Capture Barrierefrei

Eine barrierefreie Bot-Erkennungs- und Formularvalidierungslösung mit integrierter Formularerkennung und E-Mail-Versand-System.

Inhaltsverzeichnis

Einführung

Capture Barrierefrei ist eine Alternative zu herkömmlichen CAPTCHA-Systemen, die barrierefrei gestaltet wurde. Das Projekt bietet ein Grundgerüst für eine agnostische Lösung zum Schutz von Webformularen vor Spam-Bots und automatisierten Angriffen, während es gleichzeitig die Zugänglichkeit für alle Nutzer, einschließlich solcher mit Beeinträchtigungen, gewährleistet.

Projektübersicht

Die Lösung besteht aus zwei Hauptmodulen:

1. **CaptureBarriereFrei**: Erkennt automatisierte Bot-Zugriffe durch Verhaltensanalyse
2. **MailSender**: Verarbeitet Formularübermittlungen und E-Mail-Versand mit Templateunterstützung

Beide Module sind so konzipiert, dass sie den Prinzipien der Barrierefreiheit entsprechen und mit Screenreadern sowie anderen assistiven Technologien kompatibel sind.

Architektur

Modularer Aufbau

Die Anwendung folgt einem modularen Design-Ansatz mit klar getrennten Verantwortlichkeiten:

```
├── assets/
│   ├── js/
│   │   ├── captureBarriereFrei/    # Bot-Erkennungs-Modul
│   │   │   ├── index.js            # Einstiegspunkt
│   │   │   ├── core.js             # Kernfunktionalität
│   │   │   ├── botDetection.js     # Bot-Erkennungslogik
│   │   │   ├── formProtection.js   # Formularschutz
│   │   │   ├── formValidation.js   # Validierungsfunktionen
│   │   │   ├── ui.js               # UI-Komponenten
│   │   │   ├── utils.js            # Hilfsfunktionen
│   │   │   └── logger.js           # Logging-Funktionalität
│   │   ├── mailSender/             # E-Mail-Versand-Modul
│   │   │   ├── index.js            # Einstiegspunkt
│   │   │   ├── core/               # Kernfunktionalität
│   │   │   └── utils/              # Hilfsfunktionen
│   │   └── config.js               # Zentrale Konfiguration
│   ├── css/                        # Stylesheet-Dateien
│   ├── php/                        # Server-seitige Skripte
│   └── templates/                  # E-Mail-Vorlagen
└── index.html                      # Beispiel-Implementation
```

Sicherheitskonzept

Das Sicherheitskonzept von Capture Barrierefrei basiert auf einem mehrstufigen Ansatz:

1. **Mehrere Prüfebenen**: Anstatt sich auf einen einzelnen Sicherheitsmechanismus zu verlassen, kombiniert das System verschiedene Techniken:
    - Verhaltensanalyse (Mausbewegungen, Tastatureingaben)
    - Honeypot-Felder (für Bots unsichtbare Fallen)
    - Zeitanalyse der Formularausfüllung
    - Heuristische Bewertung von Interaktionsmustern
    - Human-Verification-Checkbox
2. **Punkte-basiertes Scoring-System**: Jede Interaktion des Benutzers wird bewertet und trägt zu einem Sicherheits-Score bei, der entscheidet, ob die Formularübermittlung erlaubt wird.
3. **Kontinuierliche Überwachung**: Das System bewertet Benutzerinteraktionen während der gesamten Session und aktualisiert den Sicherheits-Score dynamisch.

Diese mehrschichtige Architektur macht das System resistenter gegen Angriffe, da ein Bot alle Sicherheitsebenen überwinden müsste, um erfolgreich zu sein.

Module

CaptureBarriereFrei

Das CaptureBarriereFrei-Modul ist verantwortlich für:

- Erkennung von automatisierten Bot-Zugriffen
- Schutz von Formularen
- Validierung von Benutzereingaben
- Bereitstellung barrierefreier UI-Komponenten

Kernfunktionen:

```
class CaptureBarriereFrei {
    constructor(options = {}) {
        // Konfiguration initialisieren
        this.config = {
            autoProtect: true,
            formSelectors: 'form',
            botScoreFieldName: 'security-score',
            startTimeFieldName: 'interaction-start-time',
            honeypotFieldName: 'url-field',
            minTimeToFill: 1800,
            thresholdScore: 5,
            enableLogging: false,
            ...options
        };

        // Benutzerinteraktionen verfolgen
        this.interactions = {
            mouseEvents: 0,
            keyboardEvents: 0,
            scrollEvents: 0,
            interactionStarted: false,
            securedForms: new Map()
        };
    }
    // ...weitere Methoden
}
```

