Ý KIẾN CHO PHÉP BẢO VỆ ĐỒ ÁN/ KHÓA LUẬN TỐT NGHIỆP**

**CỦA GIẢNG VIÊN HƯỚNG DẪN**

**Giảng viên hướng dẫn:**

**Sinh viên thực hiện: Lớp:**

**Ngày sinh: Nơi sinh:**

**Tên đề tài:**

**Ý kiến của giảng viên hướng dẫn về việc cho phép sinh viên được bảo vệ đồ án/ khóa luận trước Hội đồng:**

*Thành phố Hồ Chí Minh, ngày … tháng … năm ……*

Người nhận xét

…………………………

LỜI CẢM ƠN

Lời đầu tiên xin được cảm ơn các giảng viên tại trường đại học Mở thành phố Hồ Chí Minh đã cùng đồng hành với sinh viên chúng em trong suốt thời gian học tại trường. Cảm ơn các anh chị cùng các bạn tại khoa Công nghệ thông tin đã cùng gắn bó, chia sẻ, dìu dắt để em có thể lựa chọn và tìm ra hướng đi cho riêng mình trong tương lai.

Đặc biệt cảm ơn giảng viên hướng dẫn của em trong đồ án ngành lần này là thầy Lê Ngọc Hiếu đã cùng đồng hành và giúp đỡ em hoàn thành đồ án của mình. Dù gặp nhiều khó khăn khi thực hiện nhưng thầy đã luôn giúp đỡ động viên để có thể hoàn thành đồ án khiến em vô cùng cảm kích.

Đồ án của em lần này còn nhiều thiếu sót nhưng thực sự là những kiến thức em đã tìm hiểu và học hỏi được qua sự giúp đỡ của mọi người. Mong rằng các thầy cô nếu được có thể góp ý giúp em hoàn thiện hơn phục vụ cho công việc sau này.

Một lần nữa xin chân thành cảm ơn tất cả!

*Thành phố Hồ Chí Minh, ngày … tháng … năm ……*

Phạm Quang Minh

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

TÓM TẮT ĐỒ ÁN

Trong đồ án này ta sẽ thực hiện công cụ để chạy được các mã nguồn của các ngôn ngữ lập trình như python, javascript, java từ website. Công cụ này sẽ vận hành bằng cách gửi một yêu cầu lên server có nội dung bao gồm mã nguồn và ngôn ngữ thực hiện sau đó server sẽ trả về kết quả chạy đoạn mã nguồn đó .Để thực hiện được điều này đồ án vận dụng các kiến thức về python django để tạo API, Docker để dùng làm môi trường chạy code từ website, React để tạo giao diện và sử dụng các câu lệnh command line của linux để thực thi code trong môi trường Docker. Kết quả thực hiện là đã chạy được mã nguồn từ website trong Docker container nhưng vẫn còn chậm và phụ thuộc rất nhiều vào tốc độ mạng, từ công cụ này có thể phát triển thêm để trở thành một website học lập trình hiệu quả hoặc một website luyện code, luyện thuật toán mà không cần cài đặt môi trường trên máy tính cá nhân.

**MỤC LỤC**

[LỜI CẢM ƠN 2](#_Toc87798767)

[NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN 3](#_Toc87798768)

[TÓM TẮT ĐỒ ÁN 4](#_Toc87798769)

[DANH MỤC TỪ VIẾT TẮT 7](#_Toc87798770)

[DANH MỤC HÌNH VẼ 8](#_Toc87798771)

[DANH MỤC BẢNG 9](#_Toc87798772)

[Chương 1. GIỚI THIỆU ĐỀ TÀI 10](#_Toc87798773)

[1.1. Tổng quan 10](#_Toc87798774)

[1.2. Mục đích 10](#_Toc87798775)

[1.3. Phạm vi và đối tượng nghiên cứu 10](#_Toc87798776)

[Chương 2. NỀN TẢNG LÝ THUYẾT 11](#_Toc87798777)

[2.1. Giới thiệu Docker 11](#_Toc87798778)

[2.1.1. Tổng quan về Docker 11](#_Toc87798779)

[2.1.2. Cơ chế hoạt động 11](#_Toc87798780)

[2.1.3. Kiến trúc của Docker 11](#_Toc87798781)

[2.1.4. Các thành phần chính trong Docker 12](#_Toc87798782)

[2.1.5. Cài đặt và sử dụng Docker 12](#_Toc87798783)

[2.2. Giới thiệu Django và Django REST Framework 14](#_Toc87798784)

[2.2.1. Django là gì? 14](#_Toc87798785)

[2.2.2. Mô hình MVT 14](#_Toc87798786)

[2.2.3. Tổng quan về Django REST Framework 15](#_Toc87798787)

[2.2.4. Tạo project 15](#_Toc87798788)

[2.2.5. Tạo ứng dụng trong Django 17](#_Toc87798789)

[2.3. Giới thiệu React 22](#_Toc87798790)

[2.3.1. Tổng quan về React 22](#_Toc87798791)

[2.3.2. DOM ảo trong React 22](#_Toc87798792)

[2.3.3. Components và props trong React 23](#_Toc87798793)

[2.3.4. State trong React 24](#_Toc87798794)

[2.3.5. Vòng đời của React component 24](#_Toc87798795)

[2.3.6. React Hook 26](#_Toc87798796)

[Chương 3. CÔNG CỤ ONLINE COMPILER 31](#_Toc87798797)

[3.1. Kiến trúc hệ thống 31](#_Toc87798798)

[3.2. Thiết kế API 32](#_Toc87798799)

[3.3. Hiện thực API 33](#_Toc87798800)

[3.4. Demo công cụ online compiler 35](#_Toc87798801)

[Chương 4. Chương 4. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 40](#_Toc87798802)

[4.1. Kết quả đạt được 40](#_Toc87798803)

[4.2. Chức năng cần cải thiện 40](#_Toc87798804)

[4.3. Hướng phát triển 40](#_Toc87798805)

