ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

Thực tập công nghệ phần mềm

Báo cáo

Ecommerce Chatbot sử dụng Rasa

GVHD: Băng Ngọc Bảo Tâm
SV: Trần Thị Ngọc Diệp - 1827005
Lương Thành Nhân - 1820047
Nguyễn Lê Quốc Cường - 1710721
Nguyễn Văn Quyền Lâm - 1711901

TP. HỒ CHÍ MINH, THÁNG 08/2020

Mục lục

1 Giới thiệu

1.1 Tổng quan (Nhân và Diệp)

Mục đích của chúng tôi là tạo ra một chatbot, có chức năng hỗ trợ người dùng ứng dụng trong thương mại điện tử, cụ thể là trong việc mua bán sách.

Chatbot có khả năng thực hiện một số tác vụ cơ bản như chào hỏi khách hàng, phân tích yêu cầu của khách hàng, từ đó thực hiện một số chức năng như hỗ trợ khách hàng tìm kiếm sách. Ngoài ra chatbot còn có chức năng hỗ trợ khách hàng làm việc với hệ thống Blockchain như hỗ trợ khách hàng tạo tài khoản, kiểm tra tài khoản, giao dịch và kiểm tra thông tin giao dịch.

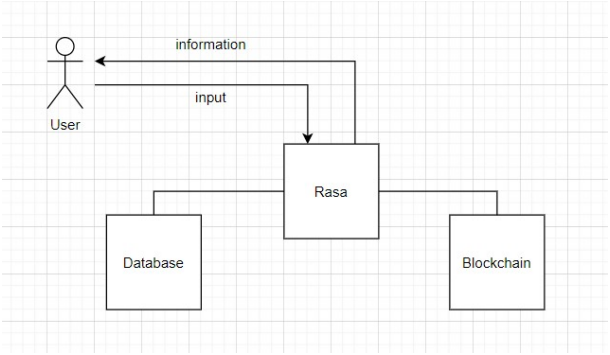
1.2 Công nghệ sử dụng (Nhân và Diệp)

Chatbot của chúng tôi sử dụng Rasa, một AI Framework mã nguồn mở cung cấp công cụ máy học (machine learning) để xử lý hội thoại theo ngữ cảnh. Rasa được huấn luyện dựa trên học có giám sát (supervised learning).

Chatbot cũng có thể thực hiện các hành động theo yêu cầu của người dùng, bao gồm truy xuất cơ sở dữ liệu (database) và gọi API từ hệ thống Blockchain để thực hiện một số tác vụ trên hệ thống này.

2 Kiến trúc hệ thống

Hình dưới mô tả kiến trúc khối của hệ thống, mô tả một cách tổng quát cách thức hoạt động của thệ thống.



Hình 1: Kiến trúc tổng quan của hệ thống

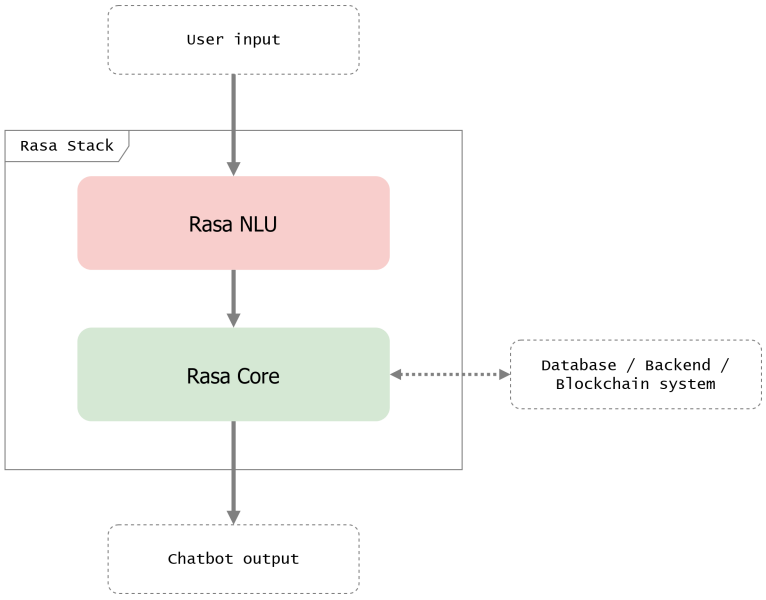
Hệ thống gồm 3 khối chính: Rasa, Database và Blockchain. Rasa chịu trách nhiệm phát hiện ý định của người dùng, từ đó lấy thông tin cần thiết từ cơ sở dữ liệu và hệ thống Blockchain, tổng hợp lại và trả thông tin thích hợp về cho người dùng.

Chi tiết của mỗi khối được trình bày ở những phần tiếp theo.

2.1 Rasa

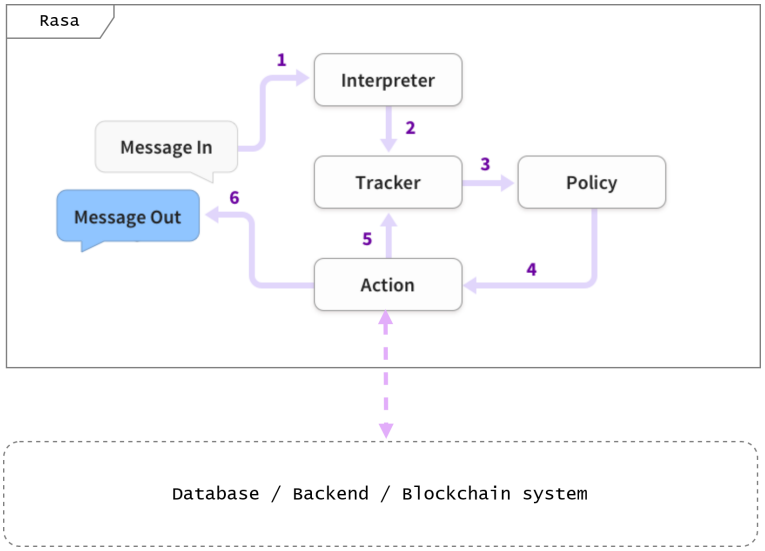
Rasa, hay còn được gọi là Rasa Stack, là ngăn xếp gồm hai thành phần độc lập:

