

Реализация слайдер-CAPTCHA в веб-приложении на Flask

Ран Максим Александрович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В работе представлена реализация веб-приложения на Python с использованием фреймворка Flask и базы данных SQLite, в котором механизм аутентификации пользователей защищён с помощью слайдер-CAPTCHA. В отличие от классических текстовых CAPTCHA, данный подход основан на совмещении вырезанного фрагмента изображения с соответствующим «пустым» участком фона посредством графического ползунка. Описываются архитектура приложения, алгоритм генерации капчи, механизмы сохранения токена и решения в сессии, а также особенности проверки корректности смещения. Показано, что выбранный подход обладает лучшей удобством использования по сравнению с текстовой CAPTCHA и обеспечивает достаточную устойчивость к базовым сценариям автоматизированных атак.

Ключевые слова: CAPTCHA, слайдер-CAPTCHA, Flask, Python, SQLite, защита от ботов, веб-приложения, аутентификация, сессии, обработка изображений.

Creating a slider CAPTCHA in a Flask web application

Ran Maksim Aleksandrovich

Sholom-Aleichem Priamursky State University

Student

Abstract

This paper presents the implementation of a Flask-based web application using an SQLite database, where user authentication is protected by a slider CAPTCHA. Unlike traditional text-based CAPTCHA, the proposed approach relies on aligning a cut-out image fragment with a corresponding “hole” on the background using a graphical slider. The paper describes the application architecture, CAPTCHA generation algorithm, session-based token and solution storage, as well as the verification procedure. It is shown that the solution provides better usability compared to classic text CAPTCHA while maintaining sufficient resistance to basic automated attacks.

Keywords: CAPTCHA, slider CAPTCHA, Flask, Python, SQLite, bot protection, web application, authentication, sessions, image processing.

1 Введение

1.1 Актуальность

С ростом количества онлайн-сервисов и веб-приложений увеличивается и число автоматизированных ботов, нацеленных на массовую регистрацию аккаунтов, подбор паролей, рассылку спама и другие виды злоупотреблений. Одним из базовых механизмов защиты от подобных атак остаётся CAPTCHA — тест, позволяющий отличить действия человека от действий программы.

Классические текстовые CAPTCHA, основанные на вводе искажённых символов с картинки, постепенно теряют эффективность. Современные алгоритмы оптического распознавания символов и методы машинного обучения успешно справляются с такими изображениями, особенно если разработчики стремятся сохранять их удобочитаемость для пользователей. В результате приходится искать баланс между удобством и безопасностью либо переходить к альтернативным форматам тестов.

Одним из таких форматов является слайдер-CAPTCHA, где пользователю предлагается совместить вырезанный фрагмент изображения с «дыркой» на фоне путём перемещения ползунка. Такой тест воспринимается интуитивно понятным для человека, не требует ввода текста и в то же время сложнее для простых скриптов, ориентированных только на распознавание символов.

1.2 Обзор исследований

Ранее проблематика противодействия атакам типа brute force и использования графических объектов CAPTCHA исследовалась целым рядом авторов. Так, А.В. Яковлев, А.М. Шуваева и М.А. Пеливан в работе «Противодействие атаке brute force на основе графических объектов типа CAPTCHA» [1] рассматривают различные разновидности CAPTCHA, анализируют их сложность для преодоления автоматизированными системами и предлагают подходы к использованию CAPTCHA для повышения устойчивости парольной аутентификации к атакам грубой силы. О.Н. Жданов в исследовании «Распознавание современных CAPTCHA» [2] подробно анализирует задачу автоматизированного распознавания современных типов CAPTCHA. В рамках этой работы автор выделяет три ключевых этапа: предварительную обработку изображений, сегментацию, а также распознавание отдельных символов или элементов CAPTCHA. Кроме того, М.П. Явич и И.З. Пирцхалава в статье «Проблемы развития CAPTCHA» [3] предлагают различные варианты CAPTCHA и обсуждают направления их усложнения в ответ на появление более мощных ботов, способных обходить традиционные системы защиты. В работе М. Кумар, М.К. Джиндал и М. Кумар (2018) представлен систематический обзор различных типов CAPTCHA, включая текстовые, изображенческие, аудио- и видео-CAPTCHA, с оценкой их стойкости и удобства использования. Рассматриваются 16 методов генерации текстовых CAPTCHA на разных языках с применением разнообразных техник деформации, которые были успешно взломаны с использованием алгоритмов машинного обучения (CNN, KNN, SVM).

