THE UNIVERSITY OF HONG KONG

DEPARTMENT OF COMPUTER SCIENCE

Logo

Description automatically generated

**FITE 3012: E-PAYMENT AND CRYPTO-CURRENCY**

**PROGRAMMING ASSIGNMENT**

**READ ME FILE**

**Varun Singh**: 3035452898 (author)

**TABLE OF CONTENTS**

**Content:**

1. [Installation](#_Installation:)
   1. [Smart contract deployment](#_Smart_Contract_Deployment)
   2. [Stripe setup](#_Stripe_setup)
   3. [Flask webapp installation](#_Flask_webapp_installation)
2. [Project Details](#_Project_Details)
   1. [Smart Contract](#_Smart_Contract)
   2. [Flask WebApp](#_Flask_Webapp)
   3. [Stripe Integration](#_Stripe_Integration)
   4. [ERC-20 Coin Transfer](#_ERC_20_Coin)
3. [Business Model](#_Business_Model)
4. [Limitations and Backlog](#_Limitations_and_Backlog)
5. [User Stories](#_User_Stories)

# **Installation:**

This section explains the process of setting up the environment for the flask webapp and guides one through the process of smart contract deployment using truffle and ganache.

## **Smart Contract Deployment**

**Step 1:**

Download and install NPM using the following link:

[Windows: Windows Installer (.msi)]

[MacOS: macOS Installer (.pkg)]

<https://nodejs.org/en/download/>

**Step 2:**

Download MetaMask plugin in your Chrome or Firefox browser using the following link (you need to have chrome or Firefox web browser installed for this step):

<https://metamask.io/download.html>

**Step 3:**

Install truffle using npm

$ npm install truffle -g

**Step 4:**

Download and install ganache using the following link:

<https://www.trufflesuite.com/ganache>

**Step 5:**

Install npm after downloading the submission file in the following directory:

**singh-assignment -> tutorial-token**

$ npm install

**Step 6:**

Compile contract using truffle

$ truffle compile

**Step 7:**

Setup Ganache

1. Create work-stream

Graphical user interface

Description automatically generated

1. Give a Name then click on add project

Graphical user interface, text, application, email

Description automatically generated

1. Select file

[after clicking on add project navigate to **singh-assignment -> tutorial-token** directory and select **truffle.js**]

Graphical user interface, application

Description automatically generated

1. Save workplace

Graphical user interface, text, application, email

Description automatically generated

1. Run truffle tests

$ truffle test

1. Deploy using truffle

$ truffle deploy

Key

### Output (Export Path and Private Key)

Ganache Export Path

A picture containing graphical user interface

Description automatically generated

### Output (Contract Address)

[Ganache -> Contracts]

Ganache Contract Address

Graphical user interface, text, email, website

Description automatically generated

**Step 8:**

Setup MetaMask

1. Create custom network [click network name to see menu and select custom RPC]

Graphical user interface, text

Description automatically generated

1. Fill network name, [URL (ganache export path)](#_Output_(Export_Path)) and Chain ID (use 0x5b)

Graphical user interface, text, application, email

Description automatically generated

1. Import account [click on account image to see menu]

Graphical user interface, application

Description automatically generated

1. Fill [private key](#_Output_(Export_Path) of ganache account

Graphical user interface, application

Description automatically generated

1. Add Token in new imported account

Graphical user interface, text, application, email

Description automatically generated

1. Chose custom token and input [ganache contract address](#_Output_(Contract_Address)) [click on next -> add token]

Graphical user interface, text, application, email

Description automatically generated

### Output (metamask account address) [CET token visible in metamask]

Graphical user interface, text, application, email

Description automatically generated

## **Stripe setup**

In case you want to test using my stripe credentials, please use the following or else the documentation covers steps for creating a stripe account and using personal API keys:

|  |  |
| --- | --- |
| Email | [singhvarun1999@gmail.com](mailto:singhvarun1999@gmail.com) |
| Password | FiteProgramming |
| Publishable Key | pk\_test\_51HpR84IdoMdf5WR8cgKJqlkBkkYttwLupknNgCZu3SziMGmAPLtgfewdwxq41i02bj4QJEaz2fjPL0WUapmb8qTw00n7I5bMqW |
| Secret Key | sk\_test\_51HpR84IdoMdf5WR8irfUY6Klvf1StDEducP8vvuDcusESqcx99n0hlLLDai3klQ8EPgOPixg8I2QU1rnB6GjnYcB00QLocneWw |

