**Academic E-Certificate Verification System using Algorand Blockchain**

**Abstract:**In the rapidly advancing digital landscape, nearly everything has undergone a transformation into the digital realm. Educational institutions have followed suit by digitalizing various certificates, including SSLC, HSC, and academic certificates, and making them accessible to students. However, students often struggle with the cumbersome task of maintaining their physical degree certificates. Moreover, organizations and institutions face significant challenges when it comes to verifying and validating these certificates.

To address these issues, our project aims to leverage the potential of blockchain technology to issue E-certificates securely. The process begins by creating E-certificates on a blockchain platform, incorporating essential student details. As a result, a unique transaction hash is generated for each certificate. This transaction hash serves as a key that students can utilize to access their E-certificates, and it can be readily shared with any interested party for verification purposes.

By harnessing the power of blockchain technology, our solution provides a significantly enhanced level of security and efficiency in the validation of digital E-certificates. This innovative approach ensures the immutability and tamper-resistant nature of the certificates, minimizing the risk of fraud or unauthorized alterations. With the implementation of blockchain, the process of issuing, accessing, and verifying E-certificates becomes streamlined, transparent, and reliable, benefiting both students and educational institutions alike.

**Application Context:**

The application is a decentralized application (Dapp) developed using Python and the Flask web framework. Its primary purpose is to interact with the Algorand blockchain, offering various functionalities within the context of a university environment.

The application features a login system that allows users to access the Dapp using a mnemonic. This login process ensures secure and authorized access to the application's functionality. Once logged in, users, who are affiliated with a university, can upload their academic E-certificates onto the Algorand blockchain.

The integration with the Algorand blockchain provides several benefits. Firstly, it ensures the immutability and integrity of the E-certificates, as the blockchain's decentralized nature makes it highly resistant to tampering or unauthorized alterations. Secondly, the use of Algorand blockchain allows for easy retrieval of the E-certificates. Users can access their E-certificates by providing the unique E-Certificate ID assigned by their respective universities.

To facilitate the development of the web application, the Flask library is utilized. Flask provides a robust and flexible framework for building web applications, enabling the implementation of various features and functionalities. The `algosdk` library is employed to establish communication with the Algorand blockchain, facilitating the uploading and retrieval of E-certificates. The `json` library is used for serializing and deserializing JSON data, ensuring seamless data exchange between the application and the blockchain. Additionally, the `base64` library is utilized for encoding and decoding base64 data, which may be necessary for certain operations within the application. The `flask\_cors` library enables Cross-Origin Resource Sharing (CORS), allowing the application to handle requests from different domains and ensuring proper data exchange between the Dapp and external resources.

Overall, this application leverages the Python programming language, the Flask web framework, and the Algorand blockchain to create a secure and efficient system for uploading and verifying academic E-certificates. By utilizing blockchain technology, the application enhances the transparency, security, and trustworthiness of the certification process within the university context.

The application has the following endpoints:

* **/:** The home page route. It renders an HTML template that contains a login form and links to other pages.
* **/uc:** The upload E-Certificate page route. It renders an HTML template that contains a form for uploading a E-Certificate.
* **/login:** The login route. It receives a POST request with a mnemonic from a form, derives the private key from the mnemonic, and returns a JSON response with a status indicating whether the mnemonic is valid and owned by a university.
* **/logout:** The logout route. It receives a GET request and removes the user session from the Flask session object.
* **/auth:** The authentication route. It receives a GET request and returns a JSON response with a status indicating whether the user is authenticated.
* **/upload\_certificate:** The upload E-Certificate route. It receives a POST request with a E-Certificate data from a form, signs the E-Certificate data with the user's private key, and submits the signed transaction to the Algorand network. It then returns a JSON response with a status indicating whether the transaction was successful and the transaction ID and block number of the transaction.
* **/get\_certificate:** The get E-Certificate route. It receives a POST request with a transaction ID and searches the Algorand blockchain for a transaction with the specified ID. If the transaction is found and it contains a valid E-Certificate, it returns a JSON response with a status indicating that the E-Certificate was found and the E-Certificate data. Otherwise, it returns a JSON response with a status indicating that the E-Certificate was not found.

**How Python Micro Web Frame Work – Flask useful For This Application ?**

A Python web application framework like Flask is a powerful tool that enables developers to build web applications efficiently and effectively using the Python programming language. Flask specifically is a lightweight and flexible framework that follows the Model-View-Controller (MVC) architectural pattern, making it popular among developers for its simplicity and ease of use.

At its core, Flask provides the necessary infrastructure and functionality for handling web requests and responses, managing routing, and rendering dynamic HTML templates. It allows developers to define routes, which are URLs that map to specific functions, called view functions, in the application. These view functions handle incoming requests, perform necessary actions, and generate appropriate responses to be sent back to the client.

One of the key features of Flask is its simplicity and minimalistic approach. It provides just the essential components required for building web applications, allowing developers the freedom to choose and integrate additional libraries and tools based on their specific needs. This lightweight nature makes Flask a great choice for smaller projects or when simplicity and flexibility are desired.

Flask also supports the use of various extensions, which are third-party libraries that provide additional functionality to the framework. These extensions cover a wide range of features such as form validation, database integration, user authentication, and more. These extensions greatly enhance Flask's capabilities and enable developers to build complex and feature-rich web applications without reinventing the wheel.

Furthermore, Flask promotes the use of Jinja2 templating engine, which provides a powerful and intuitive way to generate HTML templates dynamically. This allows developers to separate the presentation logic from the business logic, making the code more maintainable and readable.