[TÀI LIỆU THAM KHẢO 41](#_Toc87798806)

DANH MỤC TỪ VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| STT | Từ viết tắt | Giải nghĩa |
| 1 | REST | Representational state transfer |
| 2 | API | Application Programming Interface |
| 3 | MVC | Model – View – Controller |
| 4 | MVT | Model – View – Template |
| 5 | CLI | Command line interface |
| 6 | DOM | Document Object Model |
| 7 | t | Tag |
| 8 | rm | Remove |
| 9 | rmi | Remove image |
| 10 | npm | Node Package Manager |
| 11 | npx | Node Package Execute |

DANH MỤC HÌNH VẼ

[Hình 2.1: Kiến trúc của Docker (https://docs.docker.com/get-started/overview/) 12](#_Toc87798741)

[Hình 2.2: Tạo project Pycharm 16](#_Toc87798742)

[Hình 2.3: Giao diện tại localhost:8000 sau khi chạy server 17](#_Toc87798743)

[Hình 2.4: Giao diện khi vào localhost 21](#_Toc87798744)

[Hình 2.5: API mới tạo chỉ có phương thức POST 21](#_Toc87798745)

[Hình 2.6: Tương tác DOM ảo (https://viblo.asia/p/virtual-dom-trong-reactjs-YWOZrMdEKQ0) 22](#_Toc87798746)

[Hình 2.7: Vòng đời của một React component (https://viblo.asia/p/vong-doi-cua-mot-react-component-RQqKLMRzZ7z) 25](#_Toc87798747)

[Hình 3.1: Kiến trúc ban đầu 31](#_Toc87798748)

[Hình 3.2: Kiến trúc hệ thống sử dụng Docker 32](#_Toc87798749)

[Hình 3.3: Giao diện của công cụ 36](#_Toc87798750)

[Hình 3.4: Phân vùng trả về kết quả sau khi chạy mã 36](#_Toc87798751)

[Hình 3.5: Chọn ngôn ngữ lập trình 37](#_Toc87798752)

[Hình 3.6: Nhập mã nguồn 37](#_Toc87798753)

[Hình 3.7: Nhấn vào nút Run 38](#_Toc87798754)

[Hình 3.8: Kết quả được trả về 38](#_Toc87798755)

[Hình 3.9: Đoạn mã có input 39](#_Toc87798756)

[Hình 3.10: Kết quả mã nguồn có input 39](#_Toc87798757)

DANH MỤC BẢNG

[Bảng 2.1: Mô hình MVC và MVT 14](#_Toc86545172)

# GIỚI THIỆU ĐỀ TÀI

Trong chương này giúp ta nắm được khái quát về đề tài

## Tổng quan

Trình biên dịch trực tuyến thường được sử dụng để chạy các mã nguồn trực tuyến mà không cần phải cài đặt môi trường trên máy, đề tài dựa trên các kiến thức về docker, django và react để tạo ra một trình biên dịch trực tuyến từ đó có thể phát triển thêm để sử dụng vào ứng dụng web để học lập trình trực tuyến hoặc code challenge

## Mục đích

Đề tài sẽ tập trung hoàn thiện việc thực hiện trình biên dịch trực tuyến cho đa ngôn ngữ như java, python, javascript để giúp người sử dụng có thể chạy các dòng code đơn giản mà không cần cài đặt các môi trường trên máy cũng như từ công cụ này có thể phát triển lên một website học tập hoặc code challenge.

## Phạm vi và đối tượng nghiên cứu

Để tạo ra một công cụ biên dịch trực tuyến, đề tài tìm hiểu sâu về các khái niệm, cách thức hoạt động, tổ chức của Python Django Rest Framework, React, Docker.

# NỀN TẢNG LÝ THUYẾT

Chương này giúp ta làm rõ, nắm được các khái niệm và một số các câu lệnh cơ bản được sử dụng để xây dựng ứng dụng

## Giới thiệu Docker

### Tổng quan về Docker

Docker là một nền tảng mã nguồn mở được dùng để phát triển, triển khai, thực thi ứng dụng. Docker giúp tách biệt các ứng dụng khỏi môi trường máy tính phát triển để có thể phân phối phần mềm, ứng dụng một cách nhanh chóng.