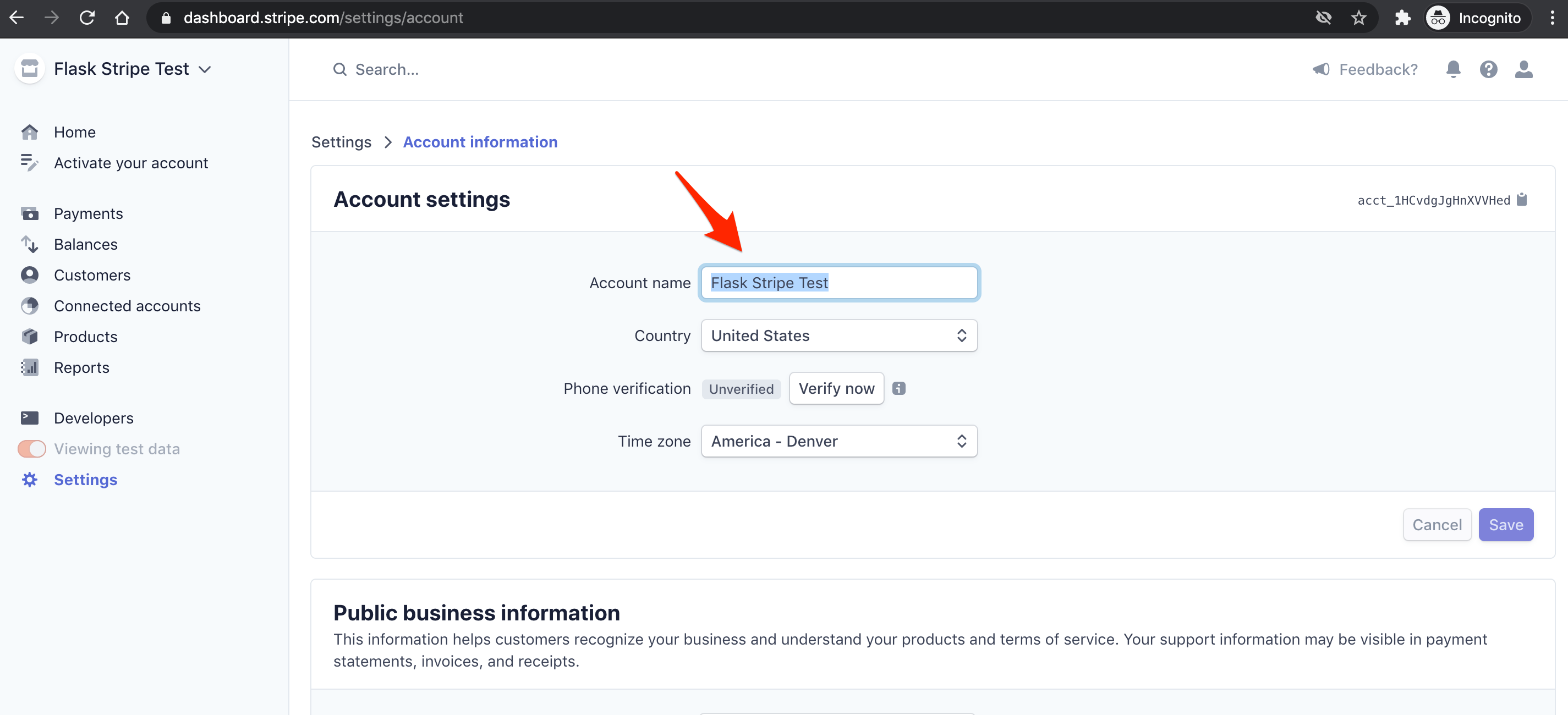
**Step 1:**

Register for a stripe account using the following link:

<https://dashboard.stripe.com/register>

**Step 2:**

Set Account Name



**Step 3:**

### API keys [Navigate to **Developers -> API keys**]

Graphical user interface, text, application, email

Description automatically generated

## **Flask webapp installation**

**Step 1:**

Create and activate a python virtual environment after downloading the submission file in the following directory: **singh-assignment -> flask-stripe-checkout -> env**

$ python3 -m venv venv && source venv/bin/activate

**Step 2:**

Install requirement.txt

(venv)$ pip3 install -r requirements.txt

**Step 3:**

Add stripe [publishable key](#_API_keys_[Navigate) and [secret key](#_API_keys_[Navigate) as environment variables

(env)$ export STRIPE\_PUBLISHABLE\_KEY=<YOUR\_STRIPE\_PUBLISHABLE\_KEY>

(env)$ export STRIPE\_SECRET\_KEY=<YOUR\_STRIPE\_SECRET\_KEY>

(env)$ export STRIPE\_ENDPOINT\_SECRET=<YOUR\_STRIPE\_SECRET\_KEY>

**Step 4:**

Add [ganache export path](#_Output_(Export_Path)), [ganache contract address](#_Output_(Contract_Address)) and [metamask account address](#_Output_(metamask_account)

(env)$ export HTTP=<GANACHE\_EXPORT\_PATH>

(env)$ export CONTRACT\_ADDRESS=<GANACHE\_CONTRACT\_ADDRESS>

(env)$ export SENDERS\_ADDRESS=<METAMASK\_ACCOUNT\_ADDRESS>

**Step 5:**

Run the server using the following command:

(env)$ FLASK\_ENV=development python app.py

**Step 6:**

Open [http://localhost:5000](http://localhost:5000/) in the browser

# **Project Details**

This section provides detail about the methods and programming used to build the smart contract, flask webapp, stripe gateway and ERC-20 coin transfer.