In conclusion, a Python web application framework like Flask is a versatile and lightweight tool that simplifies the development of web applications using Python. It provides the necessary infrastructure, routing mechanisms, and template rendering capabilities, allowing developers to focus on building the application's logic and functionality. With its minimalistic design and extensibility, Flask offers developers the flexibility to create tailored web applications suited to their specific requirements.

**Why This Application Needs The Use Of Algorand Blockchain Instead of Regular DB ?**

Algorand blockchain can provide several benefits over a regular database when it comes to academic E-Certificate verification, including:

1. **Decentralization:** Algorand is a decentralized blockchain, which means that there is no single point of failure. This ensures that the verification process cannot be manipulated or tampered with by any single entity, making it more secure and trustworthy.
2. **Immutability:** Once a E-Certificate is verified and stored on the Algorand blockchain, it cannot be modified or deleted. This ensures that the integrity of the E-Certificate remains intact and cannot be tampered with, providing a more reliable and transparent verification process.
3. **Transparency:** Algorand provides transparency by allowing anyone to access the blockchain and verify the E-Certificates. This eliminates the need for a centralized authority to verify the E-Certificates, making the verification process more efficient and cost-effective.
4. **Speed:** Algorand is designed to process transactions quickly and efficiently, making it ideal for verifying large volumes of E-Certificates in a short period of time.
5. **Cost-effectiveness:** Algorand offers a cost-effective solution for E-Certificate verification compared to traditional methods, such as relying on third-party verification services.

Overall, using Algorand blockchain for academic E-Certificate verification can provide a secure, transparent, efficient, and cost-effective way to verify E-Certificates while ensuring their integrity and immutability.

**Algorand Facilities Used By The Application :**

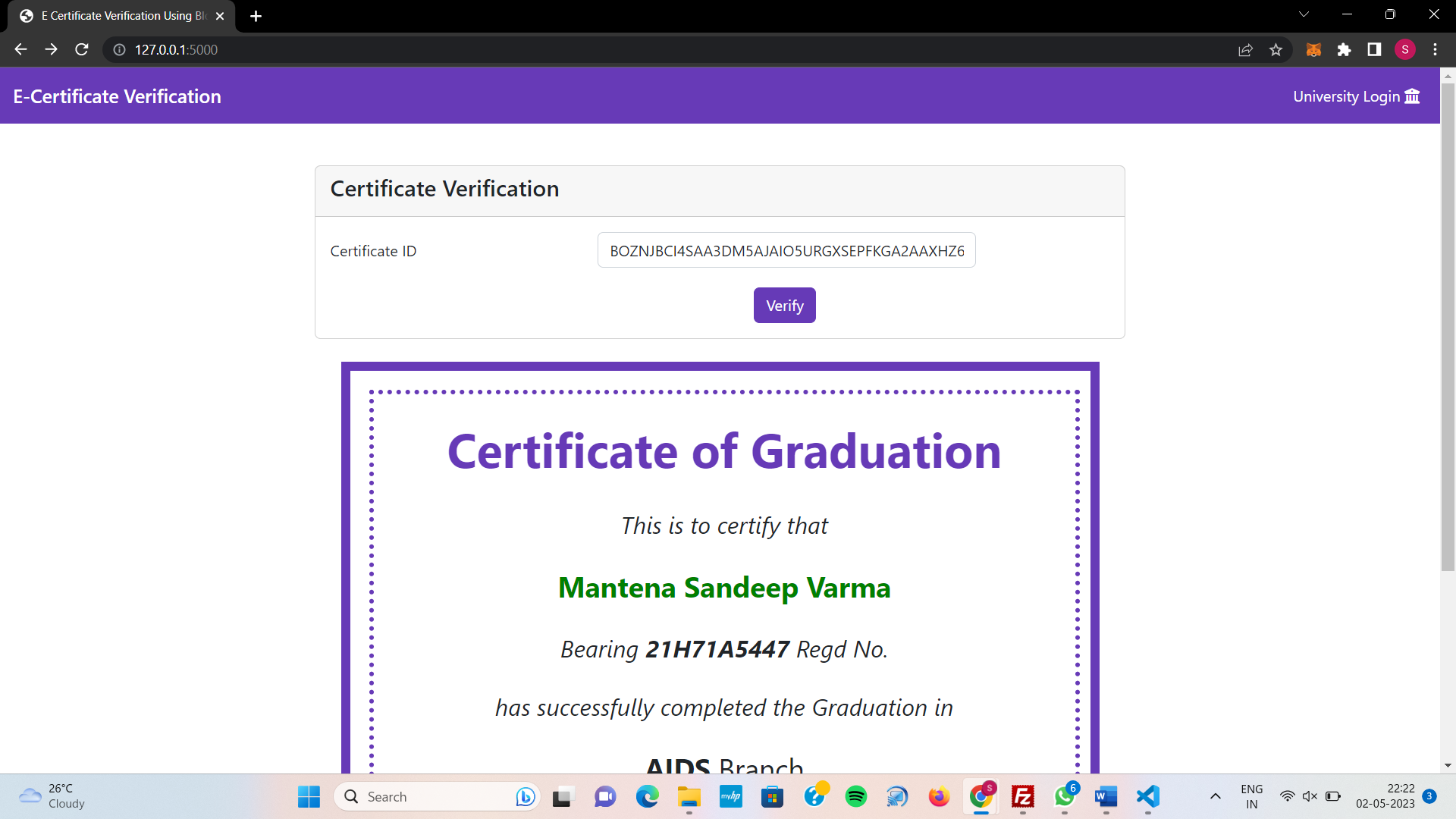
1. Algorand Account : An Algorand Account For the University is used for Uploading the E-Certificates By Making The Transactions with E-Certificate Data to the Algorand Blockchain.
2. PureStake API : An API Key is used in the Application Which is used for Interacting With The Algorand Blockchain.
3. Signatures : While Uploading The E-Certificates to the Blockchain, The Roll No of The Student is Signed With The private key of the university and the signature is also uploaded to the blockchain along with the E-Certificate. This Signature Will Be Verified When The E-Certificate is retrieved from the Blockchain, To Check the Origin of the E-Certificate.
4. AlgoSDK : An Python Development Kit For Building Applications on Algorand Blockchain which contains various Classes & Utility Functions regarding Transactions, Cryptography, Signatures etc. Some of The Functions used in this Application are :