MailSender

Das MailSender-Modul übernimmt:

- Verarbeitung von Formularübermittlungen
- Validierung von Dateianlagen
- Anwendung von E-Mail-Templates
- Kommunikation mit dem Backend

Kernfunktionen:

```
class MailSender {
    constructor(config = {}) {
        // Konfigurationsmanager initialisieren
        this.configManager = new ConfigManager(config);
        this.config = this.configManager.getConfig();

        // Template-Konfiguration
        this.templateConfig = null;
    }

    // Verarbeitet ein Formular-Submit-Event
    handleSubmit(event) {
        // Verhindert Standardverhalten
        event.preventDefault();

        // Formular verarbeiten und E-Mail senden
        // ...
    }

    // Weitere Methoden für Template-Anwendung, E-Mail-Versand etc.
}
```

Implementierungsdetails

Bot-Erkennung

Die Bot-Erkennung basiert auf einer Analyse mehrerer Verhaltensindikatoren:

1. **Mausbewegungen**: Natürliche Mausbewegungen sind unregelmäßig und haben verschiedene Geschwindigkeiten.
2. **Tastatureingaben**: Menschen tippen mit unterschiedlichen Geschwindigkeiten und machen Pausen.
3. **Scrollverhalten**: Menschen scrollen meist nicht mit konstanter Geschwindigkeit.
4. **Interaktionsbalance**: Menschen nutzen sowohl Maus als auch Tastatur in einem relativ ausgewogenen Verhältnis.

Beispiel-Scoring:

```
// Auszug aus analyzeBotBehavior in botDetection.js
let score = 0;

// Natürliche Mausbewegungen bewerten (0-8 Punkte)
if (this.interactions.mouseEvents > 4) {
    score += 8;
}

// Tastaturaktivität bewerten (0-12 Punkte)
if (this.interactions.keyboardEvents > 3) {
    score += 12;
}

// Scrollverhalten bewerten (0-7 Punkte)
if (this.interactions.scrollEvents > 0) {
    score += 7;
}

// "Ich bin kein Roboter" Checkbox-Status prüfen (0-25 Punkte)
if (formConfig.humanVerification &&
formConfig.humanVerification.checkbox.checked) {
    score += 25;
}

// Verhältnis zwischen Maus- und Tastaturaktivität (0-5 Bonus-Punkte)
const interactionRatio = this.interactions.mouseEvents /
(this.interactions.keyboardEvents || 1);
if (interactionRatio > 0.3 && interactionRatio < 3) {
    score += 5; // Bonus für ausgewogenes Verhalten
}
```

Rechenbeispiel:

- Ein echter Mensch hat typischerweise: 10 Mausbewegungen (+8), 8 Tastatureingaben (+12), 3 Scrollereignisse (+7), bestätigte Checkbox (+25) → Gesamtscore: 52 (weit über dem Schwellenwert

von 5)
- Ein einfacher Bot hat: 0 Mausbewegungen (+0), 0 Tastatureingaben (+0), 0 Scrollereignisse (+0), könnte die Checkbox automatisch anklicken (+25) → Gesamtscore: 25 (bestenfalls)
- Ein ausgeklügelter Bot müsste Mausbewegungen, Tastatureingaben und Scrollverhalten in einem menschenähnlichen Muster simulieren, was deutlich schwieriger ist als herkömmliche CAPTCHAs zu umgehen.

Fallback-Verhalten bei nicht erreichtem Score:

Wenn der Sicherheits-Score den konfigurierten Schwellenwert (standardmäßig 5) nicht erreicht, werden folgende Maßnahmen ergriffen:

1. **Formularsperre**: Die Formularübermittlung wird blockiert und ein Submit-Event kann nicht erfolgreich durchgeführt werden.

2. **Benutzerfeedback**: Eine visuelle und barrierefreie Rückmeldung informiert den Benutzer, dass weitere Interaktionen notwendig sind.

3. **Zugängliche Anweisungen**: Klare Anweisungen erklären, welche Schritte der Benutzer unternehmen sollte (z.B. "Bitte bestätigen Sie, dass Sie kein Roboter sind").