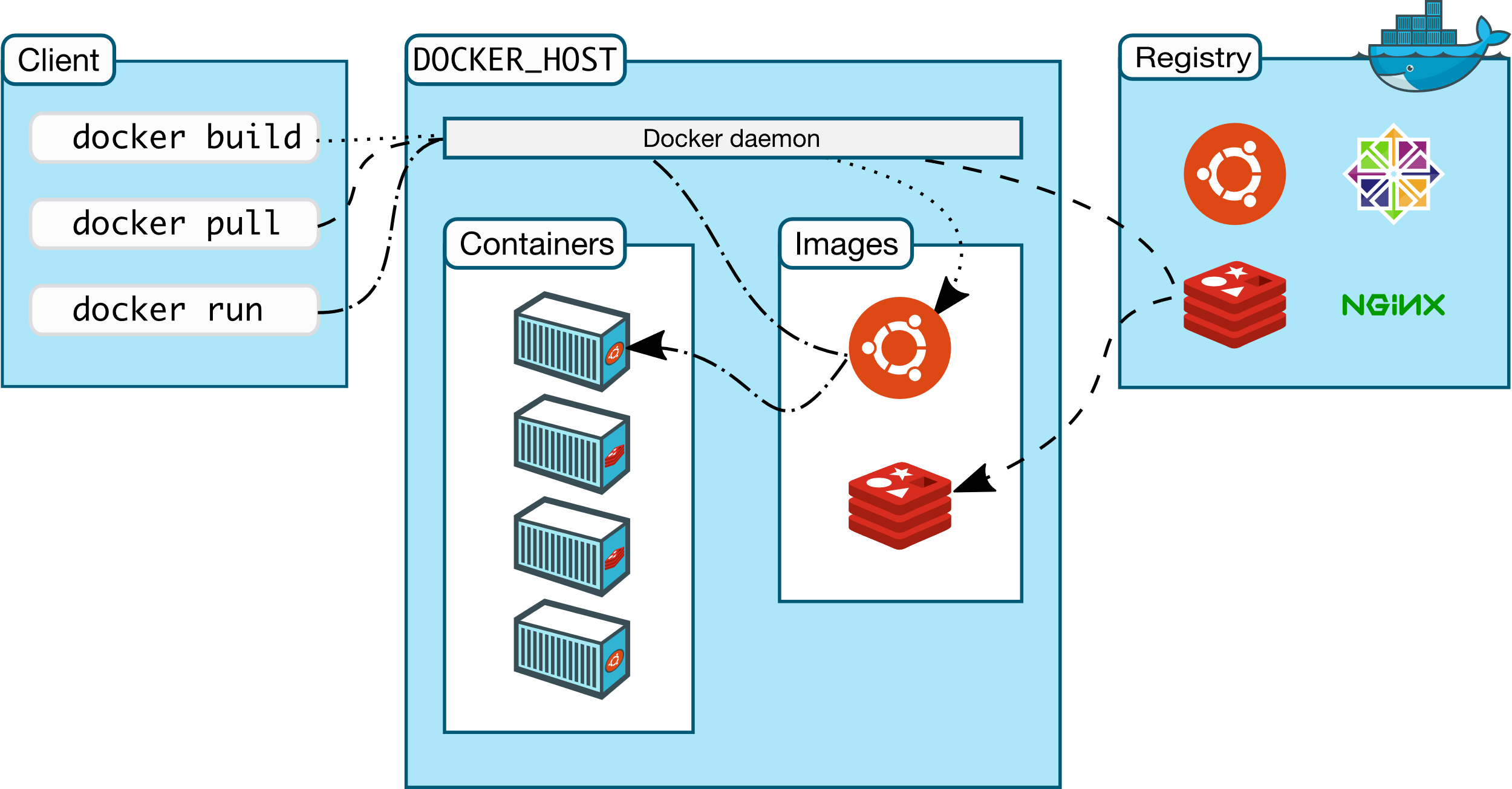
### Cơ chế hoạt động

Docker cung cấp khả năng đóng gói và chạy ứng dụng trong một môi trường bị cô lập được gọi là container. Sự cô lập và bảo mật cho phép ta có thể chạy nhiều conainer đồng thời trên cùng một máy chủ. Các container có dung lượng nhẹ và chứa mọi thứ cần thiết để chạy ứng dụng bên trong nó, chính vì thế ta không cần phải dựa vào những gì được cài đặt trên máy chủ để chạy phần mềm. Nhờ vậy nên ta có thể dễ dàng chia sẻ các container và chạy chúng ở các máy chủ khác mà không cần cài đặt các công cụ mà ứng dụng yêu cầu lên máy chủ.

Docker được tạo ra để sử dụng trên nền tảng Linux nhưng hiện nay đã được phát triển để chạy trên các nền tảng khác như Windows, MacOS.

### Kiến trúc của Docker

Docker sử dụng kiến trúc client – server. Docker client sẽ giao tiếp với Docker daemon là nơi thực hiện các công việc xây dựng, chạy và phân phối các Docker container. Docker client và daemon có thể chạy trên cùng một hệ thống hoặc kết nối từ xa. Hai bên giao tiếp với nhau bằng cách sử dụng REST API, Unix socket hoặc kết nối mạng. Một dạng Docker client khác là Docker Compose, cho phép ta làm việc với các ứng dụng trên một tập các container.



Hình 2.1: Kiến trúc của Docker (<https://docs.docker.com/get-started/overview/>)

### Các thành phần chính trong Docker

Khi sử dụng Docker, ta tạo ra và sử dụng các image, container, mạng, volume, plugin, và các thành phần khác. Ở đây ta cần nắm một số các thành phần:

* Image là các mẫu chỉ đọc với các hướng dẫn để tạo ra Docker container. Ta có thể tự tạo một image hoặc sử dụng image do người khác tạo và được đưa lên registry.
* Dockerfile là một file chứa các lệnh để xây dựng các Docker image.
* Container là nơi thực thi các thể hiện của một image. Ta có thể tạo, chạy, di chuyển hoặc xóa các container bằng Docker API hoặc CLI. Ta có thể kết nối container với một hoặc nhiều mạng, đính kèm bộ nhớ hoặc tạo một image dựa trên trạng thái hiện tại của container.

### Cài đặt và sử dụng Docker

Để sử dụng Docker và các dịch vụ của nó ta cần cài đặt phiên bản đầy đủ của Docker.

Đối với Windows hoặc MacOS ta có thể tải và cài đặt phiên bản mới nhất tại <https://docs.docker.com/desktop/>.

Còn đối với hệ điều hành linux ta có thể xem cụ thể cách cài đặt trên từng phiên bản hệ điều hành tại <https://docs.docker.com/engine/install/>.

Sau khi đã cài đặt được Docker ta có thể khởi chạy container đầu tiên trên Docker bằng cách mở command line và gõ lệnh sau

docker run hello-world

Để tạo Docker image ta cần viết Dockerfile sau đó chạy lệnh cli như sau

docker build [OPTIONS] [path to Dockerfile]

Để chạy Docker image

docker run [OPTIONS] [image tag]

Các OPTIONS để tạo Docker image có thể xem đầy đủ tại <https://docs.docker.com/engine/reference/commandline/build/>

Trong đồ án này ta sẽ chỉ sử dụng một số các OPTIONS như -t, -rm, -rmi.

Cụ thể từ đồ án này ta cần tạo các image chứa và chạy các đoạn code của người dùng viết và gửi từ website. Đoạn code gửi về đơn giản là in ra dòng chữ “Hello world!!” trên ngôn ngữ python thì ta cần một tập tin python có chứa đoạn code và một Dockerfile như sau

src

|---app

|---Dockerfile

|---main.py

Nội dung của Dockerfile

FROM python:3

WORKDIR /usr/src/app

COPY . .