Анализируются также изображения и аудио CAPTCHA, которые, несмотря на высокую степень защищённости, подвержены атакам на основе аналогичных методов. Отдельно рассматриваются видео CAPTCHA, демонстрирующие высокие показатели безопасности (98–100%), но при этом имеющие уязвимости, позволяющие осуществлять успешный взлом с использованием SIFT, NN и простого OCR [4]. Данный обзор подчёркивает необходимость разработки более сложных и устойчивых CAPTCHA-систем для противодействия современным автоматическим атакам. В статье Н. Тарик, Ф.А. Хан, С.А. Мокурраб и Г. Шривастава (2021) проводится обзор механизма CAPTCHA как средства аутентификации пользователей и защиты от вредоносных ботов. Авторы рассматривают различные типы CAPTCHA, их достоинства, недостатки и дают рекомендации по проектированию подобных систем. Подробно анализируются методы обхода CAPTCHA с применением алгоритмов машинного обучения и нейронных сетей (CNN, KNN, SVM). Обзор, основанный более чем на 200 исследованиях, акцентирует внимание на необходимости создания более устойчивых и удобных CAPTCHA-систем, а также выделяет нерешённые проблемы, сохраняющиеся в области их проектирования [5].

1.3 Цель исследования

Целью данной работы является разработка и описание небольшого веб-приложения на Flask с реализованным механизмом слайдер-CAPTCHA для форм регистрации и входа. Основное внимание уделяется:

- алгоритму генерации изображений CAPTCHA (фон с «дыркой» и вырезанный фрагмент);
- механизму передачи данных на фронтенд и их представлению в интерфейсе;
- процедуре проверки корректности смещения фрагмента относительно исходного решения;
- особенностям интеграции капчи в процесс аутентификации пользователей.

2 Материалы и методы

Для реализации приложения использовались следующие технологии:

- язык программирования Python;
- веб-фреймворк Flask для организации маршрутов и обработки HTTP-запросов;
- база данных SQLite для хранения информации о пользователях (логин, e-mail, хеш пароля, дата регистрации);
- стандартные средства Werkzeug для хеширования паролей;
- лёгкий CSS-фреймворк PicoCSS для оформления интерфейса;
- библиотека работы с изображениями (например, Pillow) для генерации фонового изображения и вырезанного фрагмента;
- механизм сессий Flask для хранения токена и решения CAPTCHA.

Данные пользователей сохраняются в файле `users.db` в корне проекта. Таблица `users` содержит поля `id`, `username`, `email`, `password_hash`, `created_at`. Работа с БД реализована через контекстные менеджеры, что упрощает управление соединением и обработку ошибок.

3 Результаты и обсуждения

Начнём исследование с создания файла `app.py`, в котором размещён основной код веб-приложения на Flask, а также вся логика регистрации, авторизации и работы с слайдер-CAPTCHA.

Python-часть — обработка запросов и проверка CAPTCHA

Импорт модулей и задание констант.

