Pitch Deck

NEW BUSINESS OPPORTUNITY

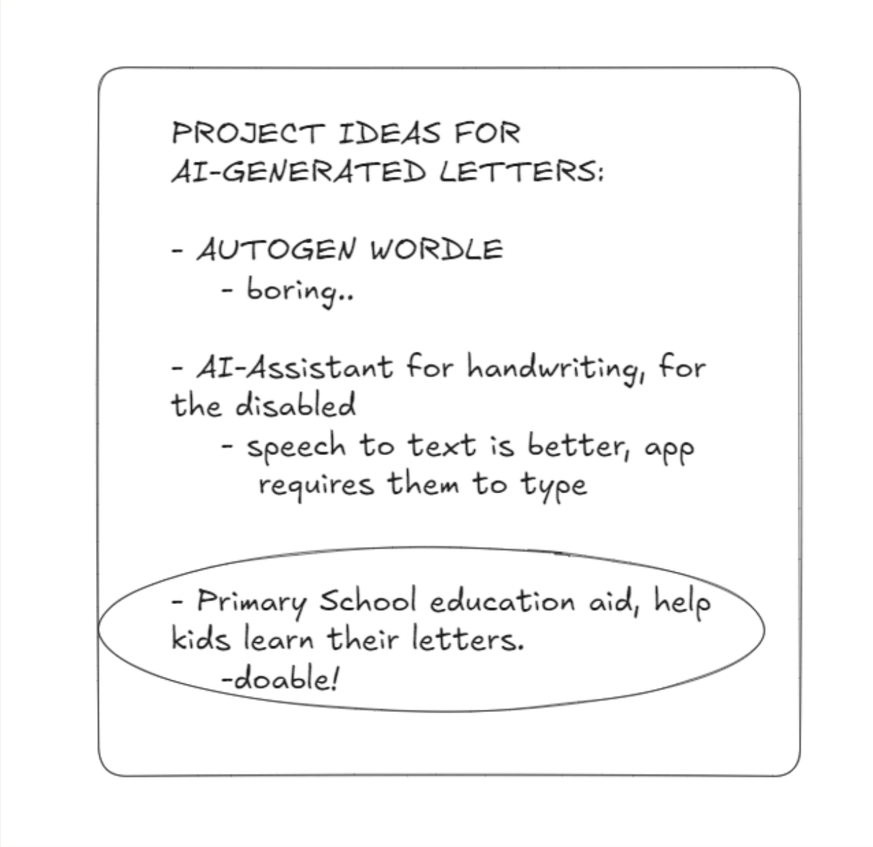
Henrietta Mitchell, Founder & CEO
Matt Zhang, Founder & CTO
13 September, 2023

Ingoude Company

TASK

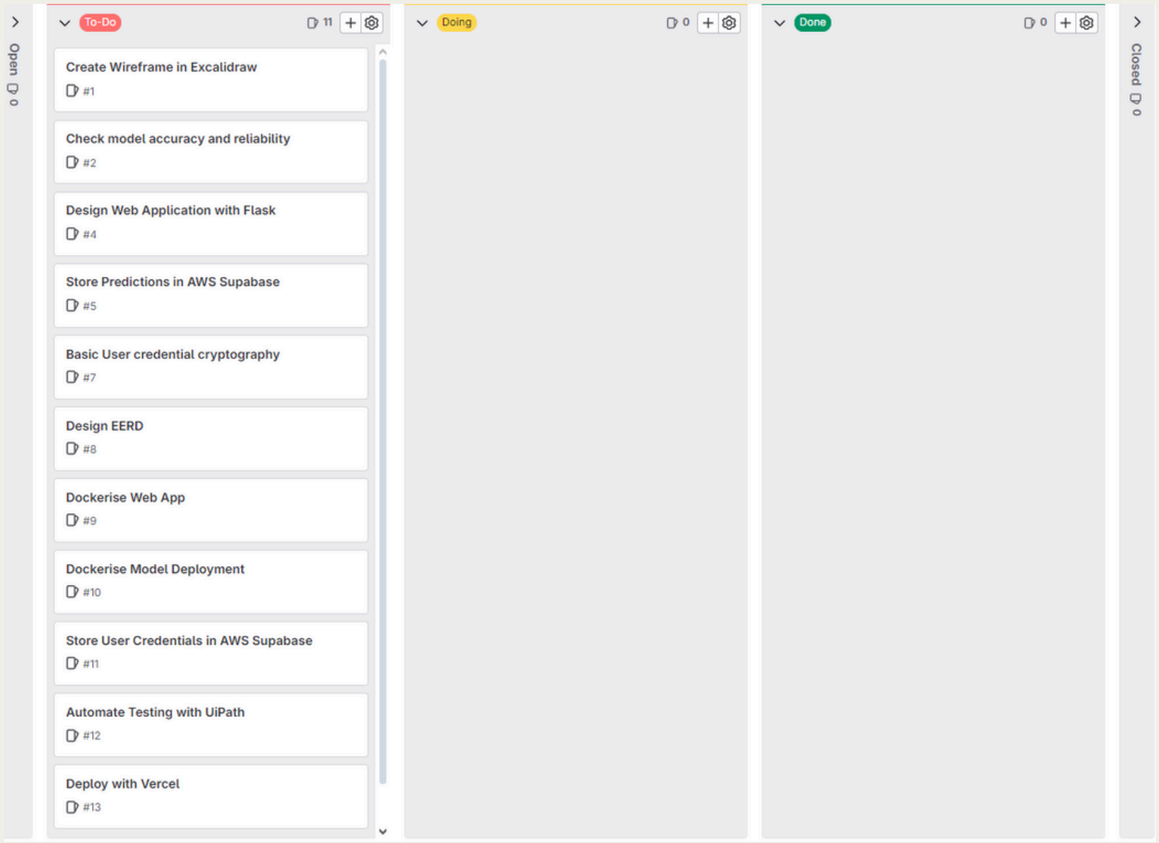
Develop an interactive web application for Deep learning with Flask, model serving, Docker and DevOps Best Practices.

