

# Temporary Password Generator with Twilio

## Project Overview:

Project Name: Temporary Password Generator with Twilio

Project Duration: 05/2023 – 06/2023

By: Ranganath Krishna Kumar

## Summary:

The Temporary password generator with Twilio project aims to create a web application that generates and sends one-time passwords (OTP) to a user's phone number. The project leverages the Flask web framework and the Twilio API to provide a secure and user-friendly OTP verification process. So basically, this application can be used in a company only by admins who will have the privileges to reset a user's password.

## Project Objectives:

- Develop a Flask-based web application.
- Implement OTP generation and sending functionality using the Twilio API.
- Create a user-friendly web interface for OTP input and verification.

## Technologies used:

- Python (Programming Language)
- Flask (Web-framework)
- Twilio API
- HTML/CSS (Front-end)
- JavaScript (Programming Language)
- Visual-Studio Code (Code Editor)

## Project Description:

Flask Application Structure

The project is structured as follows:

**pwdgenflask.py:** The main Flask application file containing route definitions, OTP generation logic, and Twilio integration.

**templates/:** Directory containing HTML templates for the web pages.

**static/:** Directory for static assets like CSS and JavaScript files.

### OTP Generation and Sending:

The core functionality of the project involves generating a random OTP, associating it with a user, and sending it to their registered phone number via the Twilio API. The logic of this program is done through python. So basically, what happens is when I enter a name and an employee code it generates the OTP based on the input, it removes any white spaces, it takes 5 non-numeric characters, 3 numbers and 2 special characters and appends them to a list, then the list is jumbled and completely random, all the characters from the employee name and number are not taken only a few, then this jumbled list is converted to a string as the final output and sent as an SMS to the provided phone number using Twilio. The Twilio credentials (account SID, authentication token, and Twilio phone number) are stored securely as environment variables.

## Project Workflow:

- The user accesses the home page of the web application.
- They enter the employee's name, employee code and their phone number.
- The web application generates a random OTP and sends it to the user's phone number via Twilio SMS.

## Source code:

**app.py:**

```
import random
import re
from flask import Flask, request, render_template
import twilio
from twilio.rest import Client

account_sid = 'ACfc5fa130a484b417e270870b16eece61'
auth_token = '07fda4421c60f166562fd925bc9d0a87'
twilio_phone_number = '+15392142533'

app = Flask(__name__, template_folder='templates')
client = Client(account_sid, auth_token)

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
```

```

        name = request.form['nameInput']
        phone_number = request.form['phoneInput']
        global para_input
        para_input = name
        final_otp = generate_password()
        send_sms(final_otp, phone_number)
    return render_template('index.html')
    return """
<form method="post">
    <label for="nameInput">Enter your Name and Employee number:</label>
    <input type="text" id="nameInput" name="nameInput" required>
    <input type="submit" value="Submit">
</form>
"""

def generate_password():
    special_char = ['@', '#', '$', '%', '&']
    password = []
    capitalize_char1 = []
    pwd = ''
    upper_char = ''

    i = 0
    j = 0
    l = 0

    while i < 5:
        z = re.findall("[a-zA-Z]", para_input) # Use regular expression to
        find alphabetic characters
        if z:
            password.append(random.choice(z))
            i += 1

    while j < 3:
        x = re.findall("\d", para_input)
        if x:
            password.append(random.choice(x))
            j += 1

    while l < 2:
        password.append(random.choice(special_char))
        l += 1

    capitalize_char = capitalize_char1.append(random.choice(list(filter(None,
z))))
    uppercase = upper_char.join(capitalize_char1).upper()

    otp = pwd.join(password).replace(" ", "") + uppercase

```

```

        convert_toList = list(otp)
        random.shuffle(convert_toList)
        final_otp = ''.join(convert_toList).replace(" ", "")
        return final_otp

def send_sms(password, phone_number):
    try:
        message = client.messages.create(
            body=f"Your Generated Password: {password}",
            from_=twilio_phone_number,
            to=phone_number
        )
        print("SMS Sent Successfully!")
    except Exception as e:
        print(f"Error sending SMS: {str(e)}")
if __name__ == '__main__':
    app.run(debug=True)

```

**index.html:**

```

<!DOCTYPE html>
<html>
<head>
    <title>Password Generator</title>
    <link rel="stylesheet" type="text/css" href="{ url_for('static',
filename='styles.css') }}">
</head>
<body>
    <form id="passwordForm">
        <label for="nameInput">Enter the Name and Employee number:</label>
        <input type="text" id="nameInput" name="nameInput" required>
        <br>
        <label for="phoneInput">Enter the phone number:</label>
        <input type="text" id="phoneInput" name="phoneInput" required>
        <br>
        <input type="submit" value="Submit">
    </form>

    <div id="message"></div>

    <script>
        document.getElementById("passwordForm").addEventListener("submit",
function (event) {
            event.preventDefault();

            const nameInput = document.getElementById("nameInput").value;