## **Smart Contract**

The smart contract is written in solidity programming language using the object oriented programming technique. I have used truffle to create a project on my local mac os machine. Truffle automatically creates the project structure and allows the user to add smart contracts in the contract folder. My smart contract is called CetaToken.sol which includes a constructor and 6 functions. The constructor is where the name, symbol, number of decimals and total initial supply of the coin is defined. The first function is transferableTokens returns the balance of token in an account. The second function transfer, allows the transfer of tokens from parent account to another account and updates the balances of both the accounts. The third function, transferFrom allows a user to transfer tokens one account to another account and updates the balances of both the accounts involved. The fourth function allowance returns the remaining token in an account. The fifth function approve returns the an approval message when the token transfer is successful. The last function balanceOf returns the balance of the token holder.

## **Flask Webapp**

Flask webapps are python based applications that can integrate both backend and frontend functionalities in one place. I have used JavaScript and python for backend and HTML, bootstrap and CSS for the frontend of the web application. The user inputs data into the HTML form. JavaScript is used to connect to the stripe gateway. It also collects the data from the HTML form and parses it onto the python sessions. Python manages sessions in the webapp and loads html pages and transfers ERC-20 depending on which session is called by the java script code.

## **Stripe Integration**

I have chosen the option of transferring the user from my webapp to stripe’s payment gateway. This is the process I have prefered over transferring user credit information through rest API, as stripe’s gateway has better security and collecting personal account information on my webapp might lead to a potential security breach. As mentioned in section 2b, JavaScript collects the information from the HTML form and connects the webapp to stripe gateway.

## **ERC 20 Coin Transfer**

After passing the credit card information on the stripe gateway, stripe either approves or disapproves the transaction. Approval and disapproval are two unique sessions in python. When cancelled session is called, a html page is rendered along with a message to retry. In the case when approve function is called, and html page is rendered that asks the user to check their metamask account and the transfer of erc-20 token takes place. To transfer the ERC-20, I have used python web3 library to make the transaction.

**PYTHON**

Manages:

Check out session to stripe gateway

Success session and erc-20 transfer

Cancelled session

**JAVA SCRIPT**

Initiates the onclick of purchase button and calls a checkout session

**HTML FRONTEND**

**[CSS, Bootstrap]**

Collects:

amount of CETA coins to buy and users Ethereum address

# **Business Model**

The idea behind CETA coin is to use it during events. CETA as a brand would sell party supplies and organize events across universities in Hong Kong. In order to attend a party or to guy ceta merchandise a person needs to go to the CETA ecommerce store to purchase merchandise or passes for CETA events. All transaction on the CETA ecommerce store are done through CETA coins only. This implementation helps in streamlining the process of purchasing drinks or food (through ceta coins, preferably through QR code scanning during events for each product that is on sale) during ceta events and makes accounting easier in case of purchases on the ecommerce platform.

# **Limitations and Backlog**

* Currently when a user is buying a CETA coin, a session is created and the amount of ceta coins to be bought and the users Ethereum address are being stored in a json file during the checkout session. In the success session, this information is being read by opening the file for every unique session. To solve this issue and make multiple sessions possible at one time, I can store this data in a database during checkout and access it when the success session is called.
* Due to time constraints the ecommerce website is not completely developed as I was facing problems in automating the process of opening user’s metamask, when the user clicks on the ‘buy’ button.

|  |  |
| --- | --- |
| Backlog in case of a possible Sprint 2 | |
| Task | **Description** |
| Database implementation | Create a database to store unique session id, user’s Ethereum address and amount of CETA coins that the user would like to purchase |
| Ecommerce app development | Complete the development of the ecommerce website along with the integration of metamask for optimal UX |

# **User Stories**

|  |  |
| --- | --- |
| User Stories | |
| User Story | **Confirmation** |
| User can access the homepage of the webapp with basic details about the coin | The homepage of the webapp is visible upon hosting the webapp on a local server |
| User can input the number of CETA coins that they want to buy | HTML form created that receives the amount of CETA coins to be bought |
| User can input the Ethereum address that they want the CETA coin to be transferred to | HTML form created that received users Ethereum account address |
| User can click a button and be redirected to Stripe check out | Purchase button on the homepage redirects the user to stripe payment checkout website |