PRE-PLANNING (IDEATION)



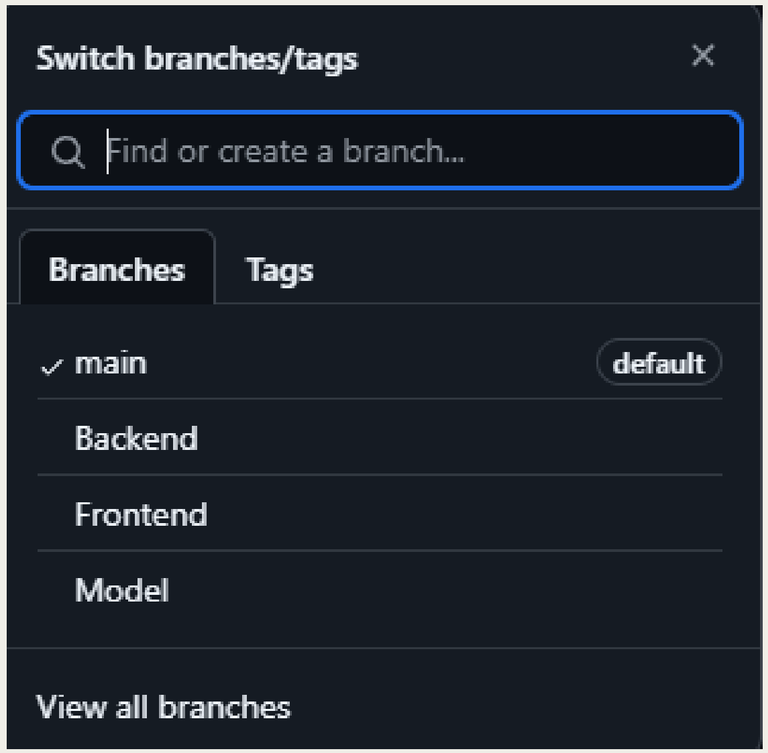
Before starting any planning, I ideated on the theme that I would like my project to have, and its target audience. I decided on an app to help young kids learn their English letters in a fun way!

PRE-PLANNING (SCRUM BOARD)



In order to prepare for the project efficiently, I first created a SCRUM board with GitLab, to ideate, identify user needs, and break down large Product Backlog Items (PBIs) to smaller tasks.

PRE-PLANNING (GIT SETUP & TECH STACK)



Dockerised Models

docker Compose

TensorFlow

(Tensorfow/Serving)

Server

Flask
web development, one drop at a time

Flask

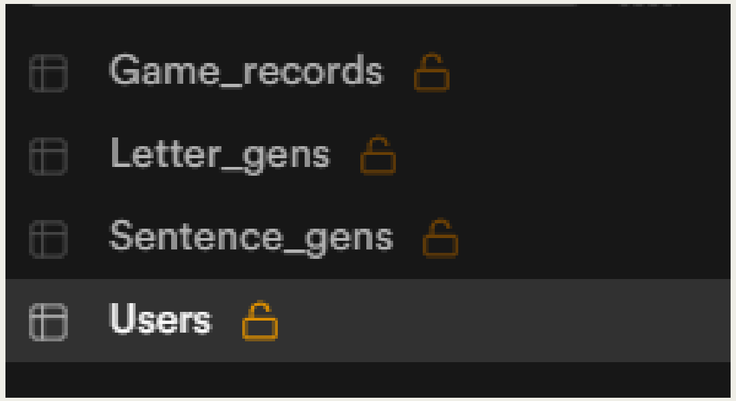
Database

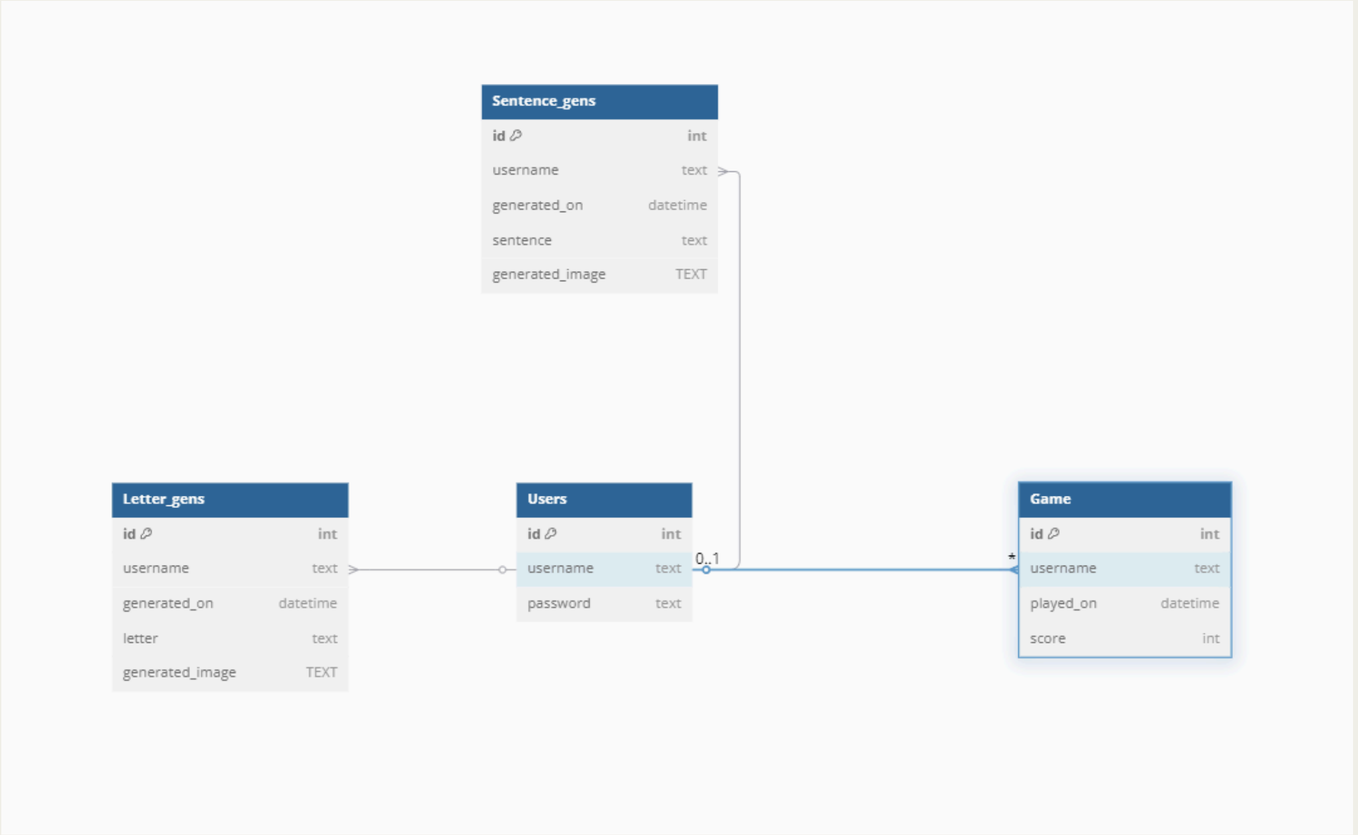
supabase

AWS Supabase

I then created my repository, and created the core branches that would make up my project. Thereafter, I decided on the tech stack that would make my project happen.