```

```

const phoneInput = document.getElementById("phoneInput").value;
const formData = new FormData();
formData.append("nameInput", nameInput);
formData.append("phoneInput", phoneInput);

fetch("/", {
  method: "POST",
  body: formData
})
.then(response => response.json())
.then(data => {
  const messageDiv = document.getElementById("message");
  messageDiv.textContent = data.message;
  // Optional: If you want to display the generated password:
  // messageDiv.textContent += " " + data.password;
})
.catch(error => console.error("Error:", error));
});
</script>
</body>
</html>

```

### Styles.css:

```

@import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@100;200;300;400;500;600;700;800;900&display=swap');

*{
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: 'Poppins', sans-serif;
}

.container{
  width: 100%;
  height: 100vh;
  background: #e3ebfe;
  display: flex;
  justify-content: center;
  align-items: center;
}

button{
  width: 270px;
  height: 80px;

```

```
border: none;
outline: none;
background: #2f2f2f;
color: #fff;
font-size: 22px;
border-radius: 40px;
text-align: center;
box-shadow: 0 6px 20px -5px rgba(0,0,0,0.4);
position: relative;
overflow: hidden;
cursor: pointer;
}

.check-box{
width: 80px;
height: 80px;
border-radius: 40px;
box-shadow: 0 0 12px -2px rgba(0,0,0,0.5);
position: absolute;
top: 0;
right: -40px;
opacity: 0;
}

.check-box svg{
width: 40px;
margin: 20px;