* **AlgodClient**: The AlgodClient is used to create a client that communicates with an Algorand node. It is used to get suggested transaction parameters and send transactions to the Algorand network.
* **IndexerClient**: The IndexerClient is used to create a client that communicates with an Algorand indexer node. It is used to search transactions and retrieve information about them.
* **util.sign\_bytes**: This function is used to sign a byte string with a private key.
* **transaction.PaymentTxn : This class is used to create a payment transaction.**
* **txn.sign:** This method is used to sign a transaction with a private key.
* **algod\_client.send\_transaction:** This method is used to send a signed transaction to the Algorand network.
* **algod\_client.pending\_transaction\_info:** This method is used to get information about a pending transaction, such as the confirmed round.
* **mnemonic.to\_private\_key:** This function is used to derive a private key from a mnemonic.
* **account.address\_from\_private\_key:** This function is used to get the public address corresponding to a private key.

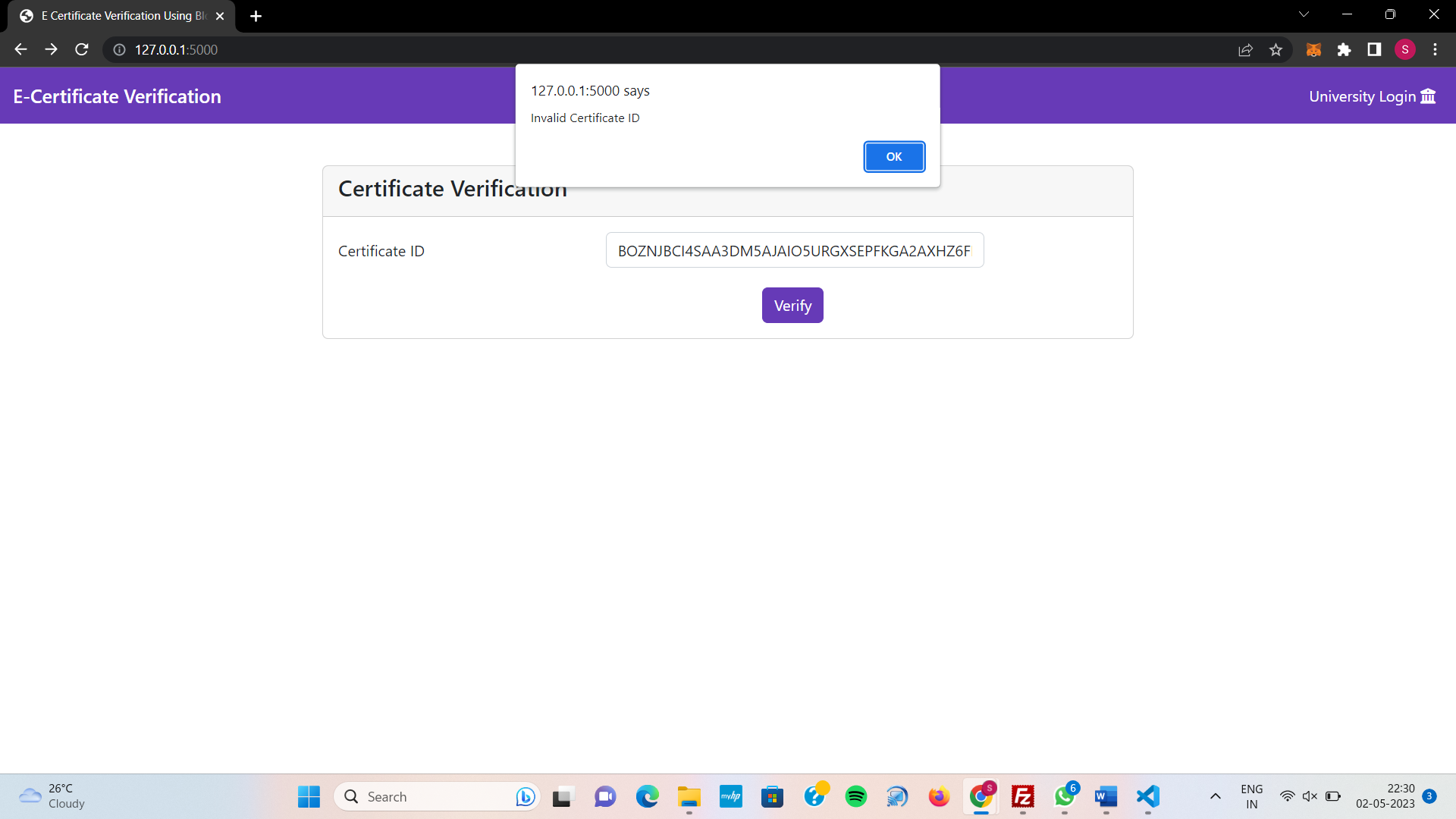
**Usage Of The Application :**

The Application Has 2 Main Functionalities in two Pages :