PRE-PLANNING (AWS SUPABASE SETUP)





After the tech stack was decided, I planned out my database schema as shown above, with Users being the central table that all other tables would reference. This makes it easy to manage FK references, and operations like ON DELETE CASCADE when I have to remove a user.

IMPLEMENTATION (DOCKER COMPOSE SETUP)

Testing functions of docker containers

```python
import random
import PIL.Image
import numpy as np
import PIL
import requests
from flask import Flask, jsonify, request
import io
import base64
import json

app = Flask(__name__)

# The URL for the TensorFlow Serving model endpoints
GENERATOR_URL = "http://localhost:8501/v1/models/generator:predict"
DISCRIMINATOR_URL = "http://localhost:8502/v1/models/discriminator:predict"

@app.route('/generate', methods=['POST'])
def generate():
    latent_vector = [round(random.random(), 2) for _ in range(100)]  # Example: Random latent vector with 100 values

    payload = {
        "instances": [
            {
                "keras_tensor_19": [3],
                "keras_tensor_23": latent_vector
            }
        ]
    }

    response = requests.post(GENERATOR_URL, json=payload)

    if response.status_code == 200:
        prediction = response.json()
        image_data = prediction['predictions'][0]

        print(type(prediction['predictions'][0]))
        image_data = np.array(image_data).reshape(28, 28)
        image_data = ((image_data + 1) * 127.5).astype(np.uint8)


        image = PIL.Image.fromarray(image_data)
        image_path = 'generated_image.png'
        image.save(image_path)
        return jsonify({"message": "Image generated successfully!", "image_path": image_path})

    else:
        return jsonify({"error": "Failed to generate image", "details": response.text}), 500
```

saving models in SavedModel format

```python
acgan.generator.export('ACGAN_Gen_SavedModel/1')

# Save the Discriminator model
acgan.discriminator.export('ACGAN_Disc_SavedModel/1')
```

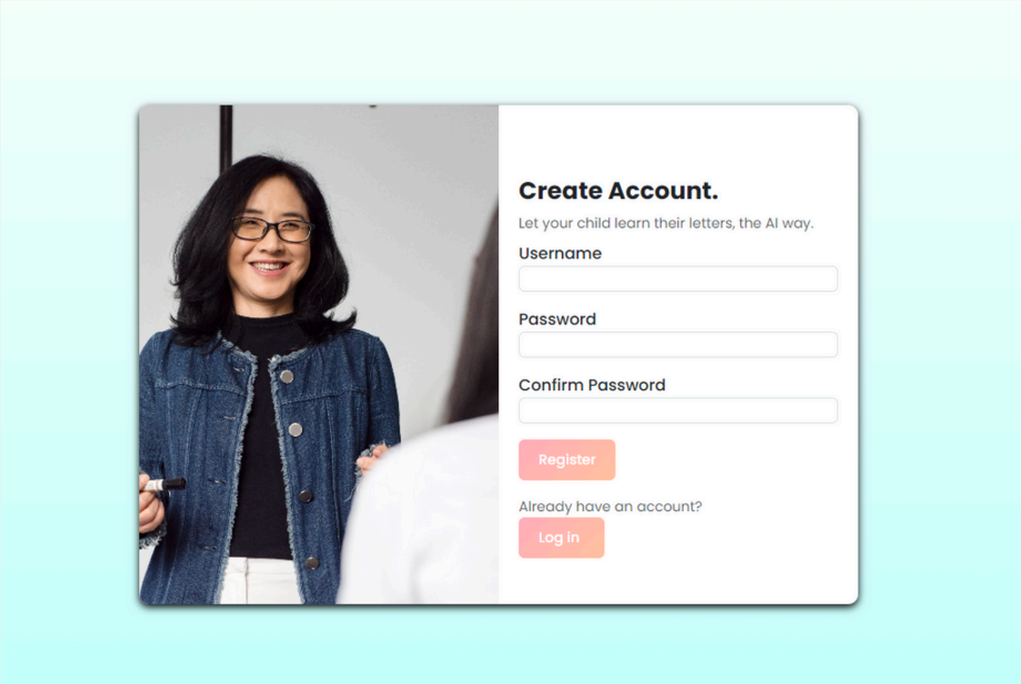
Final docker-compose.yml

```yaml
services:
  ▷ Run Service
  generator:
    image: tensorflow/serving:latest
    container_name: generator_serving
    volumes:
      - ./Model/ACGAN_Gen_SavedModel/1:/models/generator/1
    environment:
      - MODEL_NAME=generator
    ports:
      - "8501:8501"
    networks:
      - acgan_network

  ▷ Run Service
  discriminator:
    image: tensorflow/serving:latest
    container_name: discriminator_serving
    volumes:
      - ./Model/ACGAN_Disc_SavedModel/1:/models/discriminator/1
    environment:
      - MODEL_NAME=discriminator
    ports:
      - "8502:8501"
    networks:
      - acgan_network

networks:
  acgan_network:
    driver: bridge
```

Before creating the rest of my project, I first made sure I could access both my ACGAN Generator and Discriminator models using Docker Compose. This step turned out to be one of the hardest, taking many hours of debugging before getting working containers.