В начале файла подключаются стандартные модули Python (`os`, `sqlite3`, `datetime`, `random`, `secrets`, `base64`, `io`), компоненты фреймворка Flask (`Flask`, `request`, `session`, `flash`, `redirect`, `url_for`, `render_template`, `jsonify`), библиотека PIL (`Image`, `ImageDraw`, `ImageFilter`) для работы с изображениями и функции `generate_password_hash/check_password_hash` для безопасной работы с паролями (рис.1).

```
import base64
import io
import os
import random
import secrets
import sqlite3
from datetime import datetime

from flask import (
    Flask,
    flash,
    jsonify,
    redirect,
    render_template,
    request,
    session,
    url_for,
)
from PIL import Image, ImageDraw, ImageFilter
from werkzeug.security import check_password_hash, generate_password_hash
```

Рисунок 1. Импорт модулей

Далее задаются константы: путь к базе данных `DATABASE_PATH`, ключ для хранения капч в сессии `CAPTCHA_SESSION_KEY`, размеры фонового изображения `CAPTCHA_WIDTH/CAPTCHA_HEIGHT`, размер вырезаемого фрагмента `CAPTCHA_PIECE` и допустимое отклонение `CAPTCHA_TOLERANCE` в пикселях (рис.2).

```
DATABASE_PATH = "users.db"
CAPTCHA_SESSION_KEY = "captchas"
CAPTCHA_WIDTH = 320
CAPTCHA_HEIGHT = 180
CAPTCHA_PIECE = 56
CAPTCHA_TOLERANCE = 6
```

Рисунок 2. Задание констант

Создание приложения Flask и инициализация базы данных.

Функция `create_app()` создаёт экземпляр приложения Flask, задаёт `SECRET_KEY` (либо из переменных окружения, либо тестовый) и вызывает `initialize_database()` (рис.3).

```
def create_app():
    app = Flask(__name__)
    app.config["SECRET_KEY"] = os.environ.get("SECRET_KEY", "change-me")

    initialize_database()
```

Рисунок 3. Код

В `initialize_database()` создаётся (при отсутствии) таблица `users` в базе SQLite с полями `id`, `username`, `email`, `password_hash`, `created_at`. Таким образом, при первом запуске все необходимые структуры данных формируются автоматически (рис.4).

```
def initialize_database():
    with sqlite3.connect(DATABASE_PATH) as conn:
        conn.execute(
            """
            CREATE TABLE IF NOT EXISTS users (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                username TEXT UNIQUE NOT NULL,
                email TEXT UNIQUE NOT NULL,
                password_hash TEXT NOT NULL,
                created_at TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP
            )
            """
        )
        conn.commit()
```

Рисунок 4. Код

Подготовка хранилища капч в сессии.

Декоратор `@app.before_request` вызывает функцию `ensure_captcha_store()`, которая гарантирует, что в объекте `session` есть словарь по ключу `CAPTCHA_SESSION_KEY`. В этот словарь будут складываться данные о выданных капчах для разных форм (`login`, `register` и т.д.) (рис.5).

```
@app.before_request
def ensure_captcha_store():
    session.setdefault(CAPTCHA_SESSION_KEY, {})
```

Рисунок 5. Код

Маршрут /register: регистрация пользователя (рис.6).

```
@app.route("/register", methods=["GET", "POST"])
def register():
    if request.method == "POST":
        form = request.form
        username = form.get("username", "").strip()
        email = form.get("email", "").strip().lower()
        password = form.get("password", "")
        confirm_password = form.get("confirm_password", "")
        captcha_offset = form.get("captcha_offset", "")
        captcha_token = form.get("captcha_token", "")

        # Простые проверки формы
        if not username or not email or not password:
            flash("Заполните имя, email и пароль", "error")
        elif password != confirm_password:
            flash("Пароли не совпадают", "error")
        elif not validate_captcha("register", captcha_offset, captcha_token):
            flash("Капча не пройдена", "error")
        elif user_exists(username, email):
            flash("Такой логин или email уже используются", "error")
        else:
            create_user(username, email, password)
            flash("Регистрация прошла успешно. Теперь войдите.", "success")
            session.get(CAPTCHA_SESSION_KEY, {}).pop("register", None)
            return redirect(url_for("login"))

    return render_template("register.html")
```

Рисунок 6. Код

При запросе GET отображается шаблон с формой регистрации.

При POST из request.form извлекаются: логин, e-mail, пароль, подтверждение пароля, а также пара значений, связанных с капчей — captcha_offset и captcha_token.