CMD [ "python", "./main.py" ]

Sau đó ta bắt đầu tạo ra Docker image từ Dockerfile

docker build -t pythonscript ./src/app

Để khởi chạy Docker image vừa được tạo ta sử dụng câu lệnh docker run

docker run pythonscript

Trên đây là các bước cơ bản để khởi tạo Docker image và chạy một Docker container từ image được tạo đơn giản.

## Giới thiệu Django và Django REST Framework

### Django là gì?

Django là một web framework mã nguồn mở được viết bằng python và phát hành vào năm 2005 bởi Django Software Foundation. Mục tiêu của django là giúp đơn giản hóa việc phát triển website bằng python

### Mô hình MVT

Django sử dụng mô hình được gọi là MVT (Model – View – Template). Thực tế đây là một biến thể từ mô hình MVC (Model – View – Controller) và cũng chia ứng dụng ra thành 3 thành phần để xử lý: phần model là nơi chứa các thực thể để tương tác với cơ sở dữ liệu; phần view là nơi xử lý các logic tương đương với phần controller trong MVC; phần template là nơi hiển thị ra giao diện cho người dùng tương đương với view trong MVC.

|  |  |  |
| --- | --- | --- |
| Mô hình MVC | Mô hình MVT | Chức năng |
| Model | Model | Tương tác với cơ sở dữ liệu |
| View | Template | Hiển thị giao diện |
| Controller | View | Xử lý logic |

Bảng 2.1: Mô hình MVC và MVT

### Tổng quan về Django REST Framework

#### API

API là viết tắt của Application Programming Interface, là tập hợp các quy tắc và cơ chế mà thông qua đó giúp các ứng dụng và các thành phần tương tác với nhau hay có thể nói các ứng dụng hoặc các thành phần tương tác với nhau thông qua API.

#### REST và RESTful API

REST là viết tắt của Representational state transfer, là một kiểu kiến trúc phần mềm được tạo ra để thiết kế và phát triển ứng dụng sử dụng mạng. Nó là tập hợp các ràng buộc được sử dụng cho việc triển khai các phương tiện internet như web.

REST thường được sử dụng trong việc phát triển API. Một API tuân theo các ràng buộc của REST được gọi là RESTful API, các API này sử dụng phương thức HTTP đơn giản để kết nối đến các tài nguyên giữa các máy thông qua url có các tham số được mã hóa và dùng Json hoặc xml để truyền dữ liệu.

#### Django REST Framework

Django REST Framework là một framework được phát triển để giúp xây dựng các RESTful API trong python django một cách đơn giản và thuận tiện

#### GET và POST

GET và POST là hai phương thức HTTP đều được dùng để gửi dữ liệu về server để xử lý.

Đối với GET dữ liệu sẽ được gửi thông qua url nghĩa là các url sẽ được đính kèm với các dữ liệu mà chúng ta gửi. Server sẽ phân tích url đó và lấy các dữ liệu để xử lý.

Đối với POST dữ liệu sẽ được gửi thông qua một form HTML nên các giá trị sẽ không gán bên trên url nên các url của POST sẽ gọn hơn là GET.

### Tạo project

Để tạo project python có sử dụng django ta cần tạo một project python thông thường và cài đặt django.

Ở đồ án này ta sẽ sử dụng Pycharm để viết phần python django. Trước tiên ta tạo project trong Pycharm

Graphical user interface, text, application, email

Description automatically generated

Hình 2.2: Tạo project Pycharm

Sau đó để cài đặt django ta chạy lệnh pip trong terminal

pip install django

Để khởi tạo django project ta thực hiện lệnh trong terminal

django-admin startproject myproject

Sau khi chạy xong lệnh khởi tạo django project ta sẽ có được thư mục của project như sau

new\_project

|---myproject

|---myproject

|---\_init\_.py

|---asgi.py

|---settings.py

|---urls.py

|---wsgi.py

|---manage.py

Để chạy server ta vào thư mục của project vừa tạo và trên cửa sổ terminal gõ lệnh

python manage.py runserver

Server sẽ được khởi chạy mặc định tại port 8000

Graphical user interface, text, application

Description automatically generated

Hình 2.3: Giao diện tại localhost:8000 sau khi chạy server

### Tạo ứng dụng trong Django

Trong một project ta sẽ có thể tạo một hoặc nhiều ứng dụng, mỗi ứng dụng sẽ thực hiện một công việc riêng biệt.

Sau khi đã có server ta bắt đầu tạo và viết ứng dụng, để tạo ứng dụng đầu tiên trong Django ta sử dụng câu lệnh trong cửa sổ terminal

python manage.py startapp first\_app

Lúc này tại thư mục của project sẽ xuất hiện thêm một thư mục mới mang tên “first\_app” có cấu trúc như sau

new\_project

|---myproject

|---first\_app

|---migrations

|---\_\_init\_\_.py

|---admin.py

|---apps.py

|---models.py

|---tests.py

|---views.py

|---myproject

|---manage.py

Trong apps.py chính là nơi chứa config của ứng dụng.

Để ứng dụng được duyệt qua và chạy thì trong mục settings.py tại project ta khai báo ứng dụng tại vùng INSTALLED\_APPS

INSTALLED\_APPS = [

‘first\_app.apps.FirstAppConfig’

]

Tạo thêm các file python trong app để có cấu trúc như sau

|---first\_app

|---migrations

|---\_\_init\_\_.py

|---admin.py

|---apps.py

|---models.py

|---serializers.py

|---tests.py

|---urls.py

|---views.py

File urls.py của app sẽ là nơi khởi tạo các routers và các url tại đây sẽ được duyệt để dẫn đến các trang gọi API.