WEB APP DEVELOPMENT: REGISTER/LOGIN

Webpage Preview



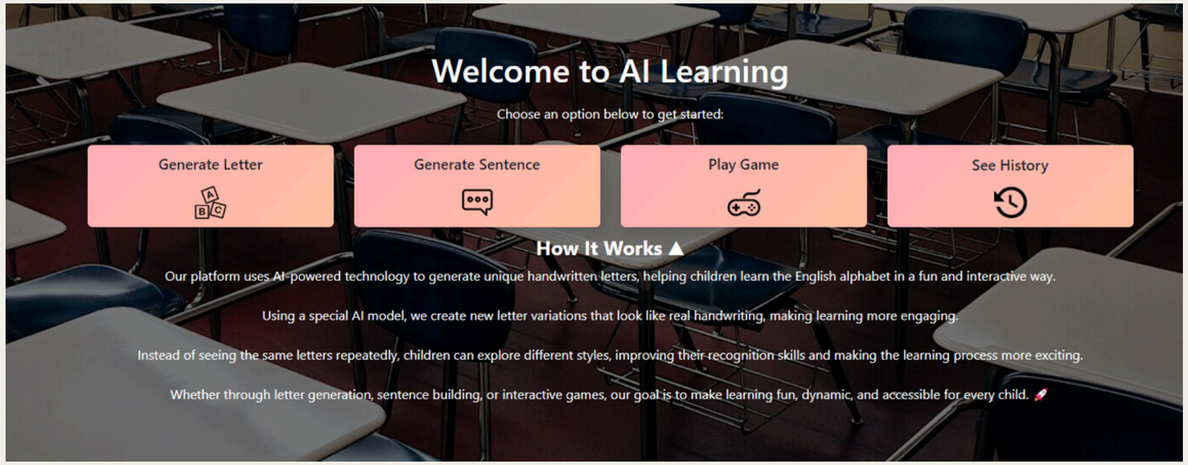
app.py, register endpoint

```python
@app.route('/register', methods=['POST'])
def register():
    username = request.form.get('username', '').strip()
    password = request.form.get('password', '').strip()
    confirm_password = request.form.get('confirm_password', '').strip()

    if not username or not password or not confirm_password:
        flash('All fields are required!', 'warning')
        return redirect(url_for('landing_register'))

    if len(username) < 3:
        flash("Username must be at least 3 characters long.", "danger")
        return redirect(url_for('landing_register'))

    if not re.match("^[a-zA-Z0-9_]*$", username):
        flash("Invalid username: Only letters, numbers, and underscores are allowed.", "danger")
        return redirect(url_for('landing_register'))


    if password != confirm_password:
        flash('Passwords do not match!', 'danger')
        return redirect(url_for('landing_register'))

    hashed_password = generate_password_hash(password, method='pbkdf2:sha256')

    try:
        # Check if username exists
        existing_user = supabase.table("Users").select("username").eq("username", username).execute()
        if existing_user.data:
            flash('Username already taken!', 'danger')
            return redirect(url_for('landing_register'))

        # Insert user into Supabase
        response = supabase.table("Users").insert({"username": username, "password": hashed_password}).execute()

        if response.data:
            flash('Registration successful! Please log in.', 'success')
            return redirect(url_for('login'))
        else:
            flash('Registration failed. Try again.', 'danger')

    except Exception as e:
        flash(f"Error: {str(e)}", 'danger')

    return redirect(url_for('landing_register'))
```

landing_register.html

The first webpage I created was my landing page (which also serves as registration), taking inspiration from registration pages online for a fun and modern look. Throughout the process, I used VSCode's "Open in default browser" extension to easily preview my webpages.

WEB APP DEVELOPMENT: HOME



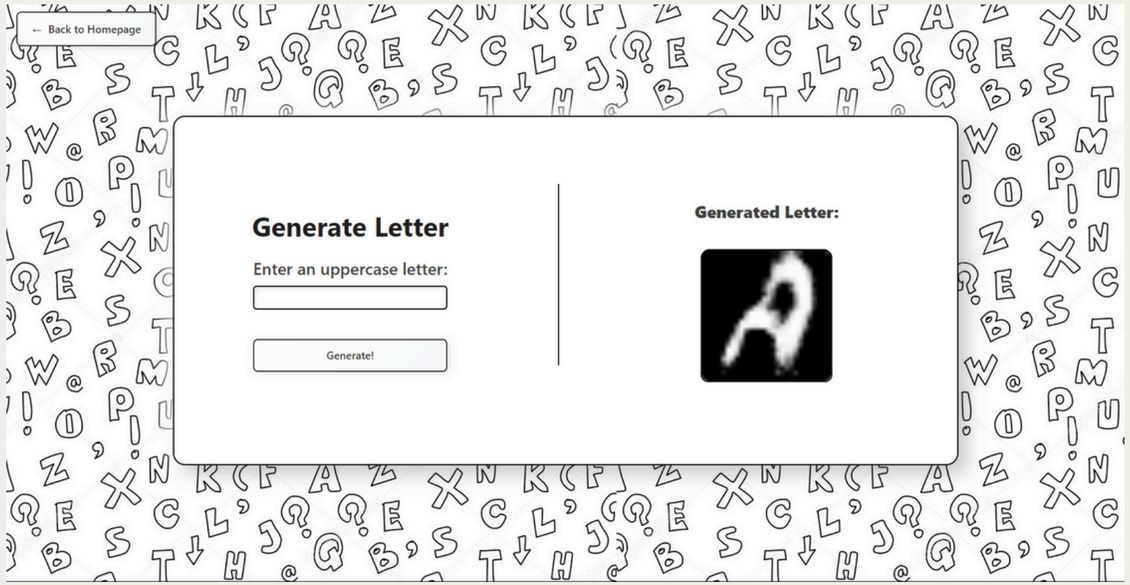
app.py, home endpoint

```python
@app.route('/home')
def home():
    if 'username' not in session:
        flash('Please log in first.', 'warning')
        return redirect(url_for('login'))
    return render_template('home.html')
```

part of home.html

Next, I made my homepage, which serves as the hub for all other functions of my website. It’s crucial here that the homepage is attractive and vibrant, as the user will be returning to this page alot. I’m quite proud of the design!