Далее последовательно выполняются проверки:

- заполнены ли логин, e-mail и пароль;
- совпадают ли пароль и подтверждение;
- успешно ли пройдена капча (validate_captcha("register", captcha_offset, captcha_token)) (рис.7);

```
def validate_captcha(form_name: str, offset_value: str, token: str) -> bool:
    stored = session.get(CAPTCHA_SESSION_KEY, {}).get(form_name)
    if not stored or stored.get("token") != token:
        return False

    try:
        offset = float(offset_value)
    except (TypeError, ValueError):
        return False

    solution = float(stored.get("solution", 0))
    tolerance = float(stored.get("tolerance", CAPTCHA_TOLERANCE))
    return abs(offset - solution) <= tolerance
```

Рисунок 7. Код

- не существуют ли уже в базе пользователь с таким логином или e-mail (user_exists) (рис.8).

```
def user_exists(username: str, email: str) -> bool:
    query = "SELECT 1 FROM users WHERE username = ? OR email = ? LIMIT 1"
    with sqlite3.connect(DATABASE_PATH) as conn:
        cursor = conn.execute(query, (username, email))
        return cursor.fetchone() is not None
```

Рисунок 8. Код

Если все условия соблюдены, вызывается функция create_user(), которая хеширует пароль и записывает нового пользователя в таблицу users (рис.9). После успешной регистрации пользователю выводится сообщение и предлагается войти в систему. Для формы register старые данные капчи из сессии удаляются, чтобы исключить повторное использование.

```
def create_user(username: str, email: str, password: str) -> None:
    hashed_password = generate_password_hash(password)
    with sqlite3.connect(DATABASE_PATH) as conn:
        conn.execute(
            "INSERT INTO users (username, email, password_hash) VALUES (?, ?, ?)",
            (username, email, hashed_password),
        )
        conn.commit()
```

Рисунок 9. Код

Маршрут /login: вход в систему (рис.10).

```
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        username = request.form.get("username", "").strip()
        password = request.form.get("password", "")
        captcha_offset = request.form.get("captcha_offset", "")
        captcha_token = request.form.get("captcha_token", "")

        user = fetch_user(username)
        if not user or not check_password_hash(user["password_hash"], password):
            flash("Неверное имя или пароль", "error")
        elif not validate_captcha("login", captcha_offset, captcha_token):
            flash("Капча не пройдена", "error")
        else:
            session["user_id"] = user["id"]
            session["username"] = user["username"]
            session["login_time"] = datetime.utcnow().isoformat()
            session.get(CAPTCHA_SESSION_KEY, {}).pop("login", None)
            return redirect(url_for("dashboard"))

    return render_template("login.html")
```

Рисунок 10. Код

Маршрут устроен аналогично регистрации: при GET показывается форма входа, при POST выполняются проверки.

- По введённому логину функция `fetch_user()` выбирает пользователя из базы SQLite (рис.11).

```
def fetch_user(username: str):
    with sqlite3.connect(DATABASE_PATH) as conn:
        conn.row_factory = sqlite3.Row
        cursor = conn.execute(
            "SELECT id, username, password_hash FROM users WHERE username = ?",
            (username,)
        )
        return cursor.fetchone()
```

Рисунок 11. Код

- Пароль сравнивается с помощью `check_password_hash`.
- Если логин и пароль корректны, дополнительно проверяется капча: `validate_captcha("login", captcha_offset, captcha_token)`.

При успешном прохождении всех проверок в `session` сохраняются идентификатор пользователя, имя и время входа. После этого происходит перенаправление на защищённую страницу `/dashboard`.

Маршруты `/dashboard`, `/logout` и `/`.