Để các url của app được duyệt thì tại file urls.py của project ta gọi file urls.py từ app như sau

urlpatterns = [

path(‘’, include(‘first\_app.urls’)),

path(‘admin/’, admin.site.urls),

]

Tại urls.py của app ta khởi tạo router

from rest\_framework.routers import DefaultRouter

router = DefaultRouter()

urlpatterns = [

path(‘’, include(router.urls))

]

Sau khi có router ta đã có thể khai báo các url cho API

Ta sẽ khai báo model cho API tại models.py, đây là nơi sẽ chứa các model. Trong đồ án này ta sẽ có một model của API để nhận thông tin từ người dùng bao gồm mã nguồn, ngôn ngữ và dữ liệu nhập.

Model được khai báo dưới dạng class

from django.db import models

class Code(models.Model):

source = models.TextField()

language = models.CharField(max\_length=20)

input = models.TextField(null=True)

Tại file serializers.py là nơi khai báo các class để tiến hành serializer các dữ liệu hoặc deserializer các dữ liệu. Ta sẽ serializer dữ liệu từ các field đã khai báo trong model như sau

from rest\_framework.serializers import ModelSerializer

from .models import Code

class CodeSerializer(ModelSerializer):

class Meta:

model = Code

fields = ['source', 'language', 'input']

Sau đó ta sẽ tiến hành xử lý API tại views.py và trả về các kết quả cho người dùng thông qua viewset

class CodeViewSet(viewsets.ViewSet,

generics.CreateAPIView):

serializer\_class = CodeSerializer

Ta sẽ có một phương thức POST để chạy mã nguồn từ người dùng gửi về và trả ra kết quả từ đoạn mã nguồn đó nên phương thức sẽ trả về như sau

def create(self, request, \*args, \*\*kwargs):

return Response(

{

"source": source.get('source'),

"input": source.get('input'),

"output": contents,

}, status=status.HTTP\_200\_OK

)

Sau khi xử lý và tạo được viewset ta đăng kí router cho viewset

router.register('compiler', views.CodeViewSet, 'compiler')

Lúc này ta đã có một API chỉ có phương thức POST được sử dụng để chạy mã nguồn từ người dùng.

Graphical user interface, text, application, email

Description automatically generated

Hình 2.4: Giao diện khi vào localhost

Graphical user interface, application

Description automatically generated

Hình 2.5: API mới tạo chỉ có phương thức POST

Để phía front end có thể sử dụng API này ta cần khai báo đường dẫn url của phía front end trong settings.py để cho phép phía front end được sử dụng API

CORS\_ALLOWED\_ORIGINS = [

"http://localhost:3000/",

]

## Giới thiệu React

### Tổng quan về React

React là một thư viện front-end mã nguồn mở được viết bằng Javascript để xây dựng giao diện người dùng. Được duy trì và phát triển bởi facebook cùng cộng đồng các lập trình viên, React được sử dụng để xây dựng các website dạng single-page hoặc các ứng dụng di động.

React chia các thành phần trên giao diện người dùng phức tạp thành các thành phần nhỏ, độc lập gọi là component. Nhờ đó mà việc thiết kế và phát triển giao diện người dùng trở nên đơn giản hơn nhiều.

### DOM ảo trong React

React không render trực tiếp lên DOM của trình duyệt mà sẽ tương tác thông qua một DOM ảo là một bản sao của DOM thật.

Diagram

Description automatically generated

Hình 2.6: Tương tác DOM ảo (<https://viblo.asia/p/virtual-dom-trong-reactjs-YWOZrMdEKQ0>)

Mỗi lần có thay đổi trong components thì nó sẽ được render lại cập nhật trên DOM ảo sau đó DOM ảo sẽ so sánh sự thay đổi và cập nhật lên DOM thật làm sao cho tối ưu nhất. Nhờ vậy mà ta không cần phải dựng lại DOM thật từ đầu mỗi lần thay đổi gây mất thời gian hoặc tốn tài nguyên giúp hiệu suất làm việc tốt hơn.

### Components và props trong React

Trong React, các thành phần trong UI được chia thành các components độc lập giúp cho ta có thể tái sử dụng chúng nhiều lần cũng như đảm bảo việc chỉnh sửa nâng cấp thành phần đó không ảnh hưởng đến các thành phần khác.

Có thể nói mỗi components trong React chính là một function trong javascript.

Props trong components chính là một đối tượng được truyền vào trong components và trả về React element. Nhờ có props các components có thể giao tiếp qua lại với nhau. Khi một props được truyền vào component con từ cha thì component con chỉ có thể đọc và không thể chỉnh sửa props.

Một components trong React có dạng như sau

function Hello(props) {

return <h1>Hello {props.name}</h1>;

}

Hoặc có thể viết React component dạng class

Class Hello extends React.Component {

render() {

return <h1>Hello {this.props.name}</h1>

}

}

Một component có thể được gọi như một thẻ html bình thường và render như sau

function Hello(props) {

return <h1>Hello {props.name}</h1>

}

ReactDOM.render(

<Hello name=”Minh” />,

Document.getElementById(“root”)

);

Đoạn mã nguồn trên sẽ render ra dòng chữ “Hello Minh” trên trang.

### State trong React

Không như props, state là một đối tượng bên trong components được sử dụng để lưu trữ dữ liệu hoặc thông tin của components và có thể thay đổi dữ liệu bất cứ khi nào. State chỉ tồn tại trong components chứa nó và không thể truyền qua các components khác, mỗi khi state thay đổi thì components sẽ được render lại.

Khởi tạo state trong class

Class Hello extends React.Component {

constructor(props) {

super(props);

//Chỉ định một state

this.state = { name: "Minh" };

}

render() {

return <h1>Hello {this.state.name}</h1>

}

}

Để thay đổi giá trị của state ta không thể gán trực tiếp mà phải thông qua phương thức setState().

### Vòng đời của React component

Mỗi một React component đều sẽ có một vòng đời riêng với 3 giai đoạn là mounting, updating, unmounting.