4. **Progressive Verbesserung**: Bei jeder Benutzerinteraktion wird der Score neu berechnet, sodass der Benutzer durch natürliche Interaktionen den Schwellenwert überschreiten kann.

Diese Maßnahmen gewährleisten einen wirksamen Schutz gegen Bots, während gleichzeitig eine positive Benutzererfahrung für Menschen sichergestellt wird.

Human-Verification

Die Human-Verification-Komponente ist eine barrierefreie Alternative zu herkömmlichen CAPTCHAs:

1. **Zugängliche Checkbox**: Vollständig mit Tastatur bedienbar und für Screenreader optimiert
2. **ARIA-Attribute**: Umfangreiche Nutzung von ARIA-Rollen und -Attributen für bessere Zugänglichkeit
3. **Fokus-Management**: Deutliche visuelle Fokus-Indikatoren für Tastaturnutzer
4. **Semantisches Markup**: Korrekte HTML-Struktur für optimale Zugänglichkeit

Implementierung:

```
// Auszug aus ui.js
export function createOrGetHumanVerification(form) {
    // Eindeutige ID für die Checkbox
    const checkboxId = `${form.id || getRandomId()}-human-verification`;

    // Container mit korrekter ARIA-Rolle
    const wrapper = document.createElement('div');
    wrapper.setAttribute('role', 'group');
    wrapper.setAttribute('aria-labelledby', `${checkboxId}-label`);

    // Unsichtbares Label für Screenreader
    const srLabel = document.createElement('span');
    srLabel.id = `${checkboxId}-label`;
    srLabel.className = 'sr-only';
    srLabel.textContent = 'Bestätigung dass Sie kein Roboter sind';

    // Checkbox mit allen notwendigen Attributen
    checkbox = document.createElement('input');
    checkbox.type = 'checkbox';
    checkbox.id = checkboxId;
    checkbox.setAttribute('aria-required', 'true');
    checkbox.setAttribute('aria-describedby', `${checkboxId}-desc`);

    // Sichtbares Label
    const label = document.createElement('label');
    label.htmlFor = checkboxId;
    label.textContent = 'Ich bin kein Roboter';

    // ...weitere Implementierung
}
```

Formularschutz

Der Formularschutz kombiniert mehrere Techniken:

1. **Honeypot-Felder**: Versteckte Felder, die für Menschen unsichtbar sind, aber von Bots ausgefüllt werden
2. **Zeitanalyse**: Überwachung der Zeit, die zum Ausfüllen des Formulars benötigt wird (zu schnelles Ausfüllen deutet auf Bots hin)
3. **Versteckte Sicherheitsfelder**: Felder zur Speicherung des Sicherheits-Scores und der Startzeit
4. **Dynamische Validierung**: Echtzeitvalidierung der Eingaben mit Fehlerrückmeldungen

Funktionsweise:

```
// Auszug aus formProtection.js
export function protectForm(form) {
    // Formular-Konfiguration erstellen
    const formConfig = {
        requiredFields: discoverRequiredFields.call(this, form),
        feedbackElement: createOrGetFeedbackElement.call(this, form)
    };

    // Verstecktes Feld für Sicherheits-Score
    formConfig.botScoreField = createOrGetHiddenField.call(
        this, form, this.config.botScoreFieldName, '0'
    );

    // Verstecktes Feld für Startzeit
    formConfig.startTimeField = createOrGetHiddenField.call(
        this, form, this.config.startTimeFieldName, Date.now().toString()
    );

    // Honeypot-Feld hinzufügen
    formConfig.honeypotField = createOrGetHoneypotField.call(this, form);

    // Human-Verification-Checkbox hinzufügen
    formConfig.humanVerification = createOrGetHumanVerification.call(this,
form);

    // Event-Listener und weitere Schutzmaßnahmen...
}
```

E-Mail-Templates

Das Template-System ermöglicht dynamische E-Mail-Inhalte:

1. **Platzhalter-Ersetzung**: `{{variableName}}` wird durch entsprechende Werte ersetzt
2. **Bedingte Blöcke**: `{{#if bedingung}}...{{/if}}` für bedingte Inhalte
3. **Schleifen**: `{{#each array}}...{{/each}}` für wiederholte Inhalte
4. **Dynamische Felder**: Automatische Generierung von E-Mail-Inhalten basierend auf Formulardaten