WEB APP DEVELOPMENT: LETTER GENERATION



app.py, /generate_letter endpoint

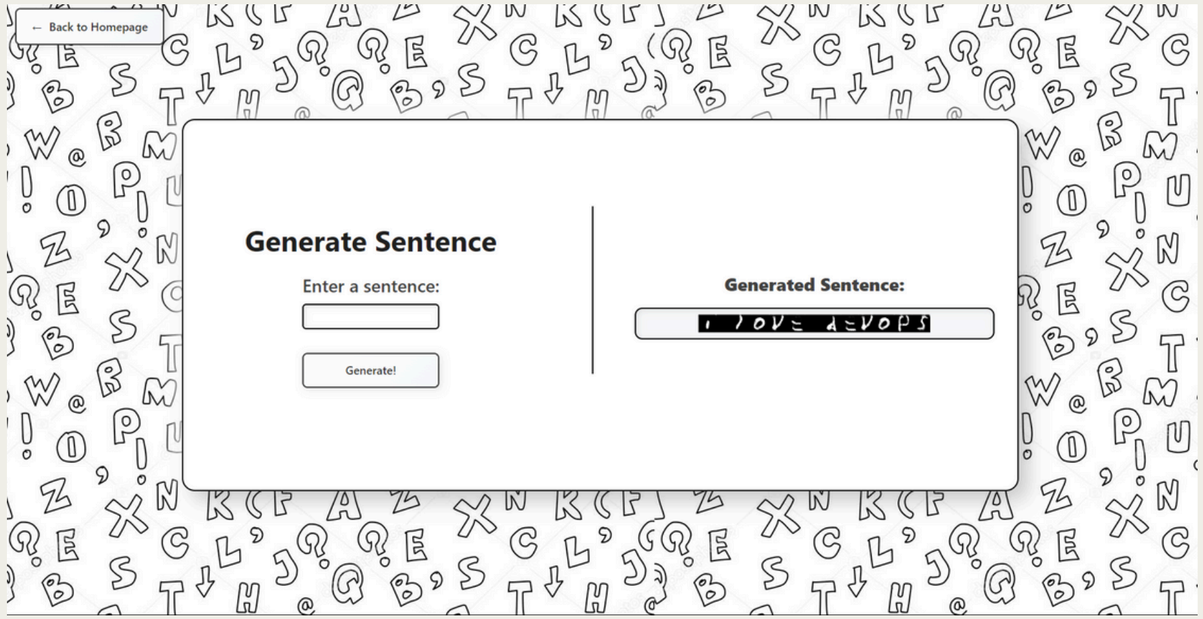
```python
@app.route('/generate_letter', methods=['GET', 'POST'])
def generate_letter():
    if 'username' not in session:
        flash('Please log in first.', 'warning')
        return redirect(url_for('login'))

    if request.method == 'POST':
        # Retrieve and validate the letter
        letter = request.form.get('letter', '').strip()
        if len(letter) == 1 and letter.isalpha() and letter.isupper():
            flash(f"Generating content for letter: {letter}", "success")

            latent_vector = [round(random.random(), 2) for _ in range(100)]
            letter_index = ord(letter) - ord('A')

            payload = {
                "instances": [
                    {
                        "keras_tensor_19": [letter_index],
                        "keras_tensor_23": latent_vector
                    }
                ]
            }

            response = requests.post(GENERATOR_URL, json=payload)

            if response.status_code == 200:
                prediction = response.json()
                image_data = prediction["predictions"][0]

                # Convert array -> numpy -> PIL image -> raw bytes
                image_data = np.array(image_data).reshape(28, 28)
                image_data = ((image_data + 1) * 127.5).astype(np.uint8)  # Rescale [-1..1] -> [0..255]
                image = PIL.Image.fromarray(image_data)

                from io import BytesIO
                buffer = BytesIO()
                image.save(buffer, format="PNG")
                buffer.seek(0)
                image_bytes = buffer.read()

                # Base64-encode before storing in a text column
                import base64
                image_b64 = base64.b64encode(image_bytes).decode("utf-8")

                # Insert into Letter_Gen (text column for generated_image)
                username = session.get("username")
                try:
                    response_supabase = supabase.table("Letter_gens").insert({
                        "username": username,
                        "letter": letter,
                        "generated_image": image_b64
                    }).execute()

                    if not response_supabase.data:
                        flash(f"Error saving to Supabase: {response_supabase.error}", "danger")

                except Exception as e:
                    flash(f"Error storing in Supabase: {str(e)}", "danger")

                # Pass the base64 string to the template
                return render_template("generate_letter.html", image_data=image_b64)
            else:
                flash(f"Error generating image: {response.text}", "danger")
                return redirect(url_for("generate_letter"))
        else:
            flash("Please enter exactly one uppercase letter (A–Z).", "danger")
            return redirect(url_for("generate_letter"))
```

part of generate_letter.html

```html
<body class="d-flex">
    <a href="/home" class="back-home-btn">
        <span class="arrow">&larr;</span> Back to Homepage
    </a>
    {% with messages = get_flashed_messages(with_categories=true) %}
        {% if messages %}
            {% for category, message in messages %}
                <div class="alert alert-{{ category }}">{{ message }}</div>
            {% endfor %}
        {% endif %}
    {% endwith %}
    <div class="container my-5">

        <!-- Two-column layout -->
        <div class="row">
            <!-- Left: Form Section -->
            <div class="col-md-6 form-section">
                <h1 class="mb-4">Generate Letter</h1>
                <form action="/generate_letter" method="POST">
                    <div class="mb-3">
                        <label for="letter" class="form-label">Enter an uppercase letter:</label>
                        <input
                            type="text"
                            class="form-control"
                            id="letter"
                            name="letter"
                            required
                            maxlength="1"
                            pattern="[A-Z]"
                            title="Please enter exactly one uppercase letter (A–Z)."
                        />
                    </div>
                    <button type="submit" class="btn btn-primary submit-btn">Generate!</button>
                </form>
            </div>

            <!-- Vertical Divider -->
            <div class="col-md-1 d-flex align-items-center justify-content-center">
                <div class="divider"></div>
            </div>

            <!-- Right: Image Section -->
            <div class="col-md-5 image-section">
                <p class="gen_image_title"><strong>Generated Letter:</strong></p>
                <div class="generated-image">
                    {% if image_data %}
                        <div class="mb-3">
                            <img class="generated-image" src="{{ image_data }}">
                        </div>
                    {% else %}
                        <p class="placeholder-text">Your generated letter will appear here.</p>
                    {% endif %}
                </div>
            </div>
        </div>
    </div>
</body>
</html>
<script>
    // Automatically hide flash messages after 5 seconds
    setTimeout(function() {
        let alerts = document.querySelectorAll(".alert");
        alerts.forEach(function(alert) {
            alert.style.transition = "opacity 0.5s ease-out";
            alert.style.opacity = "0";
            setTimeout(() => alert.remove(), 500); // Remove from DOM after fade out
        });
    }, 2000);
</script>
```

Here is the first of my website’s core functions: Generating a letter using the dockerised model. The website ensures validity of inputs, making sure the user can only enter one character, and it must be an uppercase english character. If not, an error is thrown. Each generated character is stored in AWS Supabase, after being encoded in base64.