- Rasa NLU: Thư viện hiểu ngôn ngữ tự nhiên (Natural Language Understanding). NLU hiểu input của người dùng dựa trên dữ liệu đã được huấn luyện.
  - Phân loại ý định (intent classification): xác định ý định của input người dùng dựa trên những ý định đã được học.
  - Trích xuất thực thể (entity extraction): Phát hiện và trích xuất những dữ liệu có cấu trúc.
- Rasa Core: Dựa trên học máy để dự đoán hành động tiếp theo dựa trên dữ liệu từ NLU, lịch sử cuộc hội thoại và dữ liệu huấn luyện.



Hình 2: Rasa NLU và Rasa Core trong Rasa Stack

Hình bên dưới đi sâu hơn vào chi tiết của luồng xử lý của Rasa. Trong đó Rasa NLU đảm nhận công việc 1,2 và Rasa Core đảm nhận các công việc còn lại.



Hình 3: Luồng xử lý trong Rasa

Các bước xử lý:

1. Nhận input từ người dùng và đưa vào Interpreter, Interpreter chuyển đổi input thành từ điển bao gồm: input ban đầu, ý định, các thực thể.
2. Tracker sẽ theo dõi trạng thái của cuộc hội thoại bằng cách lưu trữ thông tin phân tích được từ NLU, bên cạnh đó Tracker còn lưu trữ thông tin về đoạn hội thoại, các slot (những biến nhớ dùng để giữ thông tin về nhiều thực thể xuyên suốt đoạn hội thoại), thông tin về sự kiện, hành động đặc biệt (form action, followup action).
3. Policy nhận trạng thái hiện tại từ Tracker.
4. Policy xác định hành động tiếp theo của chatbot.
5. Tracker lưu hành động tiếp theo đã được xác định bởi Policy.
6. Phản hồi được gửi đến cho người dùng.

2.1.1 Rasa NLU

- Input và Output của NLU.

  Chẳng hạn người dùng nhập vào input như sau:

  "Tôi muốn kiểm tra số dư tại địa chỉ 0x81b7E08F65Bdf5648606c89998A9CC8164397647"

Cấu trúc dữ liệu trả về từ NLU là:


```
{
  "intent": {
    "name": "get_balance_with_userAddress",
    "confidence": 0.9980816841125488
  },
  "entities": [
    {
      "start": 36,
      "end": 78,
      "value": "0x81b7E08F65Bdf5648606c89998A9CC8164397647",
      "entity": "userAddress",
      "confidence": 0.9863805285936216,
      "extractor": "CRFEntityExtractor"
    }
  ],
  "intent_ranking": [
    {
      "name": "get_balance_with_userAddress",
      "confidence": 0.9980816841125488
    },
    {
      "name": "gen_address",
      "confidence": 0.0004896865575574338
    },
    {
      "name": "send_transaction",
      "confidence": 0.0003805412270594388
    },
    {
      "name": "give_transaction_info",
      "confidence": 0.00022241377155296504
    },
    {
      "name": "get_balance",
      "confidence": 0.0001866378734121099
    },
    {
      "name": "greet",
      "confidence": 0.0001827961386879906
    },
    {
      "name": "buy_book",
      "confidence": 0.00014661182649433613
```

```
    },
    {
      "name": "send_file",
      "confidence": 0.00013778437278233469
    },
    {
      "name": "thanks",
      "confidence": 0.00011292178533039987
    },
    {
      "name": "introduce",
      "confidence": 5.887631414225325e−05
    }
  ],
  "text": "Tôi muốn kiểm tra số dư tại địa chỉ
  0x81b7E08F65Bdf5648606c89998A9CC8164397647"
}
```


- Dữ liệu huấn luyện

  Dữ liệu huấn luyện cho Rasa NLU được chứa ở file `data\nlu.md`, viết bởi ngôn ngữ Markdown. Chúng tôi dùng hai dạng dữ liệu huấn luyện là:

  – Ví dụ thông thường: Bao gồm dữ liệu dạng văn bản (text), ý định (intent) của người dùng cho dữ liệu đó, và thực thể (entity) có thể có trong dữ liệu đó. Bên dưới là một ví dụ về dữ liệu huấn luyện thông thường cho ý định `get_balance_with_userAddress`.