`/dashboard` — простая защищённая страница. Если в сессии нет `user_id`, пользователь перенаправляется на `/login` (рис.12).

```
@app.route("/dashboard")
def dashboard():
    if not session.get("user_id"):
        return redirect(url_for("login"))

    return render_template(
        "dashboard.html",
        username=session.get("username"),
        login_time=session.get("login_time"),
    )
```

Рисунок 12. Код

/logout очищает сессию и возвращает пользователя на страницу входа (рис.13).

```
@app.route("/logout")
def logout():
    session.clear()
    flash("Вы вышли из аккаунта", "info")
    return redirect(url_for("login"))
```

Рисунок 13. Код

/ перенаправляет либо на /dashboard (если уже авторизован), либо на /login (рис.14).

```
@app.route("/")
def index():
    if session.get("user_id"):
        return redirect(url_for("dashboard"))
    return redirect(url_for("login"))
```

Рисунок 14. Код

HTML- и JavaScript-часть — вывод слайдер-CAPTCHA

В шаблонах register.html (рис.15) и login.html (рис.16) помимо стандартных полей (логин, e-mail, пароль) располагается блок со слайдер-CAPTCHA.

```
{% extends "base.html" %}
{% set title = "Регистрация" %}

{% block content %}
<form method="post">
  <label>
    Имя пользователя
    <input type="text" name="username" required>
  </label>
  <label>
    Email
    <input type="email" name="email" required>
  </label>
  <label>
    Пароль
    <input type="password" name="password" required minlength="6">
  </label>
  <label>
    Подтверждение пароля
    <input type="password" name="confirm_password" required minlength="6">
  </label>
  <label>
    Перетащите ползунок, чтобы собрать пазл
    <div class="slider-captcha" data-form="register">
      <div class="puzzle-stage">
        <img class="captcha-bg" alt="Фон капчи">
        <img class="captcha-piece" alt="Фрагмент капчи">
      </div>
      <input type="range" class="captcha-slider" min="0" value="0" step="1">
      <div class="captcha-actions">
        <button type="button" class="captcha-refresh">Обновить</button>
        <span>Совместите фрагмент  пустым местом</span>
      </div>
      <input type="hidden" name="captcha_offset" value="0">
      <input type="hidden" name="captcha_token" value="">
    </div>
  </label>
  <button type="submit">Создать аккаунт</button>
  <p>Уже входили? <a href="{{ url_for('login') }}">Войти</a></p>
</form>
{% endblock %}
```

Рисунок 15. Код регистрации

```
{% extends "base.html" %}
{% set title = "Вход" %}

{% block content %}
<form method="post">
  <label>
    Имя пользователя
    <input type="text" name="username" required>
  </label>
  <label>
    Пароль
    <input type="password" name="password" required>
  </label>
  <label>
    Сдвиньте пазл, чтобы подтвердить, что вы человек
    <div class="slider-captcha" data-form="login">
      <div class="puzzle-stage">
        <img class="captcha-bg" alt="Фон капчи">
        <img class="captcha-piece" alt="Фрагмент капчи">
      </div>
      <input type="range" class="captcha-slider" min="0" value="0" step="1">
      <div class="captcha-actions">
        <button type="button" class="captcha-refresh">Обновить</button>
        <span>Двигайте ползунок, пока фрагмент не встанет на место</span>
      </div>
      <input type="hidden" name="captcha_offset" value="0">
      <input type="hidden" name="captcha_token" value="">
    </div>
  </label>
  <button type="submit">Войти</button>
  <p>Нет аккаунта? <a href="{{ url_for('register') }}">Зарегистрироваться</a></p>
</form>
{% endblock %}
```

Рисунок 16. Код авторизации

- Структура блока CAPTCHA.
- Фоновое изображение с «дыркой» под вырезанный фрагмент.
- Сам вырезанный фрагмент, который будет двигаться по горизонтали.
- Ползунок (slider), управляющий смещением фрагмента.
- Скрытые поля формы captcha_offset и captcha_token, в которые JavaScript записывает текущее смещение и выданный сервером токен.

При загрузке страницы JavaScript отправляет AJAX-запрос к маршруту /captcha?form=login или /captcha?form=register. В ответ сервер возвращает JSON с параметрами капчи, и скрипт подставляет полученные изображения и значения на страницу.