WEB APP DEVELOPMENT: SENTENCE GENERATION



app.py, /generate_sentence endpoint (part)

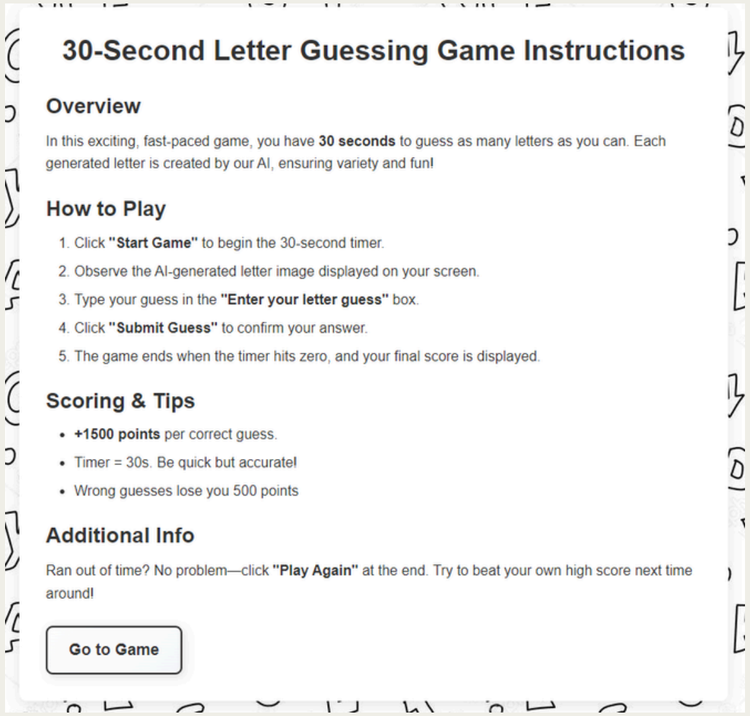
```python
@app.route('/generate_sentence', methods=['GET', 'POST'])
def generate_sentence():
    if 'username' not in session:
        flash('Please log in first.', 'warning')
        return redirect(url_for('login'))

    if request.method == 'POST':
        # 1. Retrieve and validate the text
        text = request.form.get('text', '').strip()
        if not text:
            flash("Please enter some text.", "danger")
            return redirect(url_for('generate_sentence'))
        if len(text) > 20:
            flash("text cannot exceed 20 characters.", "danger")
            return redirect(url_for('generate_sentence'))

        char_images = []

        for ch in text:
            if ch == ' ':
                # Create a black 28x28 image
                black_img = np.zeros((28, 28), dtype=np.uint8)
                pil_img = PIL.Image.fromarray(black_img)
                char_images.append(pil_img)
            else:
                # Validate uppercase letter if desired
                if not (ch.isalpha() and ch.isupper()):
                    flash("Only uppercase letters and spaces allowed!", "danger")
                    return redirect(url_for('generate_sentence'))

                # Random latent vector for each character
                latent_vector = [round(random.random(), 2) for _ in range(100)]
                letter_index = ord(ch) - ord('A')

                payload = {
                    "instances": [
                        {
                            "keras_tensor_19": [letter_index],
                            "keras_tensor_23": latent_vector
                        }
                    ]
                }

                response = requests.post(GENERATOR_URL, json=payload)
                if response.status_code != 200:
                    flash(f"Error generating letter '{ch}': {response.text}", "danger")
                    return redirect(url_for('generate_sentence'))

                # Convert array -> numpy -> PIL
                prediction = response.json()
                image_data = prediction["predictions"][0]
                image_data = np.array(image_data).reshape(28, 28)
                image_data = ((image_data + 1) * 127.5).astype(np.uint8)  # scale [-1..1]->[0..255]
                pil_img = PIL.Image.fromarray(image_data)
                char_images.append(pil_img)

        # 3. Stitch images side by side
        #    The final width is 28 * number_of_chars, height stays 28
        total_chars = len(char_images)
        final_width = 28 * total_chars
        final_image = PIL.Image.new("L", (final_width, 28))  # "L" mode for 8-bit grayscale

        x_offset = 0
        for img in char_images:
            final_image.paste(img, (x_offset, 0))
            x_offset += 28
```

part of generate_sentence.html

```html
<body class="bg-light">
    <!-- Back to Homepage Button -->
    <a href="/home" class="back-home-btn">
      <span class="arrow">&larr;</span> Back to Homepage
    </a>

    <div class="container">

      <!-- Two-column layout -->
      <div class="row">

        <!-- Left: Sentence Input Form -->
        <div class="col-md-6 form-section">
          <h1 class="mb-4">Generate Sentence</h1>

          {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
              {% for category, message in messages %}
                <div class="alert alert-{{ category }}">{{ message }}</div>
              {% endfor %}
            {% endif %}
          {% endwith %}

          <form action="/generate_sentence" method="POST">
            <label for="text" class="form-label">Enter a sentence:</label>
            <input
              type="text"
              class="form-control sentence-input"
              id="text"
              name="text"
              maxlength="20"
              required
              title="Please enter only uppercase letters and spaces."
            />
            <div class="button-group">
              <button type="submit" class="btn btn-primary submit-btn">Generate!</button>
            </div>
          </form>
        </div>

        <!-- Vertical Divider -->
        <div class="col-md-1 d-flex align-items-center justify-content-center">
          <div class="divider"></div>
        </div>

        <!-- Right: Image Preview -->
        <div class="col-md-5 image-section">
          <p class="gen_image_title"><strong>Generated Sentence:</strong></p>
          <div class="generated-image-sentence">
            {% if image_data %}
              <img src="{{ image_data }}" alt="Generated Sentence" class="generated-sentence">
            {% else %}
              <p class="placeholder-text">Your generated sentence will appear here.</p>
            {% endif %}
          </div>
        </div>

      </div>
    </div>
  </body>
</html>
<script>
  // Automatically hide flash messages after 5 seconds
  setTimeout(function() {
      let alerts = document.querySelectorAll(".alert");
      alerts.forEach(function(alert) {
          alert.style.transition = "opacity 0.5s ease-out";
          alert.style.opacity = "0";
          setTimeout(() => alert.remove(), 500); // Remove from DOM after fade out
      });
```

Next, another of my website's core functions: Generating a sentence (up to 20 characters for size) using the dockerised model. While the model cannot generate empty spaces, I used NumPy and PIL to generate the black squares for spaces, stitch all generated images together, and encode it to base64 for storage.