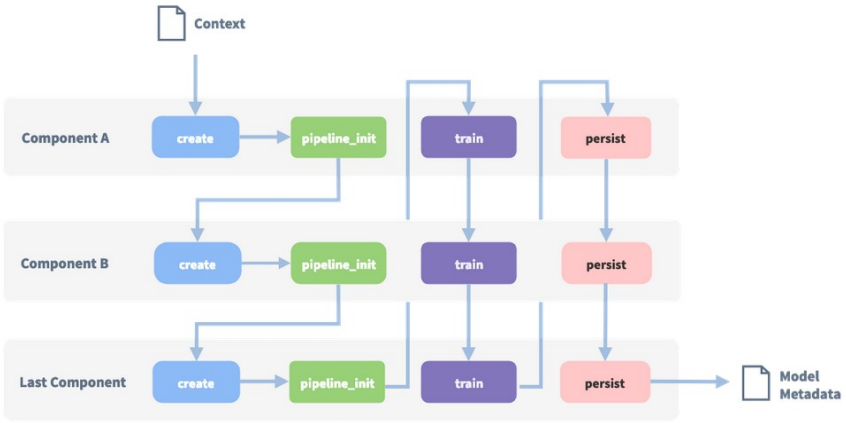
  ```
  ## intent: get_balance_with_userAddress
      – Địa chỉ của tôi là [0x467d456bd81b34109aefd5a5eccc145e1688732d]
  (userAddress), tôi muốn xem số dư
  ```

  – Biểu thức chính quy: Sử dụng để hỗ trợ xác định thực thể. Bên dưới là một ví dụ về sử dụng biểu thức chính quy để hỗ trợ trích xuất thực thể `userAddress`.

  ```
  ## regex: userAddress
      – \b0x[0−9a−fA−F]{40}\b
      – \b[13][a−km−zA−HJ−NP−Z1−9]{25,34}\b
      – \b(bc1|[13])[a−zA−HJ−NP−Z0−9]{25,39}\b
      – \b(tb1|[2nm]|bcrt)[a−zA−HJ−NP−Z0−9]{25, 40}\b
  ```

- Đường ống xử lý (pipeline)

  Input từ người dùng được xử lý bởi một chuỗi các thành phần trong NLU, những thành phần (component) này được thực thi lần lượt trong giai đoạn huấn luyện, tạo nên đường ống xử lý. Những thành phần xử lý của NLU được xác định ở file `config.yml`. Sử dụng các thành phần khác nhau cho phép tối ưu hóa mô hình dựa theo dữ liệu và mục tiêu của người huấn luyện.



Hình 4: Các thành phần được gọi theo thứ tự trong quá trình huấn luyện NLU

Chatbot của chúng tôi sử dụng đường ống xử lý NLU như sau:

```
language: "vi"

pipeline:
  - name: "WhitespaceTokenizer"
  - name: "RegexFeaturizer"
  - name: "CRFEntityExtractor"
  - name: "EntitySynonymMapper"
  - name: "CountVectorsFeaturizer"
  - name: "CountVectorsFeaturizer"
    analyzer: "char_wb"
    min_ngram: 1
    max_ngram: 4
  - name: "EmbeddingIntentClassifier"
```

- `WhitespaceTokenizer`: sử dụng khoảng trắng (whitespace) để tạo token.
- `RegexFeaturizer`: Tạo vector từ input của người dùng sử dụng biểu thức chính quy
- `CRFEntityExtractor`: Thành phần này hiện thực trường điều kiện ngẫu nhiên (conditional random fields - CRF) để nhận diện thực thể.
- `EntitySynonymMapper`: Gán giá trị giống nhau cho các thực thể đồng nghĩa.
- `CountVectorsFeaturizer`: Xây dựng Mô hình túi từ (bag-of-words) cho input người dùng, ý định và phản hồi. Thành phần này hiện thực bằng `CountVectorizer` của `sklearn`.
- `EmbeddingIntentClassifier`: Được hiện thực dựa trên mạng Starspace, thành phần này được sử dụng để xác định ý định của input người dùng.

2.1.2 Rasa Core

Thay vì sử dụng một loạt các câu lệnh if/else, Rasa Core sử dụng mô hình học máy đã được huấn luyện để xác định hành động tiếp theo cần làm là gì sau khi nhận được dữ liệu từ Rasa NLU.

Để huấn luyện Rasa Core, cần có các thành phần sau:

- Stories:

  Rasa Core stories là một dạng dữ liệu huấn luyện cho mô hình quản lý hội thoại của Rasa. Một story đại diện cho một cuộc hội thoại giữa người dùng và chatbot, viết dưới một dạng đặc biệt, trong đó: input từ người dùng là intent và hành động ứng với input đó là action / utter.

  Các story huấn luyện được viết ở file `data/stories.md`. Dưới đây là một ví dụ cho story kiểm tra tài khoản:

  ```
  ## get balance 1
  * get_balance <!-- user ask for checking balance -->
    - utter_get_balance <!-- bot asks user to provide userAddress -->
  * give_userAddress <!-- users provide userAddress-->
    - utter_confirm <!-- bot confirms -->
    - action_get_balance <!-- bot gets user balance -->
  ```

- Domain:

  Domain xác định ý định, thực thể, slot, hành động mà chatbot cần biết. Đồng thời nó cũng bao gồm câu phản hồi (utterance) mà chatbot có thể trả lời cho người dùng.

  Các nội dung trong domain của chatbot được viết ở file `domain.yml`. Domain cho chatbot của chúng tôi gồm sáu phần:

  - `intents`: Liệt kê các ý định của người dùng đã được huấn luyện trong NLU.
  - `entities`: Liệt kê các thực thể đã được huấn luyện trong NLU.
  - `slots`: Chỗ để giữ các thông tin cần thiết xuyên suốt cuộc hội thoại, chúng tôi xác định tên và kiểu của chúng ở đây.
  - `responses`: Nội dung mà chatbot sẽ gửi đến người dùng. Ví dụ khi người dùng thông báo rằng họ muốn kiểm tra tài khoản, hành động tiếp theo của chatbot sẽ là `utter_get_balance`, nội dung của hành động đó được xác định rõ trong phần responses:

    ```
    utter_get_balance:
    - text: "Địa chỉ của bạn là gì?"
    ```

  - `actions`: Liệt kê các hành động phản hồi đã huấn luyện cho chatbot.
  - `forms`: Cấu trúc form dùng để định nghĩa story.