Генерация слайдер-CAPTCHA на сервере

Основная логика формирования капчи сосредоточена в функциях build_slider_captcha() (рис.17) и generate_captcha_for() (рис.18).

```
def build_slider_captcha() -> dict:
    width, height, piece_size = CAPTCHA_WIDTH, CAPTCHA_HEIGHT, CAPTCHA_PIECE
    base = Image.new("RGB", (width, height), (244, 247, 252))
    draw = ImageDraw.Draw(base)
    for _ in range(12):
        color = (
            random.randint(120, 180),
            random.randint(150, 210),
            random.randint(160, 230),
        )
        start = (random.randint(0, width), random.randint(0, height))
        end = (random.randint(0, width), random.randint(0, height))
        draw.line([start, end], fill=color, width=random.randint(1, 3))
    for _ in range(6):
        radius = random.randint(12, 28)
        color = (
            random.randint(160, 220),
            random.randint(120, 190),
            random.randint(160, 230),
        )
        center_x = random.randint(radius, width - radius)
        center_y = random.randint(radius, height - radius)
        draw.ellipse(
            (center_x - radius, center_y - radius, center_x + radius, center_y + radius),
            fill=color,
        )
    base = base.filter(ImageFilter.SMOOTH)
    target_x = random.randint(piece_size + 10, width - piece_size - 10)
    target_y = random.randint(20, height - piece_size - 20)
    piece = base.crop((target_x, target_y, target_x + piece_size, target_y + piece_size))
    background = base.copy()
    hole = ImageDraw.Draw(background)
    hole.rounded_rectangle(
        (target_x, target_y, target_x + piece_size, target_y + piece_size),
        radius=10,
        fill=(210, 218, 233),
    )
    piece_rgba = Image.new("RGBA", (piece_size, piece_size), (0, 0, 0, 0))
    piece_rgba.paste(piece, (0, 0))
    outline = ImageDraw.Draw(piece_rgba)
    outline.rounded_rectangle(
        (0, 0, piece_size - 1, piece_size - 1),
        radius=10,
        outline=(255, 255, 255, 220),
        width=2,
    )
    return {
        "background": encode_image_to_data_uri(background),
        "piece": encode_image_to_data_uri(piece_rgba),
        "piece_y": target_y,
        "track length": width - piece_size.
```

Рисунок 17. Функция создания слайдер-капчи

```
def generate_captcha_for(form_name: str) -> dict:
    slider = build_slider_captcha()
    token = secrets.token_urlsafe(16)
    captchas = session.setdefault(CAPTCHA_SESSION_KEY, {})
    captchas[form_name] = {
        "token": token,
        "solution": slider["solution"],
        "tolerance": slider["tolerance"],
    }
    session[CAPTCHA_SESSION_KEY] = captchas
    return {
        "token": token,
        "background": slider["background"],
        "piece": slider["piece"],
        "piece_y": slider["piece_y"],
        "track_length": slider["track_length"],
    }
```

Рисунок 18. Функция генерации капчи

Функция `build_slider_captcha()`: формирование изображений.

1. Создание фонового изображения и случайных элементов.

Создаётся изображение нужного размера с мягким фоновым цветом. С помощью `ImageDraw.Draw` поверх фона рисуются случайные линии и круглые элементы (эллипсы) разных цветов и размеров. Это формирует уникальный абстрактный фон и усложняет анализ изображения автоматическими алгоритмами. Затем изображение слегка сглаживается фильтром `ImageFilter.SMOOTH`.

2. Выбор координат вырезаемого фрагмента.

Случайным образом выбираются координаты `target_x` и `target_y` так, чтобы квадратный фрагмент размером `CAPTCHA_PIECE` полностью помещался внутри фонового изображения и не касался краёв.

3. Вырезание фрагмента и создание «дырки».

По выбранным координатам из базового изображения вырезается квадратный фрагмент (`piece`). Фон копируется во второе изображение (`background`), где на месте вырезанного фрагмента рисуется закруглённый прямоугольник приглушённого цвета. Визуально это выглядит как «паз», в который нужно вставить кусочек.

4. Подготовка вырезанного фрагмента с прозрачным фоном.