WEB APP DEVELOPMENT: GAME



app.py, /game endpoint (part)

```python
@app.route('/gameinstructions')
def gameinstructions():
    if 'username' not in session:
        flash('Please log in first.', 'warning')
        return redirect(url_for('login'))
    return render_template("gameinstructions.html")

@app.route('/game', methods=['GET', 'POST'])
def game():
    if 'username' not in session:
        flash('Please log in first.', 'warning')
        return redirect(url_for('login'))

    if request.method == 'GET':
        session['score'] = 0
        session['start_time'] = time.time()
        flash("Game started! You have 30 seconds.", "info")
        _generate_new_challenge()
        return render_template("game.html")

    elapsed = time.time() - session.get('start_time', 0)
    if elapsed > 30:
        final_score = session.get('score', 0)
        flash(f"Time's up! Final score: {final_score}", "info")
        return render_template("game_over.html", final_score=final_score)

    user_guess = request.form.get('guess', '').strip().upper()
    current_letter = session.get('current_letter', '')

    if user_guess == current_letter:
        session['score'] = session.get('score', 0) + 1500
        flash("Correct! +1500 points", "success")
    else:
        session['score'] = session.get('score', 0) - 500
        flash(f"Wrong! The letter was {current_letter}. -500 points", "danger")

    _generate_new_challenge()
    return render_template("game.html")

def _generate_new_challenge():
    letter = random.choice(string.ascii_uppercase)
    session['current_letter'] = letter

    latent_vector = [round(random.random(), 2) for _ in range(100)]
    letter_index = ord(letter) - ord('A')
    payload = {
                "instances": [
                    {
                        "keras_tensor_19": [letter_index],
                        "keras_tensor_23": latent_vector
                    }
                ]
            }

    response = requests.post(GENERATOR_URL, json=payload)

    if response.status_code == 200:
        prediction = response.json()
        image_data = prediction["predictions"][0]

        # Convert array -> numpy -> PIL image -> raw bytes
        image_data = np.array(image_data).reshape(28, 28)
        image_data = ((image_data + 1) * 127.5).astype(np.uint8)  # Rescale [-1..1] -> [0..255]
        image = PIL.Image.fromarray(image_data)
        buffer = BytesIO()
        image.save(buffer, format="PNG")
        buffer.seek(0)
        session['current_image'] = base64.b64encode(buffer.read()).decode('utf-8')
    else:
        flash(f"Error generating image, try again later. ")
```

To make my webapp unique, I chose to implement a fast-paced game where the user has 30 seconds to guess as many of the generated letters correctly as possible. I chose this as both a way for my users to practice their english letters, but also to show that AI generations have a limit, as the letters are not perfect.

WEB APP DEVELOPMENT: LETTER HISTORY

Letter Generation History

Back to History Menu | Search for letters...

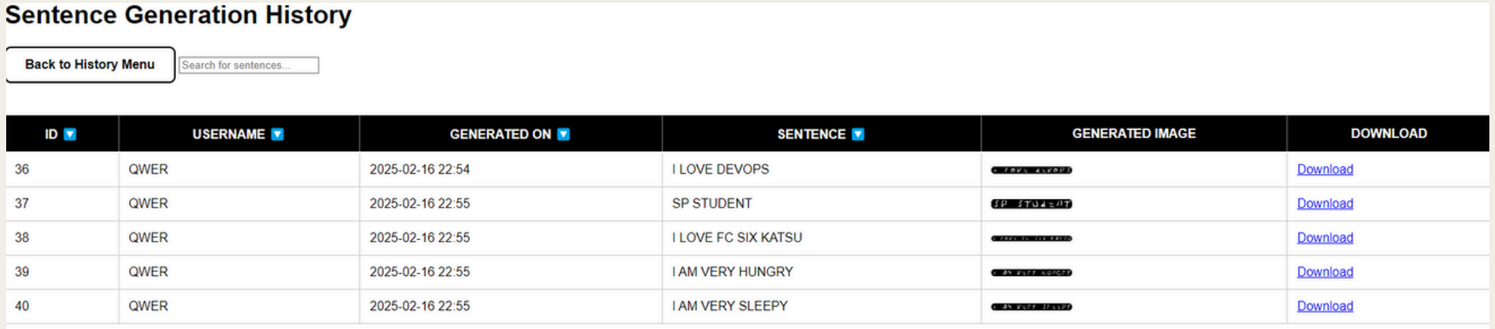
ID	USERNAME	GENERATED ON	LETTER	GENERATED IMAGE	DOWNLOAD
65	qwer	2025-02-16 17:46	A		Download
66	qwer	2025-02-16 17:46	B		Download
67	qwer	2025-02-16 17:46	C		Download
68	qwer	2025-02-16 17:46	D		Download
69	qwer	2025-02-16 17:46	E		Download
70	qwer	2025-02-16 17:46	F		Download

app.py, /history endpoints

```python
@app.route('/history')
def history_menu():
    if 'username' not in session:
        flash('Please log in first.', 'warning')
        return redirect(url_for('login'))
    return render_template("history_menu.html")

@app.route('/history/letters')
def letter_history():
    if 'username' not in session:
        flash('Please log in first.', 'warning')
        return redirect(url_for('login'))

    try:
        username = session.get("username")
        response = supabase.table("Letter_gens").select("*").eq("username", username).execute()
        letter_entries = response.data if response.data else []
    except Exception as e:
        flash(f"Error fetching letter history: {str(e)}", "danger")
        letter_entries = []

    return render_template("letter_history.html", letter_entries=letter_entries)
```

For the generation history, I showed all relevant info in a table format, including generation ID, username, date & time of generation, the character to be generated, and the final generated image. There's also a download link for each generation, if the user wishes to download. Here, the user can also filter for date ascending/descending and filter for letters

WEB APP DEVELOPMENT: SENTENCE HISTORY



app.py, /history/sentences

```python
@app.route('/history/sentences')
def sentence_history():
    if 'username' not in session:
        flash('Please log in first.', 'warning')
        return redirect(url_for('login'))

    try:
        username = session.get("username")
        response = supabase.table("Sentence_gens").select("*").eq("username", username).execute()
        sentence_entries = response.data if response.data else []
    except Exception as e:
        flash(f"Error fetching sentence history: {str(e)}", "danger")
        sentence_entries = []

    return render_template("sentence_history.html", sentence_entries=sentence_entries)
```

Similar history page for sentences and letters.