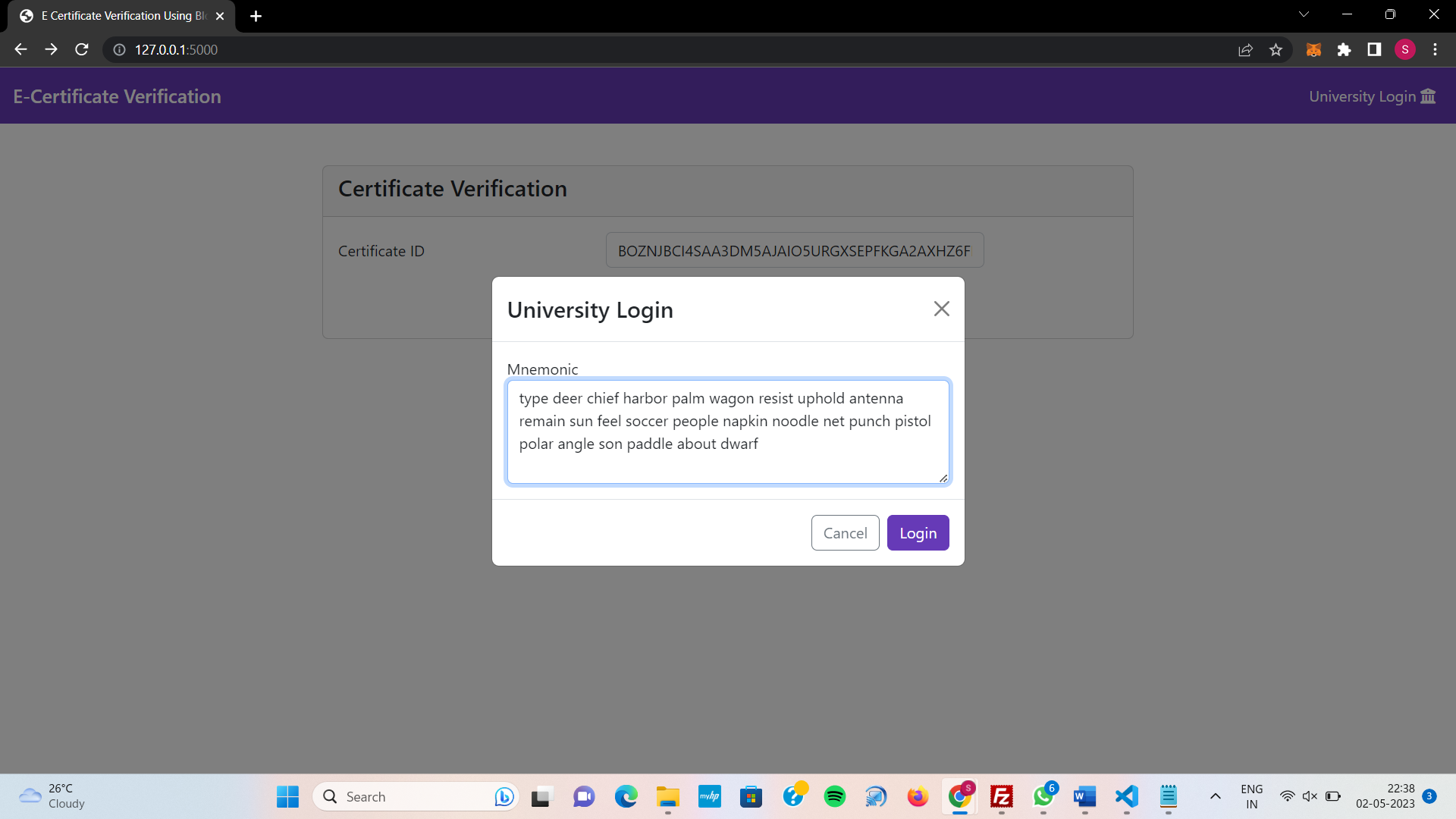
1. E-Certificate Verification ( “/” route)
2. E-Certificate Upload (“/uc” route) - **Only For University**
3. **E-Certificate Verification :** Just Enter The Certificate ID Provided By The University in Certificate ID Field and Enter Verify, If it is a Valid University Issued Certificate, Then the **Certificate will be Displayed** as Below :