Beispiel:

```
// Auszug aus templateEngine.js
export function replacePlaceholders(template, data, templateConfig) {
    let result = template;

    // Einfache Platzhalter ersetzen {{name}}
    result = result.replace(/\{\{([^#\/][^}]*)\}\}/g, (match, key) => {
        // Prüfen auf Funktionen, Daten oder Standardwerte...
        // ...
        return value || '';
    });

    // Bedingte Blöcke verarbeiten
    result = processConditionalBlocks(result, data);

    // Schleifen verarbeiten
    if (data.attachments) {
        result = processLoops(result, data);
    }

    return result;
}
```

Barrierefreiheit

Capture Barrierefrei wurde mit Fokus auf Zugänglichkeit entwickelt:

1. **Tastaturzugänglichkeit**: Alle Funktionen sind vollständig per Tastatur bedienbar
2. **Screenreader-Unterstützung**: ARIA-Attribute und semantisches HTML für optimale Screenreader-Kompatibilität
3. **Skip-Links**: Ermöglichen Tastaturnutzern, direkt zum Hauptinhalt zu springen
4. **Farben und Kontraste**: Ausreichende Kontrastverhältnisse gemäß WCAG 2.1
5. **Fehlerfeedback**: Klare, zugängliche Fehlermeldungen
6. **Fokus-Management**: Deutliche visuelle Fokusanzeigen und logische Fokusreihenfolge

Installationsanleitung

1. **Dateien kopieren**:
   Kopieren Sie den gesamten `captureBarrierefrei`-Ordner in Ihr Webprojekt.

2. **Skripte einbinden**:

```
<!-- CaptureBarriereFrei und MailSender Module -->
<script type="module" src="assets/js/captureBarriereFrei/index.js">
</script>
<script type="module" src="assets/js/mailSender/index.js"></script>

<!-- Konfiguration -->
<script src="assets/js/config.js"></script>
<script>
  document.addEventListener('DOMContentLoaded', function() {
    window.FormularKonfiguration.initialisiere();
  });
</script>
```

3. **Formular vorbereiten**:

```
<form id="kontaktFormular" method="post" class="protected" data-
template="kontakt">
  <!-- Formularfelder -->
</form>
```

4. **PHP-Backend einrichten**:
   Stellen Sie sicher, dass `send_mail.php` auf Ihrem Server korrekt konfiguriert ist.

Konfiguration

Die Hauptkonfiguration erfolgt in `config.js`:

```
// Empfänger-Einstellungen
const EMPFAENGER_EMAIL = 'ihre-email@domain.de';
const EMAIL_BETREFF = 'Neue Nachricht vom Kontaktformular';

// Formular-Einstellungen
const FORMULAR_ID = '#kontaktFormular';
const FORMULAR_ZURUECKSETZEN = true;

// Bot-Schutz-Einstellungen
const BOT_SCHUTZ_AKTIVIEREN = true;
const MIN_AUSFUELLZEIT = 3; // Sekunden

// Weitere Einstellungen für Dateiuploads, Templates etc.
```

Nutzungsbeispiele

Einfache Implementierung:

```
<form id="kontaktFormular" class="protected">
  <div class="form-group">
    <label for="name">Name</label>
    <input type="text" id="name" name="name" required>
  </div>
  <div class="form-group">
    <label for="email">E-Mail</label>
    <input type="email" id="email" name="email" required>
  </div>
  <div class="form-group">
    <label for="nachricht">Nachricht</label>
    <textarea id="nachricht" name="nachricht" required></textarea>
  </div>
  <button type="submit">Absenden</button>
</form>
```

Benutzerdefinierte Konfiguration:

```
const captureConfig = {
  formSelectors: '#meinFormular',
  minTimeToFill: 5000,  // 5 Sekunden
  thresholdScore: 20    // Höherer Schwellenwert für mehr Sicherheit
};

const mailConfig = {
  recipient: 'team@domain.de',
  subject: 'Kontaktanfrage Website',
  endpoint: 'mein-handler.php'
};

// Instanzen initialisieren
const captureInstance = new CaptureBarriereFrei(captureConfig);
const mailSender = new MailSender(mailConfig);
```

TODO und überlegung

.env
php in base64?
Dynamische Scores bilden (Jitter, interaction ...)
nichtlineare Skalen da authentischer

```
ts
Kopieren
Bearbeiten
function scoreMausbewegung(bewegungen: number): number {
  if (bewegungen === 0) return 0;
  if (bewegungen < 5) return 2;
  if (bewegungen < 15) return 6;
  return 10;
}
```

Lizenz