Diagram

Description automatically generated with low confidence

Hình 2.7: Vòng đời của một React component (https://viblo.asia/p/vong-doi-cua-mot-react-component-RQqKLMRzZ7z)

constructor(props): là phương thức được gọi khi khởi tạo component, đây cũng là nơi ta khai báo state cho component. Để sử dụng hàm này ta khai báo một tham số đầu vào props cho nó và gọi super(props) còn nếu không thì hàm sẽ không được gọi.

render(): là phương thức để kết xuất các component lên giao diện cho người dùng sử dụng. Phương thức này sẽ được gọi mỗi khi component được khởi tạo hay có thay đổi.

componentWillMount(): là phương thức được thực hiện trước khi render, thực hiện một số tác vụ và chỉ được gọi một lần duy nhất.

componentDidMount(): là phương thức được thực hiện sau khi component được render lần đầu tiên và cũng được gọi một lần duy nhất.

componentWillReceiveProps(nextProps): đây là phương thức được gọi mỗi khi props thay đổi.

shouldComponentUpdate(nextProps, nextState): phương thức này sẽ trả về kết quả là true nếu component sẽ được cập nhật hoặc false nếu component không được cập nhật. Giá trị mặc định của phương thức này là true.

componentWillUpdate(nextProps, nextState): phương thức này được gọi sau khi phương thức shouldComponentUpdate() trả về kết quả là true. Phương thức render() sẽ được gọi ngay sau phương thức này.

componentDidUpdate(prevProps, prevState): phương thức này được gọi ngay sau khi một component được render từ lần thứ hai trở đi.

componentWillUnmount(): phương thức này được gọi khi loại bỏ một component khỏi DOM. Các thao tác dọn dẹp component sẽ được xử lý tại phương thức này.

### React Hook

Hooks là một phần mới được thêm vào React 16.8. Hooks cho phép chúng ta sử dụng các trạng thái và tính năng của React mà không cần viết một class.

Với hooks việc viết một component trở nên gọn hơn rất nhiều và không phức tạp như class state có thể được sử dụng ngay trong function.

Các hooks cơ bản thường xuyên sử dụng đó là useState và useEffect.

Thay vì trước đây trong class ta phải khai báo states bên trong phương thức constructor thì với React Hook việc khai báo sẽ như sau

import React, { useState } from 'react';

function Example() {

const [count, setCount] = useState(0);

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>

Click me

</button>

</div>

);

}

Với useEffect nó giúp ta xử lý các thao tác, nhiệm vụ tương tự như các phương thức componentDidMount, componentDidUpdate và componentWillUnMount trong vòng đời của một component.

import React, { useState, useEffect } from 'react';

function Example() {

const [count, setCount] = useState(0);

// Similar to componentDidMount and componentDidUpdate:

useEffect(() => {

// Update the document title using the browser API

document.title = `You clicked ${count} times`;

});

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>

Click me

</button>

</div>

);

}

### Typescript và tạo project React

Typescript là ngôn ngữ được tạo ra dựa trên javascript do microsoft phát triển và duy trì. Về bản chất thì typescript chính là javascript kết hợp với các khai báo các kiểu dữ liệu và các lớp đối tượng.

Ví dụ với đoạn code typescript

function sum(a: number, b: number){

return a+b

}

let a = 3

let b = “4”

sum(a, b) //Lỗi vì b không phải number

Trong javascript đoạn code trên sẽ là

function sum(a, b){

return a+b

}

let a = 3

let b = “4”

sum(a, b) //Ket qua tra ve la 34

Nhờ có khai báo kiểu cho biến mà việc kiểm tra lỗi trong typescript sẽ dễ dàng hơn javascript.

Để sử dụng typescript với React ta cần cài đặt nodejs trước tiên tại website <https://nodejs.org>.

Sau đó tại command line ta cài đặt typescript với lệnh

npm install -g typescript

Tiếp đó tạo project React đầu tiên sử dụng typescript

npx create-react-app my-app --template typescript

Sau đó để chạy react app ta chỉ cần gõ lệnh

npm start

### Sử dụng một số thư viện

Trong đồ án này để tạo được một vùng cho người dùng thoải mái nhập mã nguồn và có hỗ trợ đầy đủ các syntax của ngôn ngữ như python, java, javascript thì có sử dụng một package được viết bởi người dùng dựa trên monaco editor.

Ta sẽ sử dụng component Editor để tạo một vùng nhập mã nguồn đơn giản

import Editor from "@monaco-editor/react";

function App() {

function handleEditorChange(value, event) {

console.log("here is the current model value:", value);

}

return (

<Editor

height="90vh"

defaultLanguage="javascript"

defaultValue="// some comment"

onChange={handleEditorChange}

/>

);

}

Bài đồ án sử dụng package material-ui của google để thiết kế và trình bày các giao diện, một số component được sử dụng có thể kể đến như Button để tạo nút submit mã nguồn gửi xuống server, TextAreaAutoSize để tạo 2 vùng input và output.

Tiếp đến là phần gọi API từ server, để có thể gọi API từ server ta cài một package gọi là axios.

Sử dụng axios gọi API từ server ta sẽ tạo ra một config

import axios from "axios"

export let endpoints = {

'compiler': '/compiler/'

}

export default axios.create({

baseURL: 'http://localhost:8000/'

})

Lúc này axios sẽ tạo ra các yêu cầu gọi API từ localhost:8000.

Ta có một interface để nhận các dữ liệu đưa cho server xử lý như sau

export default interface IRuncode {

source: string,

language?: string,

input?: string,

output?: string,

}

Tạo một service để tạo lệnh gọi API

import ApiConfig, { endpoints } from "../configs/ApiConfig";

import IRuncode from "../types/IRuncode";

const create = (data: IRuncode) => {

return ApiConfig.post(endpoints['compiler'], data)

}

const CompilerService = {

create,

}

export default CompilerService;

Sau khi đã có service thì tại nơi cần sử dụng ta chỉ cần gọi service đã được tạo là có thể lấy dữ liệu từ API.

# CÔNG CỤ ONLINE COMPILER

Chương này sẽ cho ta thấy kiến trúc của hệ thống cũng như cách mà nó vận hành.