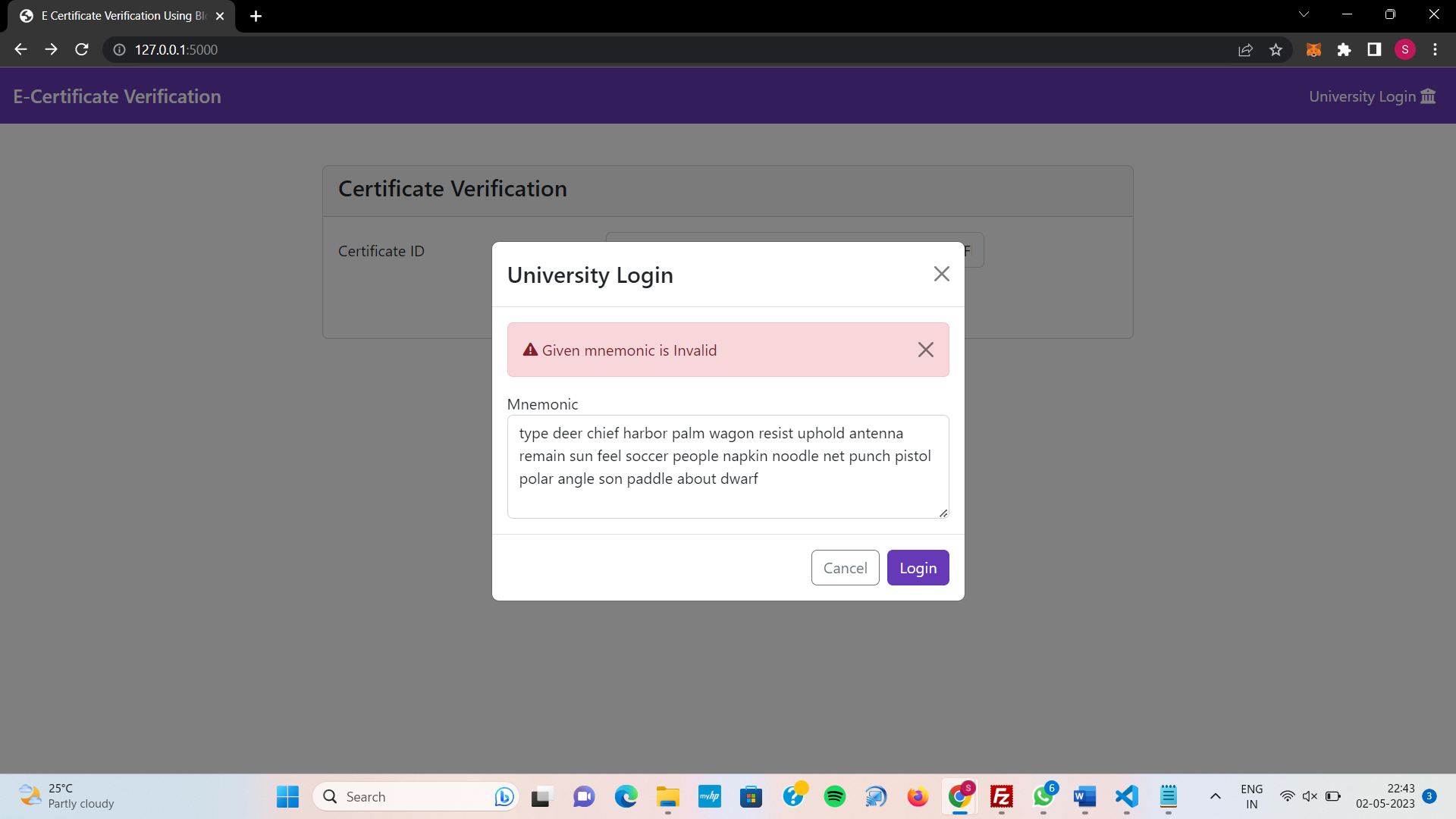
If the Certificate ID is invalid or the Certificate ID is not issued By The University, then the alert with “**Invalid Certificate ID**” will be Displayed as Shown Below :



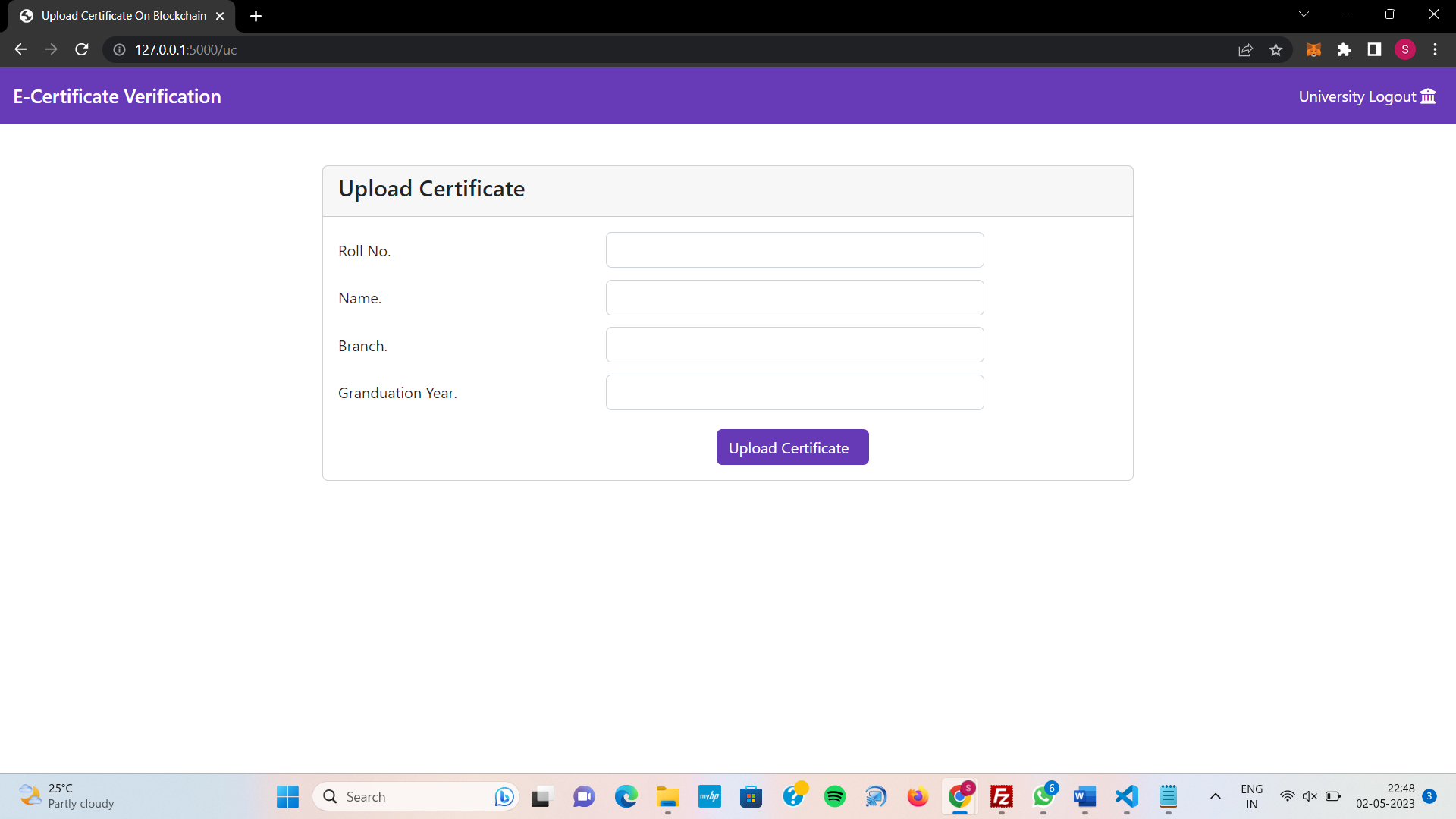
1. **E-Certificate Upload :** This Functionality is Available For The University, So We have To **Login** By Clicking University Login Icon in MenuBar of the Dapp and Entering The University Algorand Account Mnemonic as Shown Below :