Создаётся новое RGBA-изображение того же размера, куда помещается вырезанный фрагмент. Сверху по контуру рисуется светлая рамка с закруглёнными углами, чтобы визуально обозначить границы фрагмента.

5. Формирование служебных данных.

Функция возвращает словарь с:

- background — фоновым изображением, закодированным в PNG и base64 в виде data-URI;
- piece — вырезанным фрагментом в таком же формате;
- piece_y — вертикальной координатой фрагмента;
- track_length — длиной трека, по которому будет двигаться слайдер;
- solution — точным значением правильного смещения по оси X;
- tolerance — допустимым отклонением.

Функция generate_captcha_for(form_name): сохранение решения в сессии.

Эта функция вызывает build_slider_captcha(), генерирует уникальный токен с помощью secrets.token_urlsafe(16) и сохраняет в сессии по ключу CAPTCHA_SESSION_KEY запись вида:

- token — уникальный идентификатор данной попытки;
- solution — правильное смещение фрагмента;
- tolerance — допуск по расстоянию.

Пользователю возвращается JSON с полями token, background, piece, piece_y, track_length. Эти данные используются фронтендом для отображения и анимации слайдера.

Маршрут /captcha отвечает за выдачу этих данных и запрещает кеширование с помощью заголовков Cache-Control (рис.19).

```
@app.route("/captcha")
def captcha():
    form_name = request.args.get("form", "general")
    data = generate_captcha_for(form_name)
    response = jsonify(data)
    response.headers.update(
        {"Cache-Control": "no-store, no-cache, must-revalidate, max-age=0"}
    )
    return response
```

Рисунок 19. Код

Внешний вид веб-приложения.

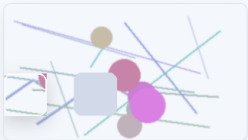
1. Внешний вид окна авторизации пользователя (рис. 20).

Вход

Имя пользователя

Пароль

Сдвиньте пазл, чтобы подтвердить, что вы человек



Обновить

Двигайте ползунок, пока фрагмент не встанет на место

Войти

Нет аккаунта? [Зарегистрироваться](#)

Рисунок 20. Экран входа

2. Внешний вид окна регистрации пользователя (рис.21).

Регистрация

Имя пользователя

Email

Пароль

Подтверждение пароля

Перетащите ползунок, чтобы собрать пазл



Обновить

Совместите фрагмент с пустым местом

Создать аккаунт

Уже входили? [Войти](#)

Рисунок 21. Экран регистрации

4 Выводы

Реализованный в файле app.py механизм слайдер-CAPTCHA обеспечивает защиту форм регистрации и входа от простых автоматизированных ботов. Использование сессий для хранения решений, генерация случайного фона и фрагментов, а также передача изображений в формате data-URI позволяют создать динамическую и удобную для пользователя систему проверки, аналогичную по структуре ранее рассмотренной реализации на PHP, но уже на стеке Flask + SQLite + Pillow.

Библиографический список

1. Яковлев А.В., Шуваева А.М., Пеливан М.А. Противодействие атаке brute force на основе графических объектов типа CAPTCHA // Приборы и системы. управление, контроль, диагностика. 2015. №7. С. 15-19.
2. Жданов О.Н. Распознавание современных CAPTCHA // Научный вестник Воронежского государственного архитектурно-строительного университета. Серия: студент и наука. 2014. С. 221-224.
3. Явич М.П. Пирцхалава И.З. Проблемы развития CAPTCHA // Современная техника и технологии. 2015. №7. С. 92-93.
4. Kumar M., Jindal M.K., Kumar M. A Systematic Survey on CAPTCHA Recognition: Types, Creation and Breaking Techniques // Archives of Computational Methods in Engineering State of the Art Reviews. 2021. Т. 29. С. 1107-1136.
5. Tariq N., Khan F.A., Moqurrah S.A., Srivastava G. CAPTCHA Types and Breaking Techniques: Design Issues, Challenges, and Future Research Directions // Cryptography and Security. 2023.