PYTEST FOR AUTOTESTING

```
PS C:\LOCAL POLY MODULES\DEVOPS\CA2\CA2\Devops.CA2> python -m unittest discover tests
.........
----------------------------------------------------------------------
Ran 9 tests in 7.256s

OK
```

some cases in test.py

```python
from flask import json
from App.app import app

class TestApp(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        cls.client = app.test_client()

    # Validity Testing
    def test_valid_registration(self):
        response = self.client.post('/register', data={
            'username': 'testuser',
            'password': 'ValidPass123',
            'confirm_password': 'ValidPass123'
        }, follow_redirects=True)
        self.assertEqual(response.status_code, 200)

    # Range Testing
    def test_invalid_username_length(self):
        response = self.client.post('/register', data={
            'username': 'a',  # Too short
            'password': 'ValidPass123',
            'confirm_password': 'ValidPass123'
        }, follow_redirects=True)

        self.assertIn(b'Username must be at least 3 characters long.', response.data)  # Ensure exact match


    # Consistency Testing
    def test_duplicate_user_registration(self):
        self.client.post('/register', data={
            'username': 'duplicateUser',
            'password': 'ValidPass123',
            'confirm_password': 'ValidPass123'
        }, follow_redirects=True)
        response = self.client.post('/register', data={
            'username': 'duplicateUser',
            'password': 'ValidPass123',
            'confirm_password': 'ValidPass123'
        }, follow_redirects=True)
        self.assertIn(b'Username already taken', response.data)

    # Unexpected Failure Testing
    def test_unexpected_input(self):
        response = self.client.post('/register', data={
            'username': 'invaliduser%$*#&@',
            'password': '<script>alert(1)</script>',
            'confirm_password': '<script>alert(1)</script>'
        }, follow_redirects=True)

        self.assertIn(b'Invalid username: Only letters, numbers, and underscores are allowed.', response.data)
```

Now that the core functions were complete, I set up test.py to perform unit and REST API endpoint testing with unittest. I tested for invalid usernames, mismatched passwords, unlogged users, invalid characters in generation input fields, and more. As I performed a lot of client-side input validation, there were not many paths to exploit my web app. The ones that were revealed by unit tests were promptly patched.

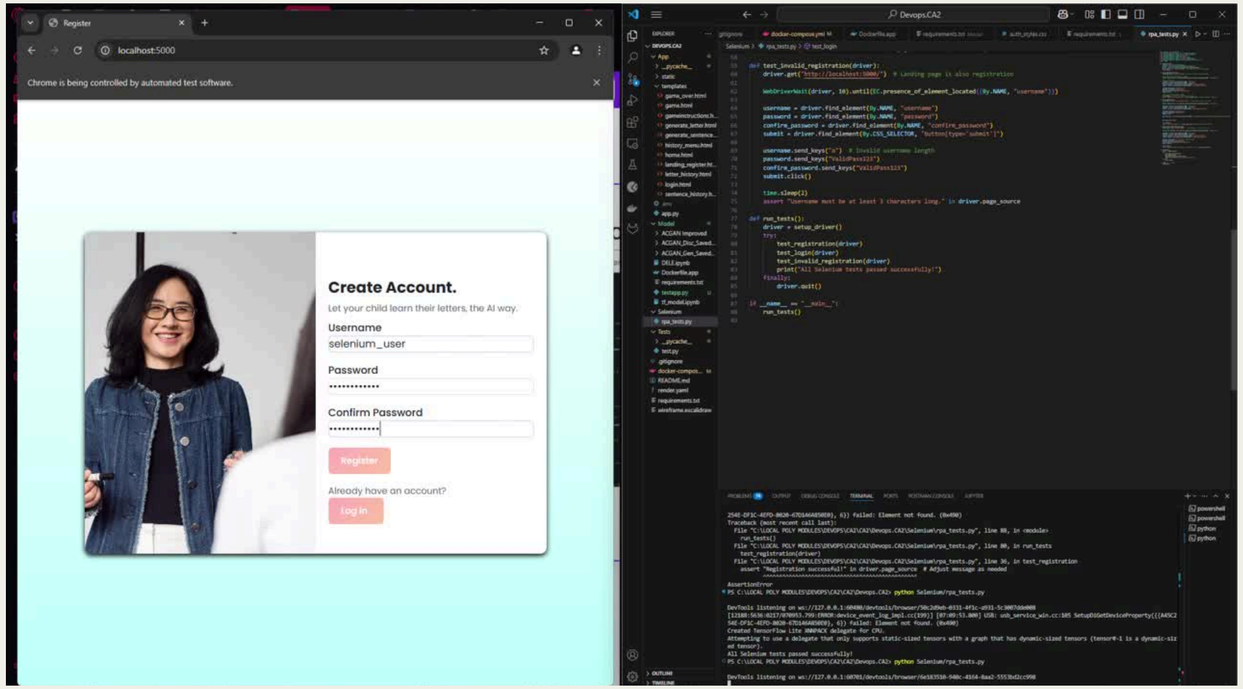
ROBOTIC PROCESS AUTOMATION (RPA) FOR TEST

Robotic Process Automation (RPA) is a technology that **automates repetitive, rule-based tasks using software robots.** In automated testing, RPA is used to streamline test execution and improve efficiency.

- Allows for low-code/no-code testing
- Highly Scalable
- Cost efficient (Selenium is free!!)

In my Project, I chose to use Selenium for my automated testing, due to familiarity and it’s ease of use.

ROBOTIC PROCESS AUTOMATION (RPA) FOR TEST



rpa_tests.py (part)

```python
from selenium import webdriver
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time

def setup_driver():
    options = webdriver.ChromeOptions()
    service = Service(ChromeDriverManager().install())
    driver = webdriver.Chrome(service=service, options=options)
    driver.get("http://localhost:5000")  # Navigate directly to landing page
    return driver

def test_registration(driver):
    driver.get("http://localhost:5000/")  # Ensure we're on the landing page

    # Wait for the username field to appear
    WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.NAME, "username")))

    # Fill in the form
    username = driver.find_element(By.NAME, "username")
    password = driver.find_element(By.NAME, "password")
    confirm_password = driver.find_element(By.NAME, "confirm_password")
    submit = driver.find_element(By.CSS_SELECTOR, "button[type='submit']")

    username.send_keys("selenium_user")
    password.send_keys("ValidPass123")
    confirm_password.send_keys("ValidPass123")
    submit.click()

    # Wait for flash message to appear
    time.sleep(2)  # Allow form submission to process
    assert "Registration successful!" in driver.page_source  # Adjust message as needed

def test_login(driver):
    driver.get("http://localhost:5000/login")  # Login page exists in your HTML

    WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.NAME, "username")))

    username = driver.find_element(By.NAME, "username")
    password = driver.find_element(By.NAME, "password")
    submit = driver.find_element(By.CSS_SELECTOR, "button[type='submit']")

    username.send_keys("selenium_user")
    password.send_keys("ValidPass123")
    submit.click()

    gen_letter_btn = driver.find_element(By.XPATH, "//a[@href='/generate_letter' and contains(@class, 'btn-option')]")


    gen_letter_btn.click()

    time.sleep(2)
    assert "Login successful!" in driver.page_source  # Adjust message if needed
```

I set up Selenium to work with Chrome using its webdriver_manager, and tested for multiple testcases for my login and register forms.

Have a look at the demo!

Thank you!