Now Click Login Button To Continue To The Upload Certificate Page, If Anything Wrong in the Mnemonic Key Leads To **Login Failure** Which Is Indicated By a Alert Shown Below :

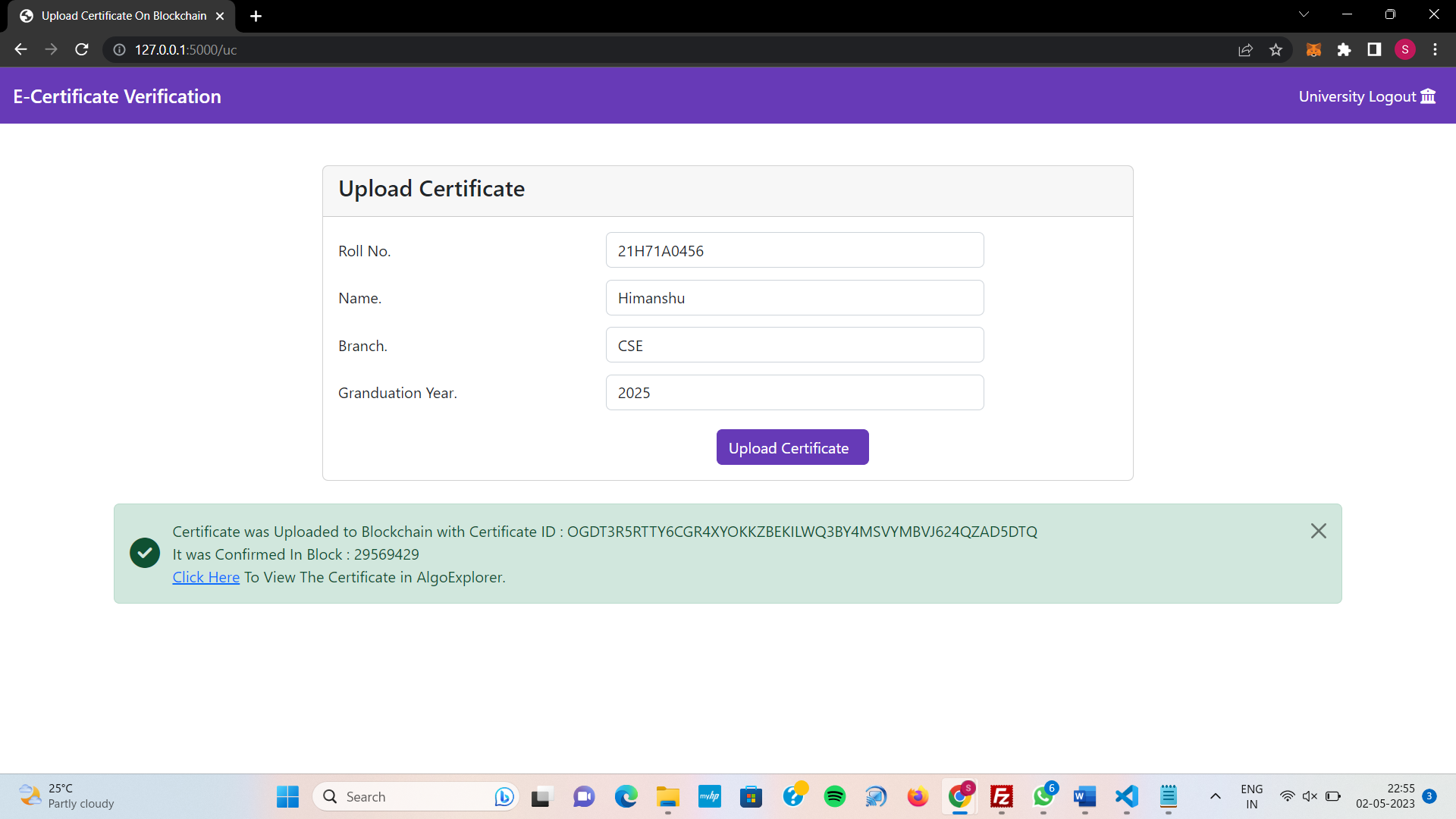


If The Login is Successful Then You Will be **Redirect to the Page** With a Form To Upload E-Certificates To The Blockchain as Shown Below :



Now, If We Fill the Form For The E-Certificate Details And hit Upload Certificate Button, the Data With Signature of University is Placed On The Blockchain and Displays The Certificate ID and Block Number (In Which Block The Data is Stored) After The Block Confirmation in the Blockchain, This Proccess Usually Takes 4-5 Seconds.

After Successful Upload of E-Certificate, The Below Success Alert will be Displayed :



**Note :** If We Click on Click Here Hyperlink in the Alert , We Can **View The Transaction** of E-certificate Data Which Was Signed By The University in the **AlgoExplorer**.