## Kiến trúc hệ thống

Để thực hiện biên dịch mã nguồn từ website thì ta sẽ tìm cách đưa mã nguồn đó xuống máy chủ sau đó thực thi mã nguồn tại máy chủ rồi trả về kết quả cho người dùng.

Diagram

Description automatically generated

Hình 3.1: Kiến trúc ban đầu

Ở phía web server sẽ cài đặt hết tất cả công cụ và môi trường để chạy các đoạn mã nguồn khác nhau ví dụ như để chạy python, java, javascript thì ở server ta sẽ cài đặt các bộ công cụ để chạy tương ứng tại đó.

Nhưng nếu ta biên dịch và chạy mã nguồn trực tiếp tại server như vậy sẽ gây nguy hiểm cho hệ thống vì tính bảo mật vô cùng lỏng lẻo và sẽ không thể kiểm soát được mã nguồn được thực thi trong server cũng như việc cài đặt quá nhiều các công cụ vào máy chủ như thế sẽ chiếm rất nhiều tài nguyên và gây ra các rủi ro.

Ở đồ án này ta sẽ sử dụng Docker để cung cấp container cho mỗi mã nguồn được người dùng gửi từ website xuống server. Các mã nguồn này sẽ được tich hợp vào Docker container và chạy bên trong đó. Vì các Docker container này được cô lập với máy chủ nên rủi ro về bảo mật sẽ được giảm đi so với việc ta chạy mã nguồn trực tiếp trên server.

Graphical user interface, diagram, application

Description automatically generated

Hình 3.2: Kiến trúc hệ thống sử dụng Docker

Nhờ cung cấp các container riêng và các container này đảm nhận duy nhất một vai trò cho mỗi ngôn ngữ nên ta không cần phải cài đặt hết tất cả các công cụ, môi trường để chạy ngôn ngữ trên server nữa vì các container đã được cài đặt những thứ cần thiết để chạy mã nguồn được người dùng gửi đến server.

## Thiết kế API

Để có thể chạy được mã nguồn ta cần các thông tin từ phía website bao gồm mã nguồn, loại ngôn ngữ, các giá trị đầu vào nếu trong mã nguồn của người dùng có phần nhập xuất.

Từ đó ta có một file json chưa các thông tin như sau

{

“source”: “Mã nguồn”

“language”: “Ngôn ngữ sử dụng”

“input”: “Dữ liệu nhập nếu có”

}

Sau khi nhận vào các thông tin trên, server sẽ xử lý và tạo các Docker container tương ứng để chạy đoạn mã nguồn từ người dùng. API sẽ trả về kết quả cho người dùng

{

“output”: “Kết quả sau khi thực thi”

}

## Hiện thực API

Để thực hiện được API này ta có bước đầu là nhận file json có các thông tin trên xuống server sau đó tiến hành viết các Docker image với các Dockerfile tương ứng với mỗi ngôn ngữ như sau

Với ngôn ngữ java

FROM openjdk:11

WORKDIR /app

USER root

COPY . .

RUN chmod a+x ./main.java

RUN chmod a+x ./entrypoint.sh

ENTRYPOINT [“./entrypoint.sh”]

Với ngôn ngữ python

FROM python:3.8-alpine

WORKDIR /app

USER root

COPY . .

RUN chmod a+x ./main.py

RUN chmod a+x ./entrypoint.sh

ENTRYPOINT [“./entrypoint.sh”]

Với ngôn ngữ Javascript

FROM node:16-alpine3.11

WORKDIR /app

USER root

COPY . .

RUN chmod a+x ./main.js

RUN chmod a+x ./entrypoint.sh

ENTRYPOINT [“./entrypoint.sh”]

Và với mỗi lần xây dựng một Docker image cho mỗi ngôn ngữ cũng sẽ tương ứng với một file shell entrypoint khác nhau

Với ngôn ngữ Java

#!/usr/bin/env sh

javac main.java

java Main < input.txt

Với ngôn ngữ python

#!/usr/bin/env sh

python3 main.py < input.txt

Với ngôn ngữ javascript

#!/usr/bin/env sh

node main.js < input.txt

Với mỗi ngôn ngữ sẽ gồm các file trên ta sẽ có cấu trúc thư mục như sau

docker

|---java

|---Dockerfile

|---entrypoint.sh

|---javascript

|---Dockerfile

|---entrypoint.sh

|---python

|---Dockerfile

|---entrypoint.sh

Mỗi lần request được gửi xuống server thì sẽ có các file main với tương ứng được tạo theo mỗi ngôn ngữ để chứa mã nguồn, file input chưa thông tin nhập vào, file output sẽ chứa các kết quả hoặc lỗi sau khi chạy mã nguồn.

Server sẽ bắt đầu chạy câu lệnh để tạo Docker image từ Dockerfile theo mỗi ngôn ngữ. Ví dụ ta có một dòng lệnh “print(‘Hello’)” được viết bởi ngôn ngữ python và gửi tới server thì ta sẽ có dòng lệnh tạo docker image như sau

docker image build ./docker/python -t pythonimagetag

Sau đó server sẽ chạy Docker image vừa tạo bằng lệnh run

docker run –rm pythonimagetag

Khi đã lấy về được kết quả, image sẽ được gỡ bỏ để giải phóng vùng nhớ

docker rmi pythonimagetag

Sau khi đã xử lý xong, kết quả từ file output sẽ được ghi và trả lên cho người dùng.

## Demo công cụ online compiler

Giao diện khi truy cập vào sẽ có một vùng để ta nhập mã nguồn, một phần danh sách các ngôn ngữ được hỗ trợ cho người dùng chọn, một nút Run để chạy đoạn mã nguồn được nhập vào từ người dùng, một vùng để người dùng nhập input cho đoạn mã nguồn nếu có và một vùng để xuất ra kết quả từ đoạn mã nguồn.