- Cấu hình cho Rasa Core:

  Cấu hình cho Rasa Core được viết ở file `config.yml`. Cấu hình cho chatbot của chúng tôi như sau:

```
policies:
  - name: MemoizationPolicy
    max_history: 5
  - name: TEDPolicy
    epochs: 100
    max_history: 5
  - name: MappingPolicy
  - name: FormPolicy
  - name: FallbackPolicy
    nlu_threshold: 0.65
    core_threshold: 0.7
    fallback_action_name: 'utter_default'
```

  - `MemoizationPolicy`: Ghi nhớ các cuộc hội thoại trong dữ liệu huấn luyện, sau đó dự đoán hành động tiếp theo cho input từ người dùng với confidence 1.0 nếu cuộc hội thoại đó đã được học, nếu không thì dự đoán None hoặc confidence 0.0.

    `max_history`: Số lượng cuộc hội thoại trước đó dùng để dự đoán hành động tiếp theo của chatbot.
  - `TEDPolicy`: Được hiện thực dựa trên kiến trúc Transformer, policy này dùng để tính toán độ tương tự giữa dialogue embedding và embedded system actions.

    `epochs`: số lượng epoch (mỗi epoch là một lần forward và backward propagation) để huấn luyện Core. `max_history`: Số lượng tối đa trạng thái của Tracker được sử dụng trong quá trình huấn luyện.
  - `MappingPolicy`: Sử dụng để ánh xạ trực tiếp ý định của người dùng và hành động tiếp theo của chatbot.
  - `FormPolicy`: Dùng để xử lý quá trình điền form. Khi FormAction được gọi, FormPolicy sẽ liên tục dự đoán FormAction cho đến khi tất cả các slot cần điền được cung cấp đầy đủ.
  - `FallbackPolicy`: Gọi hành động dự phòng/ mặc định (fallback action) nếu ít nhất một trong các trường hợp sau xảy ra:

    (1) Không có ý định nào được dự đoán bởi NLU có confidence cao hơn `nlu_threshold`.

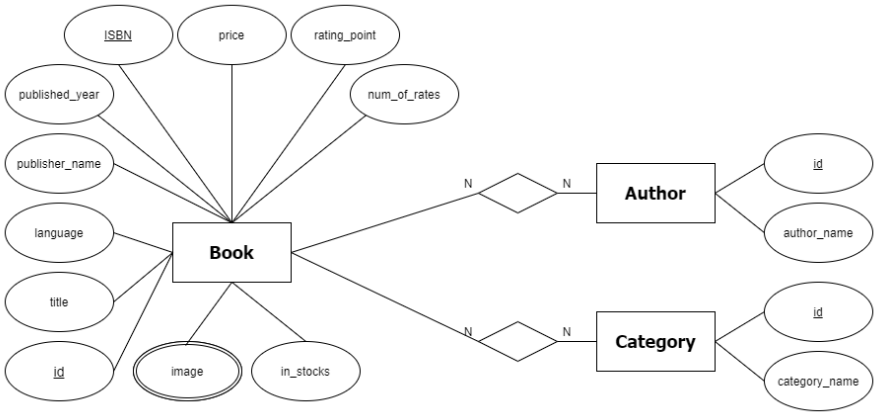
    (2) Không có hành động nào được dự đoán bởi Core có confidence cao hơn `core_threshold`.

2.2 Cơ sở dữ liệu (Cường)

Cơ sở dữ liệu chatbot sử dụng là một cơ sở dữ liệu truyền thống(traditional database-lưu trữ thông tin dạng văn bản và số) với hệ quản trị cơ sở dữ liệu PostgreSQL(hệ quản trị cơ sở dữ liệu quan hệ và hướng đối tượng).

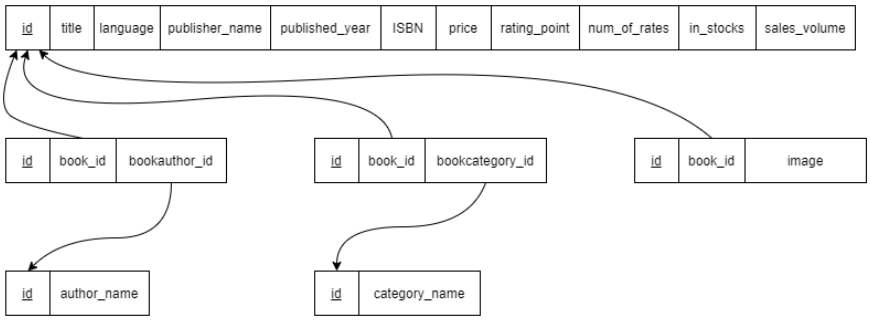
Cơ sở dữ liệu lưu trữ thông tin về: sách, tác giả, loại sách cũng như quan hệ giữa sách và tác giả và quan hệ giữa sách và loại sách.

Mô hình mối quan hệ thực thể:



Hình 5: Mô hình mối quan hệ thực thể

Lược đồ cơ sở dữ liệu:



Hình 6: Lược đồ cơ sở dữ liệu

2.3 Blockchain

Blockchain là một hệ thống cơ sở dữ liệu cho phép lưu trữ và truyền tải các khối thông tin (block). Và các khối này được liên kết với nhau nhờ mã hoá.

Hình 7: Blockchain

Các khối thông tin này hoạt động độc lập và được mở rộng theo thời gian. Chúng được quản lý bởi những người tham gia hệ thống chứ không phải qua bất cứ một đơn vị trung gian nào.