**Application Code Structure :**

This application code can be divided into the following components:

1. Importing Libraries
2. Global Constants Declaration
3. Flask Setup and Routes:
4. Login and Authentication:
5. Certificate Upload:
6. Certificate Retrieval:
7. **Importing Libraries :**

The provided application code imports several libraries to enable specific functionalities within the web application. Here's a brief explanation of each imported library:

**1. `Flask`:** Flask is a web framework for Python that provides the core functionality for building web applications. It handles routing, request handling, and response generation.

**2. `request`:** The `request` module from Flask allows the application to access data from incoming HTTP requests, such as form data or query parameters.

**3. `session`:** The `session` module from Flask provides session management capabilities. It allows the application to store user-specific data securely across multiple requests.

**4. `render\_template`:** The `render\_template` function from Flask is used to render HTML templates. It allows the application to generate dynamic HTML pages by combining templates with data.

**5. `algosdk.v2client.algod`:** This module is part of the `algosdk` library, which provides functionality for interacting with the Algorand blockchain. The `algod` module specifically enables communication with the Algorand node to perform operations such as sending transactions.

**6. `algosdk.v2client.indexer`:** Another module from the `algosdk` library, `indexer` allows the application to interact with the Algorand Indexer API. It provides functionality for searching and retrieving information about transactions and accounts on the blockchain.

**7. `algosdk.account`:** The `account` module from the `algosdk` library provides functions for working with Algorand accounts, such as deriving public addresses from private keys.

**8. `algosdk.mnemonic`:** The `mnemonic` module from the `algosdk` library offers functions for working with mnemonic phrases, including converting them to private keys.

**9. `algosdk.util`:** The `util` module from the `algosdk` library provides utility functions for working with Algorand, such as signing and verifying transactions.

**10. `algosdk.transaction`:** The `transaction` module from the `algosdk` library allows the application to create and manipulate Algorand transactions.

**11. `algosdk.error`:** The `error` module from the `algosdk` library contains error classes that are raised in case of Algorand-related errors.

**12. `json`:** The `json` module is a standard Python library for working with JSON data. It provides functions for serializing and deserializing JSON objects.

**13. `base64`:** The `base64` module is a standard Python library for working with base64-encoded data. It provides functions for encoding and decoding data in base64 format.

**14. `flask\_cors`:** The `flask\_cors` library is a Flask extension that enables Cross-Origin Resource Sharing (CORS). It allows the web application to handle requests from different domains.

By importing these libraries, the application gains access to various functionalities required for routing, request handling, Algorand blockchain interaction, session management, and data serialization, which are essential for the functionality of the web application.

**Code Block :**

from flask import Flask, request,session,render\_template

from algosdk.v2client import algod,indexer

from algosdk import account, mnemonic

from algosdk import util,transaction,error

import json,base64

from flask\_cors import CORS

1. **Global Constants Declaration :**

There is a section dedicated to declaring global constants. These constants are used throughout the application and play a crucial role in its functionality. Let's understand the purpose of each global constant:

1. **API\_KEY**: This constant represents the API key required for interacting with the Algorand blockchain. It is used for authenticating API requests sent to the Algorand network.
2. **UNIVERSITY\_PUBLIC\_KEY**: This constant holds the public key of the university. It is used for verifying the authenticity of the mnemonic provided during the login process. If the provided mnemonic corresponds to the private key associated with this public key, the user is authenticated as a representative of the university.

**Code Block :**

##### GLOBAL CONSTANTS ##########

API\_KEY="<purestake-api-key>"

UNIVERSITY\_PUBLIC\_KEY="<university-public-key>"

#################################

**3. Flask Setup and Routes:**

This section initializes the Flask application, sets a secret key for session management, and enables Cross-Origin Resource Sharing (CORS) to handle requests from different domains. It also defines various routes for different pages, such as the homepage, login page, certificate upload page, and logout functionality.

**Code Block :**

**app = Flask(\_\_name\_\_)**

**app.secret\_key = 'algo-project'**

**CORS(app,resources={r"/\*": {"origins": "\*"}})**

**@app.route('/')**

**def home():  
@app.route('/uc')**

**def uc():**

**@app.route('/login', methods=['POST'])**

**def login():**

**@app.route('/logout', methods=['GET'])**

**def logout():**

**@app.route('/auth', methods=['GET'])**

**def auth():**

**@app.route('/upload\_certificate', methods=['POST'])**

**def upload\_certificate():’’**

**@app.route('/get\_certificate', methods=['POST'])**

**def get\_certificate():**

**4. Login and Authentication:**

Let's break down the login and authentication module step by step:

1. **Login** (**/login** route):
   * This route is accessed using the POST method.
   * It retrieves the mnemonic from the request parameters.
   * It uses the mnemonic to derive the private key.
   * The public address corresponding to the private key is obtained.
   * If the public address matches the UNIVERSITY\_PUBLIC\_KEY, the user is considered authenticated, and their private key is stored in the session as user.
   * Returns a JSON response indicating the login status and any potential errors.

**Code Block :**

@app.route('/login', methods=['POST'])

def login():

    # Get the mnemonic from the request parameters

    mnemoni = request.form.get('mnemonic')

    # Derive the private key from the mnemonic

    try:

        private\_key = mnemonic.to\_private\_key(mnemoni)

    except Exception as e:

        print(e)

        return json.dumps({"status":False,"err":"Given mnemonic is Invalid"})

    # Return the public address corresponding to the private key

    public\_address = account.address\_from\_private\_key(private\_key)

    if(public\_address==UNIVERSITY\_PUBLIC\_KEY):

        session['user']=private\_key

        return json.dumps({"status":True})

    else:

        return json.dumps({"status":False,"err":"Given mnemonic is Not Owned By University"})

1. **Logout** (**/logout** route):
   * This route is accessed using the GET method.
   * If a user session exists (**session.get('user')**), it removes the **user** key from the session, effectively logging the user out.
   * Returns a JSON response indicating the logout status.