Graphical user interface, application

Description automatically generated

Hình 3.3: Giao diện của công cụ

A picture containing text

Description automatically generated

Hình 3.4: Phân vùng trả về kết quả sau khi chạy mã

Ta sẽ thử chạy mã nguồn đầu tiên trên công cụ với ngôn ngữ python. Hiện tại có thể chọn một trong 3 ngôn ngữ trên công cụ

Graphical user interface, application, Word

Description automatically generated

Hình 3.5: Chọn ngôn ngữ lập trình

Nhập thử đoạn mã nguồn in ra dòng chữ “Hello world!!!” với ngôn ngữ python

Graphical user interface, text, application, Word

Description automatically generated

Hình 3.6: Nhập mã nguồn

Sau đó ta nhấp chọn nút Run và sẽ mất một khoảng thời gian chờ để server xử lý và kết quả sẽ được trả về ở vùng Output

Graphical user interface, text, application

Description automatically generated

Hình 3.7: Nhấn vào nút Run

Khi đã xử lý xong thì vùng Output sẽ hiển thị ra kết quả của đoạn mã nguồn được nhập.

A picture containing text

Description automatically generated

Hình 3.8: Kết quả được trả về

Trên đây là thử một đoạn mã nguồn đơn giản không có đầu vào.

Đối với đoạn mã nguồn có đầu vào, ta nhập đầu vào tương ứng ở vùng Input và kết quả cũng sẽ được in ở vùng Output.

Graphical user interface, text, application

Description automatically generated

Hình 3.9: Đoạn mã có input

Kết quả trả về cho đoạn mã trên hình 3.9 sẽ là “Hello Minh”

Text

Description automatically generated with medium confidence

Hình 3.10: Kết quả mã nguồn có input

Trên đây là toàn bộ hoạt động biên dịch mã nguồn mà công cụ online compiler thực hiện.

# Chương 4. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Đây là chương kết thúc để rút ra kết luận, đánh giá và định hướng phát triển cho đồ án.

## Kết quả đạt được

Sau khi tìm hiểu và tiến hành thực hiện, công cụ compiler đã hoàn thành được khả năng chạy được mã nguồn từ website.

Học hỏi thêm được các kiến thức về python Django, React, Docker

## Chức năng cần cải thiện

Do hạn chế về kinh nghiệm và thời gian thực hiện cũng như ảnh hưởng từ dịch bệnh nên đồ án có nhiều hạn chế về cả sự đa dạng trong chức năng cũng như hiệu năng và các trải nghiệm.

Việc sử dụng Docker còn chưa đủ chuyên sâu nên việc vận dụng vẫn còn hạn chế đó là mỗi lần chạy lại mã nguồn từ người dùng gửi từ website sẽ phải build lại Docker image từ đầu làm cho hiệu năng mỗi lần chạy sẽ có thể rất chậm và phụ thuộc nhiều vào mạng.

Giao diện sơ sài, chỉ có chức năng chạy mã nguồn

## Hướng phát triển

Từ công cụ online compiler ta có thể phát triển lên thành một website học lập trình, luyện thuật toán.

Có thể nghiên cứu và tìm ra một giải pháp, mô hình mới trong việc chạy mã nguồn, tổ chức lại Docker container để hiệu năng mỗi lần biên dịch mã nguồn trở nên nhanh hơn và có tính bảo mật tốt hơn.

Phát triển thêm các bộ công cụ khác cho việc chỉnh sửa mã nguồn trên website được tốt hơn.

Từ công cụ hiện tại có thể tổ chức lại để thêm khả năng chạy các loại mã nguồn phức tạp hơn, sử dụng các thư viện của ngôn ngữ lập trình đa dạng hơn.

Thêm nhiều ngôn ngữ hơn đa dạng trình biên dịch

TÀI LIỆU THAM KHẢO

(Theo chuẩn IEEE)

“Docker Document,” [Online]. Available: <https://docs.docker.com/>

Wikipedia, “Docker,” 10 10 2021. [Online]. Available: https://vi.wikipedia.org/wiki/Docker\_(phần\_mềm)

“Docker,” [Online]. Available: <https://searchitoperations.techtarget.com/definition/Docker>

“Django documentation,” [Online]. Available: <https://docs.djangoproject.com/en/3.2/>

“Django REST framework,” [Online]. Available: <https://www.django-rest-framework.org/>

“Python Django,” [Online]. Available: <https://www.youtube.com/playlist?list=PLlVHoHHccp28J9qaK2sau8hs346bxiXsA>

“Tutorial: Intro to React,” [Online]. Available: <https://reactjs.org/tutorial/>

“Javascript,” [Online]. Available: <https://www.youtube.com/playlist?list=PLlVHoHHccp2_jIce6hcp5WhNhEvc6NYrQ>

“Tìm hiểu căn bản typescript với reactjs,” [Online]. Available: <https://viblo.asia/p/tim-hieu-can-ban-typescript-voi-reactjs-bWrZnxev5xw>

“Virtual DOM trong ReactJS,” [Online]. Available: <https://viblo.asia/p/virtual-dom-trong-reactjs-YWOZrMdEKQ0>

“Vòng đời của Component trong reactjs,” [Online]. Available: <https://levunguyen.com/laptrinhjavascript/2021/12/08/vong-doi-component-trong-reactjs>

“Vòng đời của một React Component,” [Online]. Available: <https://viblo.asia/p/vong-doi-cua-mot-react-component-RQqKLMRzZ7z>

“Cùng tìm hiểu về các hook trong React hooks,” [Online]. Available: <https://viblo.asia/p/cung-tim-hieu-ve-cac-hook-trong-react-hooks-Ljy5VYgjlra>

“TypeScript Documentation,” [Online]. Available: <https://www.typescriptlang.org/docs/>

“@monaco-editor/react,” [Online]. Available: <https://www.npmjs.com/package/@monaco-editor/react>

“Button,” [Online]. Available: <https://mui.com/components/buttons/>

“Usage,” [Online]. Available: <https://mui.com/getting-started/usage/>

“Textarea Autosize,” [Online]. Available: <https://mui.com/components/textarea-autosize/>