Những khối thông tin được nhắc đến ở đây chính là những cuộc giao dịch, trao đổi trong thực tế.

2.3.1 Hoạt động của Blockchain

Để một block - khối thông tin được thêm vào block chain thì phải có 4 yếu tố:

- Phải có giao dịch: nghĩa là có hoạt động mua bán diễn ra giữa 2 địa chỉ hợp lệ trên hệ thống.
- Giao dịch phải được xác minh: Các thông tin giao dịch như thời gian, địa điểm, người tham gia, số tiền ... đều phải được ghi lại.
- Giao dịch phải được lưu trữ trong block: để trở thành một giao dịch hợp lệ thì giao dịch này phải được lưu trữ trên block và có thể xem lại các thông tin của giao dịch.
- Block phải được nhận hash(hàm chuyển đổi một giá trị sang giá trị khác): Chỉ khi nào block nhận được hash thì một block mới có thể thêm vào blockchain.

Trong báo cáo này chúng tôi dựa vào những thông tin trên để thực hiện một giao dịch trên blockchain Ethereum. Chi tiết xem bên dưới

2.3.2 API server với Javascript

Đây là một server được chạy trên localhost của máy tính, được tạo ra nhằm: để chỉ với 2 phương thức là GET và POST người dùng có thể thực hiện được một giao dịch trên blockchain tương ứng.

Kiến trúc của server(Xem ở trong thư mục blockchain_wallet_api):

- app.js

  Đây là file chương trình chính. Mục tiêu của file này là định tuyến cho những tệp tin đến các mô-đun của những loại tiền tương ứng.Ví dụ như `\BTC`, `\BCH`, `\ETH`...

  - Các gói thông tin sẽ đến file app.js
  - Sau đó dựa vào URL (Network blockchain), các gói tin sẽ được chuyển tiếp đến những thư mục xử lí phù hợp với blockchain
  - Cuối cùng các gói tin dựa vào phương thức (GET, POST) và end-point API sẽ được chuyển đến các hàm xử lí phù hợp.

- Thư mục api. Trong thư mục này có chứa 3 thư mục nhỏ là controllers, middleware và routes.

  - Thư mục controller: Thư mục này có chứa 7 file javascipt file tương ứng với 7 network blockchain và những end-point API cũng được định nghĩa trong những file này. Có hai phương thức được dùng là GET và POST.

    Ví dụ:

```
const express = require("express");
const router = express.Router();
const controller = require('../controllers/ethereumControllers');

router.post("/transaction/send",controller.sendTransaction);

router.get("/transaction/fee", controller.getTransactionFee);

router.get("/transaction/info",controller.getTransaction);

router.get("/user/balance",controller.getUserBalance);

router.get("/user/address/generate",controller.getUserAddress);

module.exports = router;
```

– Thư mục middleware: Thư mục này dùng để xác minh người dùng trong các mạng blockchain qua phương thức login bởi username và password.
– thư mục routes: Cũng có 7 file javascript, các file này chứa những hàm xử lí cho end-point với phương thức tương ứng.
Ví dụ:

```
module.exports = {
sendTransaction : (req,res,next) => {

},
getTransaction : (req,res,next) => {

},
getTransactionFee : (req,res,next) => {

},
getUserBalance : (req,res,next) => {

},
 getUserAddress : (req,res,next) => {

},
}
```

- Thư mục config: chứa những biến môi trường để cài đặt cho server cũng như các url của các server blockchain (trong file url_blockchain.js).

2.3.3 Các API tương ứng với Python

Như đã nói ở trên, chúng tôi chỉ tập trung vào loại tiền ảo Ethereum và mạng blockchain của nó. Các loại tiền ảo khác sẽ được phát triển về sau.

Để phù hợp với Rasa, các actions phải sử dụng ngôn ngữ Python. Vì vậy để gọi các API tương ứng từ Server API phải có các hàm tương ứng trong ngôn ngữ Python. Theo như đặc tả ở trên, các hàm này sẽ được gọi theo phương thức GET và POST là các phương thức của HTTP. Đối với Python, thư viện request hỗ trợ điều này.

Hiện thực các hàm gọi API trong file API.py.

```
#API.py
import request
import base64

def authentication(user_name: str, pass_word: str, url : str):
    pass
```

```
def generateAddress(user_name: str, pass_word: str, url : str, coin : str):
    pass

def getTransaction(user_name: str, pass_word: str, transID :str , url : str, coin: str):
    pass

def getBalance(user_name: str, pass_word: str, address :str , url : str, coin: str):
    pass

def send(user_name: str, pass_word: str, adr_send :str, adr_receive: str,
private_key: str ,amount: float ,url : str, coin: str):
    pass
```

- url : là địa chỉ của API server.
- coin: là tên rút gọn của mạng blockchain giao dịch.
- Hàm authentication: Để xác thực người dùng nhờ vào user_name và pass_word.
- Hàm generateAddress : Để tạo một địa chỉ mới cho người dùng nếu chưa có.
- Hàm getTransaction: Lấy các thông tin từ transaction được cung cấp bởi transID.
- Hàm getBalance : Lấy thông tin về số dư trong địa chỉ (address) mà người dùng cung cấp.
- Hàm send: Đây là hàm dùng để thực hiện một giao dịch từ người dùng (adr_send) đến môt địa chỉ người nhận (adr_receive). Người dùng phả icung cấp khoá riêng tư (private_key), kèm theo đó là lượng tiền cần chuyển (amount), thì mới có thể thực hiện giao dịch.