}

svg path{
stroke-width: 3;
stroke: #fff;
stroke-dasharray: 34;
stroke-dashoffset: 34;
stroke-linecap: round;
}

.active{
background: #ff2b75;
transition: 1s;
}

.active .check-box{
right: 0;
opacity: 1;
transition: 1s;
}
```

```
.active p{
  margin-right: 125px;
  transition: 1s;
}

.active svg path{
  stroke-dashoffset: 0;
  transition: 1s;
  transition-delay: 1s;
}

html, body {
  width: 100%;
  height:100%;
}

body {
  font-family: Arial, sans-serif;
  background: linear-gradient(-45deg, #ee7752, #e73c7e, #23a6d5, #23d5ab);
  margin: 20px;
  background-size: 400% 400%;
  animation: gradient 15s ease infinite;
}

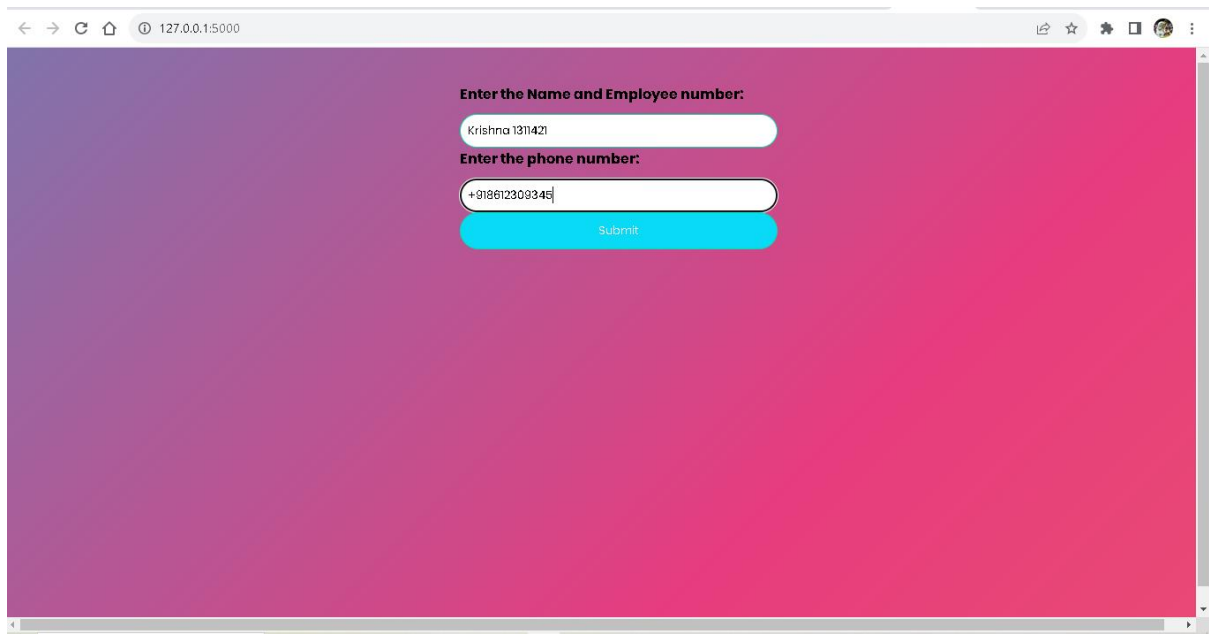
@keyframes gradient {
  0% {
    background-position: 0% 50%;
  }
  50% {
    background-position: 100% 50%;
  }
  100% {
    background-position: 0% 50%;
  }
}

form {
  max-width: 400px;
  margin: 0 auto;
  padding: 20px;
  border-radius: 5px;
}

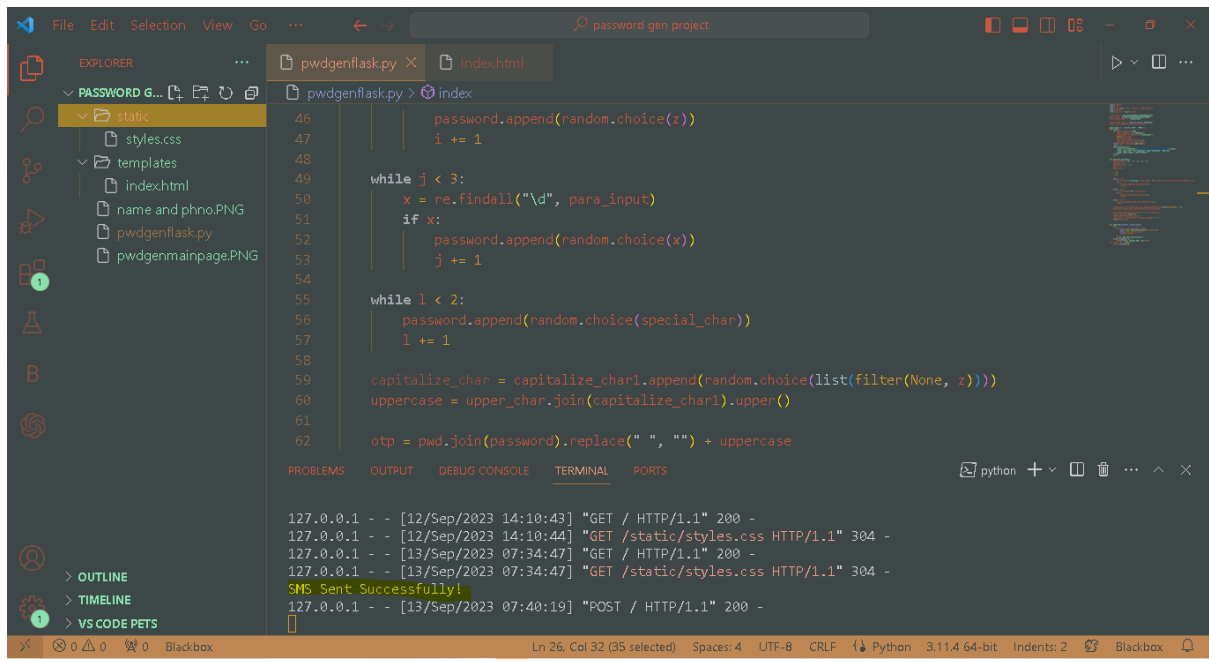
label {
  display: block;
  margin-bottom: 10px;
  font-weight: bold;
}
```

```
input[type="text"] {  
  width: 100%;  
  padding: 8px;  
  border: 1px solid #08daa9;  
  border-radius: 90px;  
}  
  
input[type="submit"] {  
  width: 100%;  
  padding: 10px;  
  background-color: #09daf5;  
  color: #fff;  
  border: 1px solid #08daa9;  
  border-radius: 90px;  
  cursor: pointer;  
}  
  
input[type="submit"]:hover {  
  background-color: #0056b3;  
}
```