**Code Block :**

@app.route('/logout', methods=['GET'])

def logout():

    if session.get('user'):

        session.pop('user')

        return json.dumps({"status":True})

    else:

        return json.dumps({"status":False,"err":"Please Login First.!!"}

1. **Authentication** (**/auth** route):
   * This route is accessed using the GET method.
   * It checks if a user session exists (**session.get('user')**).
   * Returns a JSON response indicating the authentication status.

**Code Block :**

@app.route('/auth', methods=['GET'])

def auth():

    if session.get('user'):

        return json.dumps({"status":True})

    else:

        return json.dumps({"status":False})

The login and authentication module ensures that only users with a valid mnemonic and corresponding public key can log in and perform authorized actions within the application. It relies on the Flask session object to store the authenticated user's private key temporarily, allowing subsequent requests to maintain the user's session state.

By separating the login and authentication functionality into distinct routes, the application provides a secure login process and enables subsequent authorization checks for protected routes.

1. **Certificate Upload:**

Steps Involved in this Module in brief :

* **S**The route is a POST endpoint that allows users to upload a certificate to the Algorand blockchain.
* The route first checks if the user is logged in. If the user is not logged in, the route returns an error message.
* If the user is logged in, the route retrieves the user's private key from the session.
* The route then converts the JSON data to bytes and signs it with the user's private key.
* The route then creates a payment transaction with the signed data and submits it to the Algorand network.
* The route waits for the transaction to be confirmed and then returns the transaction ID and block number.

**Code Block :**

@app.route('/upload\_certificate', methods=['POST'])

def upload\_certificate():

algod\_address = "https://testnet-algorand.api.purestake.io/ps2"

algod\_token = ""

headers = {

"X-API-Key": API\_KEY,

}

if session.get('user'):

private\_key=session['user']

data=request.form.get('data')

print('data',request.form)

data=json.loads(data)

algod\_client = algod.AlgodClient(algod\_token, algod\_address, headers)

try:

sender\_address = account.address\_from\_private\_key(private\_key)

except Exception as e:

return json.dumps({"status":False,"err":"Your Private Key Length is too Short"})

# Convert the JSON data to bytes

signature = util.sign\_bytes(data['rollno'].encode('utf-8'), private\_key)

data['hash']=signature

note = json.dumps(data).encode()

params = algod\_client.suggested\_params()

txn = transaction.PaymentTxn(sender\_address,params,sender\_address,0,note=note)

# Sign the transaction with your private key

signed\_txn = txn.sign(private\_key)

# Submit the transaction to the Algorand network

try:

tx\_id = algod\_client.send\_transaction(signed\_txn)

except error.AlgodHTTPError as e:

return json.dumps({"status":False,"err":"Your Private Key is Invalid For Signature"})

# Wait for transaction confirmation

confirmed\_txn = algod\_client.pending\_transaction\_info(tx\_id)

while not confirmed\_txn.get('confirmed-round'):

confirmed\_txn = algod\_client.pending\_transaction\_info(tx\_id)

return json.dumps({"status":True,"resp":{"txn\_id":tx\_id,"block":confirmed\_txn["confirmed-round"]}})

else:

return json.dumps({"status":False,"err":"Please Login To Upload Certificate"})

1. **Certificate Retrival :**

Steps Involved in this Module in brief :

* The route is a POST endpoint that allows users to get a certificate from the Algorand blockchain.
* The route first retrieves the transaction ID from the request body.
* The route then uses the indexer library to retrieve the transaction information from the Algorand blockchain.
* The route then verifies that the transaction was sent by the university and that the signature is valid.
* If the transaction is valid, the route returns the certificate data.
* If the transaction is not valid, the route returns an error message.

**Code Block :**

@app.route('/get\_certificate', methods=['POST'])

def get\_certificate():

algod\_address = "https://testnet-algorand.api.purestake.io/idx2"

algod\_token = ""

headers = {

"X-API-Key": API\_KEY,

}

txn\_id=request.form.get("txn\_id")

algod\_client = indexer.IndexerClient(algod\_token, algod\_address, headers)

try:

info=algod\_client.search\_transactions(txid=txn\_id)

except error.AlgodHTTPError as e:

print(e)

if(str(e)=='no valid transaction ID was specified'):

return json.dumps({"status":False,"err":"Invalid Certificate ID"})

try:

print(info)

note=info['transactions'][0]['note']

except KeyError as e:

return json.dumps({"status":False,"err":"Invalid Certificate ID"})

except IndexError as e:

return json.dumps({"status":False,"err":"Invalid Certificate ID"})

sender=info['transactions'][0]['sender']

reciever=info['transactions'][0]['payment-transaction']['receiver']

if(sender==reciever and sender==UNIVERSITY\_PUBLIC\_KEY):

note=base64.b64decode(note).decode('utf-8')

note=json.loads(note)

sig=note['hash']

typ=note['type']

rollno=note['rollno']

name=note['name']

branch=note['branch']

year=note['year']

verified = util.verify\_bytes(rollno.encode('utf-8'),sig, sender)

if(verified):

return json.dumps({"status":True,"data":{"type":typ,"rollno":rollno,"name":name,"branch":branch,"year":year}})

else:

return json.dumps({"status":False,"err":"Certificate Not Signed By The University"})

else:

return json.dumps({"status":False,"err":"Certificate Origin is Not From University"})