3 Cài đặt hệ thống

3.1 Cài đặt cơ sở dữ liệu

- Cài đặt PostgresSQL

  Nếu chưa cài đặt PostgresSQL, bạn có thể tải về PostgresSQL tại đây

- Cài đặt cơ sở dữ liệu cho chatbot

  Bạn có thể sử dụng công cụ pgAdmin 4 để xây dựng cơ sở dữ liệu một cách trực quan hoặc cũng có thể sử dụng các lệnh sql trong thư mục create_database như sau:

  – Mở terminal vào thư mục **create_database**

  – Đăng nhập vào postgreSQL bằng cách nhập

  ```
  psql −U postgres
  ```

PostgreSQL sẽ yêu cầu nhập mật khẩu, mật khẩu mặc định là 'postgres'

- Tạo cơ sở dữ liệu mới với tên SampleDatabase

  ```
  \i createDb.sql
  ```

- Chỉ định cơ sở dữ liệu sử dụng

  ```
  \c sampledatabase
  ```

  Trong trường hợp không chỉ định database, thì khi thực hiện 1 lệnh sql trên database, postgresql sẽ thực hiện trên database mặc định là postgres

- Tạo bảng cho cơ sở dữ liệu mới tạo

  ```
  \i createTable.sql
  ```

- Thêm một vài giá trị ví dụ vào các bảng

  ```
  \i insertSampleValues.sql
  ```

  Bạn có thể tuỳ chỉnh các giá trị trong insertSampleValues.sql để thêm giá trị phù hợp

3.2 Cài đặt API Server cho ví Blockchain

- Cài đặt Nodejs và npm để chạy server
  - Cài đặt Nodejs.
    Hãy chắc chắn là trong máy đã cài đặt Nodejs phiên bản 10.0 hoặc cao hơn và npm.
    Để kiểm tra version của Nodejs và npm nhập lệnh:

    ```
    $ node −v
    $ npm −v
    ```

    Nếu chưa có, hãy truy cập vào trang web https://nodejs.org/ và tải phiên bản Nodejs tương ứng, sẽ có luôn cả npm khi cài đặt.
  - Cài đặt các dependencies tương ứng.
    Truy cập vào thư mục blockchain_wallet_api. Gõ lệnh:

    ```
    $ npm installl
    ```

    Giúp cài các dependencies còn thiếu cho server.
- Hướng dẫn sử dụng server. Để khởi động server, gõ lệnh:

  ```
  $ npm start
  ```

  Server được mở tại 127.0.0.1:3003. Có thể xem tại website http://localhost:3003.

  Nhấn tổ hợp phím 'Ctrl + S' , khi cần update server.

  Nhấn tổ hợp phím 'Ctrl + C', để tắt server.

3.3 Cài đặt Rasa

- Cài đặt môi trường lập trình Python với một số hệ điều hành thông dụng sau:
  - Ubuntu

    ```
    $ sudo apt update
    $ sudo apt install python3-dev python3-pip
    ```

  - MacOS

    ```
    $ brew update
    $ brew install python
    ```

  - Windows

    ```
    C:\> pip3 install -U pip
    ```

- Cài đặt mã nguồn mở Rasa (Ubuntu, MacOS và Windows)
  - Đầu tiên chắc chắn rằng bạn có phiên bản mới nhất của pip

    ```
    $ pip install -U pip
    ```

  - Tiếp theo, cài đặt Rasa

    ```
    $ pip install rasa
    ```

- Để sử dụng, ta cần huấn luyện cho mô hình

  ```
  $ rasa train
  ```

- Sau khi huấn luyện xong, chạy actions server

  ```
  $ rasa run actions
  ```

- Và cuối cùng, có thể bắt đầu nói chuyện với chatbot

  ```
  $ rasa shell
  ```

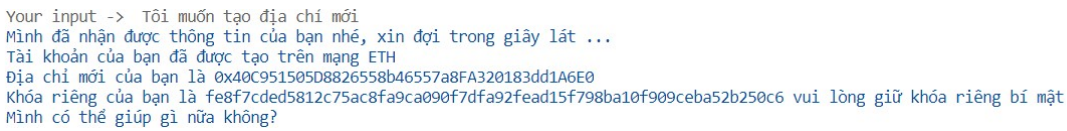
4 Kết quả

4.1 Đối thoại cơ bản

```
Your input ->  xin chào
Xin chào
Your input ->  bạn là ai
Mình là bot bán hàng!
Mình có thể tìm kiếm cuốn sách tốt nhất cho bạn, chỉ cần bạn nhập câu hỏi!
Your input ->  cảm ơn
Cảm ơn bạn nhìu lắm nè!
Your input ->  tạm biệt
Tạm biệt và hẹn gặp lại bạn!
```

Hình 8: Đối thoại cơ bản với chatbot

4.2 Tạo địa chỉ mới



Hình 9: Tạo địa chỉ mới với chatbot

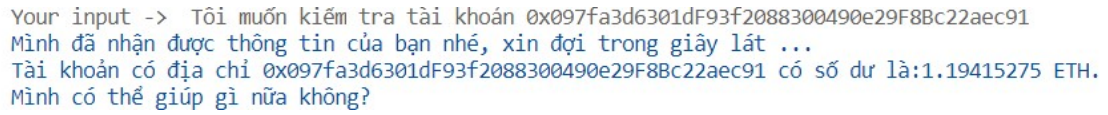
4.3 Kiểm tra tài khoản

Có thể kiểm tra tài khoản qua một trong ba kịch bản sau.