## Screenshots:

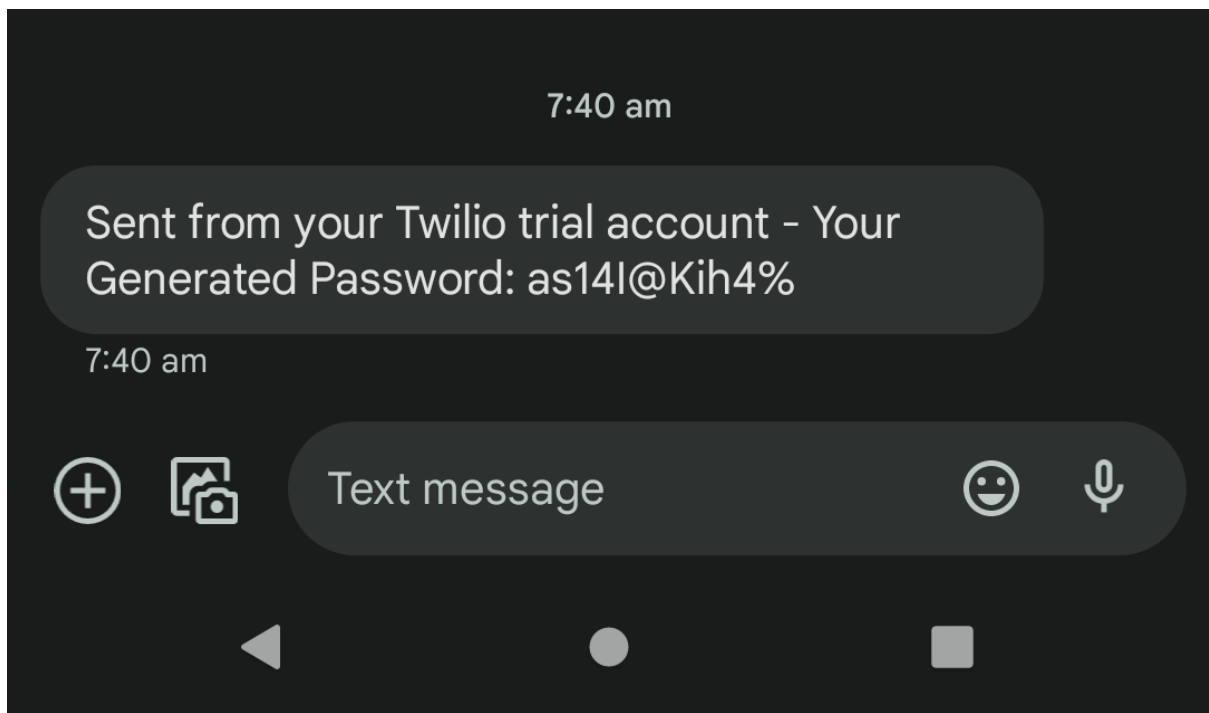






```
46 password.append(random.choice(z))
47 i += 1
48
49 while j < 3:
50     x = re.findall("\d", para_input)
51     if x:
52         password.append(random.choice(x))
53         j += 1
54
55 while l < 2:
56     password.append(random.choice(special_char))
57     l += 1
58
59 capitalize_char = capitalize_char1.append(random.choice(list(filter(None, z))))
60 uppercase = upper_char.join(capitalize_char1).upper()
61
62 otp = pwd.join(password).replace(" ", "") + uppercase
```

127.0.0.1 - - [12/Sep/2023 14:10:43] "GET / HTTP/1.1" 200 -  
127.0.0.1 - - [12/Sep/2023 14:10:44] "GET /static/styles.css HTTP/1.1" 304 -  
127.0.0.1 - - [13/Sep/2023 07:34:47] "GET / HTTP/1.1" 200 -  
127.0.0.1 - - [13/Sep/2023 07:34:47] "GET /static/styles.css HTTP/1.1" 304 -  
**SMS Sent Successfully!**  
127.0.0.1 - - [13/Sep/2023 07:40:19] "POST / HTTP/1.1" 200 -



## Challenges Faced:

**Twilio Integration:** Configuring and integrating the Twilio API for SMS sending required careful handling of API keys and credentials.

**Error Handling:** Handling various error scenarios, such as network errors, Twilio API errors, and invalid OTP inputs, is important to provide clear feedback to users.

**Rate Limiting:** Twilio may impose rate limits on the number of SMS messages you can send within a certain time frame. Handling rate limiting errors and ensuring a smooth user experience can be challenging.

## Lessons Learned:

- Improved understanding of Flask web development.
- Enhanced knowledge of Twilio API integration.
- Experience with handling sensitive data and credentials securely.

## Future Enhancements:

- Implement user authentication and user account management.
- Add a time-limit for the expiration of the OTP.
- Enhance the user interface with more styling and responsiveness.

## Conclusion:

This project successfully achieves its objective of providing a secure and user-friendly OTP verification process. It serves as a foundation for further enhancements and can be extended to support more advanced features in the future.