- Cách 1

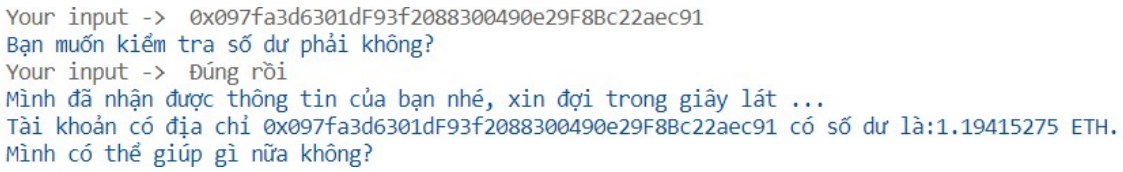
Hình 10: Kiểm tra tài khoản với chatbot (cách 1)

- Cách 2



Hình 11: Kiểm tra tài khoản với chatbot (cách 2)

- Cách 3



Hình 12: Kiểm tra tài khoản với chatbot (cách 3)

4.4 Kiểm tra thông tin giao dịch

Có thể kiểm tra thông tin giao dịch qua một trong ba kịch bản sau.

- Cách 1

```
Your input ->  Tôi muốn kiểm tra thông tin giao dịch 0x03559ceea68feb19bca79c3a8f037cd84d9165d10fe4062ca87257f004e48a9c
Mình đã nhận được thông tin của bạn nhé, xin đợi trong giây lát ...
Thông tin giao dịch với ID 0x03559ceea68feb19bca79c3a8f037cd84d9165d10fe4062ca87257f004e48a9c là:
Địa chỉ của người gửi là 0x81b7e08f65bdf5648606c89998a9cc8164397647
Địa chỉ của người nhận là 0xb2faf2c2089f512027f4a88ad683f0290da32a06
Số tiền giao dịch là 1ETH
Mình có thể giúp gì nữa không?
```

Hình 13: Kiểm tra thông tin giao dịch với chatbot (cách 1)

- Cách 2

```
Your input ->  Kiểm tra thông tin giao dịch giúp tôi
Xin vui lòng cung cấp số giao dịch?
Your input ->  đây 0x03559ceea68feb19bca79c3a8f037cd84d9165d10fe4062ca87257f004e48a9c
Mình đã nhận được thông tin của bạn nhé, xin đợi trong giây lát ...
Thông tin giao dịch với ID 0x03559ceea68feb19bca79c3a8f037cd84d9165d10fe4062ca87257f004e48a9c là:
Địa chỉ của người gửi là 0x81b7e08f65bdf5648606c89998a9cc8164397647
Địa chỉ của người nhận là 0xb2faf2c2089f512027f4a88ad683f0290da32a06
Số tiền giao dịch là 1ETH
Mình có thể giúp gì nữa không?
```

Hình 14: Kiểm tra thông tin giao dịch với chatbot (cách 2)

- Cách 3

```
Your input ->  0x03559ceea68feb19bca79c3a8f037cd84d9165d10fe4062ca87257f004e48a9c
Bạn muốn kiểm tra thông tin giao dịch phải không?
Your input ->  đúng rồi
Mình đã nhận được thông tin của bạn nhé, xin đợi trong giây lát ...
Thông tin giao dịch với ID 0x03559ceea68feb19bca79c3a8f037cd84d9165d10fe4062ca87257f004e48a9c là:
Địa chỉ của người gửi là 0x81b7e08f65bdf5648606c89998a9cc8164397647
Địa chỉ của người nhận là 0xb2faf2c2089f512027f4a88ad683f0290da32a06
Số tiền giao dịch là 1ETH
Mình có thể giúp gì nữa không?
```

Hình 15: Kiểm tra thông tin giao dịch với chatbot (cách 3)

4.5 Thực hiện giao dịch

```
Your input ->  Tôi muốn chuyển khoản
Địa chỉ của người gửi là gì? (Bạn vui lòng ghi theo cú pháp 'se:<địa chỉ người gửi>' nhé!)
Your input ->  se:0x097fa3d6301dF93f2088300490e29F8Bc22aec91
Địa chỉ của người nhận là gì? (Bạn vui lòng ghi theo cú pháp 'de:<địa chỉ người nhận>' nhé!)
Your input ->  de:0x9fd32A78Cc1Aa71CBe2aF06e47e3F6D0e9951b5F
Private key của bạn là gì?
Your input ->  aa96f903e71f0204cd45b4298428760a1a0d70165059aab0abcbc7ee003cc3f0
Bạn muốn gửi bao nhiêu? (Bạn vui lòng ghi theo cú pháp 'am:<amount>' nhé!)
Your input ->  am:0.01
Mình đã nhận được thông tin của bạn, giao dịch sẽ được hoàn tất trong chốc lát :)
Giao dịch với thông tin:
Địa chỉ của người gửi là 0x097fa3d6301dF93f2088300490e29F8Bc22aec91
Địa chỉ của người nhận là 0x9fd32A78Cc1Aa71CBe2aF06e47e3F6D0e9951b5F
Số tiền giao dịch là 0.01ETH
Giao dịch thành công. Mã giao dịch là 0x6a5af6b578d60b88ee37b4ffe57733fb96d8aa77a51d350ba3a97a3d27f37bb6
```

Hình 16: Thực hiện giao dịch với chatbot

4.6 Tìm sách theo tên

```
Your input ->  Tìm quyển Artificial Intelligence
Mình đã nhận được thông tin của bạn nhé, xin đợi trong giây lát ...
Tìm thấy 2 cuốn sách:
--------------------------------
Title          : Artificial Intelligence
Language       : English
Publisher name : Prentice Hall
Published year : 2009
ISBN           : 9780714874746
Price          : 8.0
Rating point   : 4.0
Num of rates   : 1
In stocks      : 5
Sales volume   : 4
Author         : Stuart Russell
               : Peter Norvig
Category       : AI
               : ML
--------------------------------
Title          : Artificial Intelligence
Language       : English
Publisher name : Prentice Hall
Published year : 2020
ISBN           : 9781541768130
Price          : 6.0
Rating point   : 0.0
Num of rates   : 0
In stocks      : 2
Sales volume   : 4
Author         : Stuart Russell
Category       : AI
--------------------------------
Bạn hãy cho mình biết số ISBN của quyển sách bạn muốn mua nhé!
```

Hình 17: Tìm sách với chatbot

4.7 Mua sách với ISBN

```
Your input ->  Tôi muốn mua quyển sách có ISBN 9781541768130
Mình đã nhận được thông tin của bạn nhé, xin đợi trong giây lát ...
Thông tin về quyển sách Artificial Intelligence:
ISBN: 9781541768130
Author: Stuart Russell
Price: 6.0 BTC
Admin Address ID: 0x9fd32A78Cc1Aa71CBe2aF06e47e3F6D0e9951b5F
Vui lòng thực hiện chuyển khoản cho Admin để mua sách bạn nhé!
Your input ->  chuyển khoản
Địa chỉ của người gửi là gì? (Bạn vui lòng ghi theo cú pháp 'se:<địa chỉ người gửi>' nhé!)
Your input ->  se:0x097fa3d6301dF93f2088300490e29F8Bc22aec91
Địa chỉ của người nhận là gì? (Bạn vui lòng ghi theo cú pháp 'de:<địa chỉ người nhận>' nhé!)
Your input ->  de:0x9fd32A78Cc1Aa71CBe2aF06e47e3F6D0e9951b5F
Private key của bạn là gì?
Your input ->  aa96f903e71f0204cd45b4298428760a1a0d70165059aab0abcbc7ee003cc3f0
Bạn muốn gửi bao nhiêu? (Bạn vui lòng ghi theo cú pháp 'am:<amount>' nhé!)
Your input ->  am:0.06
Mình đã nhận được thông tin của bạn, giao dịch sẽ được hoàn tất trong chốc lát :)
Giao dịch với thông tin:
Địa chỉ của người gửi là 0x097fa3d6301dF93f2088300490e29F8Bc22aec91
Địa chỉ của người nhận là 0x9fd32A78Cc1Aa71CBe2aF06e47e3F6D0e9951b5F
Số tiền giao dịch là 0.06ETH
Giao dịch thành công. Mã giao dịch là 0x1673a0863fec9b4c065c65eca8a6782bd7728f6a95278dd4ff334df36b2b42dc
Key book bạn đã mua: fbcd687b-2924-4d5d-a742-4508aec8db55
```

Hình 18: Mua sách với chatbot

5 Nhận xét

5.1 Dự đoán input và sửa lỗi chính tả

5.2 Database

5.2.1 Ưu điểm của PostgreSQL và thư viện psycopg2

- PostgreSQL cho phép xây dựng các mô hình dữ liệu quan hệ theo dạng bảng nên trực quan và dễ hiểu với hầu hết người dùng.
- psycopg2 là thư viện hỗ trợ giúp các dự án sử dụng python kết nối với cơ sở dữ liệu bằng các câu truy vấn sql nên cũng dễ cài đặt và sử dụng

5.2.2 Nhược điểm của PostgreSQL và thư viện psycopg2

- PostgreSQL sử dụng mô hình dữ liệu quan hệ nên tốc độ truy vấn chậm hơn so với các mô hình dữ liệu phi quan hệ. Điều này làm tốc độ phản hồi của chatbot cũng chậm hơn
- khi sử dụng psycopg2 cần xây dựng các câu truy vấn vào cơ sở dữ liệu. Khi lược đồ cơ sở dữ liệu thay đổi,ta cần điều chỉnh lại các câu truy vấn và mã nguồn cho phù hợp.

5.3 Blockchain

5.3.1 Ưu điểm của Server API

- Việc tạo ra một server rất hữu ích khi cần cập nhật hay sửa chữa.
- Server API được tạo ra giúp dễ sử dụng và dễ kiểm soát lỗi nhờ việc chỉ dùng hai phương thức GET và POST.

5.3.2 Nhược điểm của Server API

- Các blockchain thường cập nhật thường xuyên dẫn đến việc server cũng phải liên tục cập nhật mới.
- Cần phải tăng cường bảo mật cho người dùng đối với server API để không bị lộ các thông tin.
- Chưa có một cơ chế chung cho tất cả các loại tiền ảo.

5.4 Rasa

5.4.1 Ưu điểm của Rasa

- Rasa NLU xác định ý định và thực thể rất tốt với rất ít dữ liệu huấn luyện. Đồng thời các thành phần trong đường ống xử lý cũng dễ dàng tùy biến để phù hợp với nhu cầu.
- Có thể sử dụng các mô hình học máy phức tạp một cách đơn giản.

5.4.2 Nhược điểm của Rasa

- Rất khó để xác định ngữ cảnh khi cần trích xuất nhiều thực thể có cấu trúc giống nhau. Chẳng hạn trong chatbot của chúng tôi, `senderAddress` và `destAddress` có cấu trúc giống nhau và cùng xuất hiện trong ngữ cảnh thực hiện giao dịch chuyển khoản, vì vậy nên cần phải thêm tiếp đầu ngữ "se:" và "de:" để chatbot xác định chính xác thực thể.