# Documentation de L’API Recaptcha IUC

## Cahier des Charges

Le projet Recaptcha IUC repose sur la création d’une api de sécurité pour l’accès aux sites web. De la même manière que son homologue Google Recaptcha, il vise à empêcher tout robot ou tout comportement suspect pouvant nuire au site web. Ainsi, il est donc fondamental pour noter API d’avoir les fonctionnalités suivantes :

* Un site web ou tout utilisateur pourra se procurer un moyen d’utiliser l’API
* Le Conditionnement de l’achat pour entrer en possession des identifiants pour l’API
* L’expression explicite de l’implémentation de l’API dans le site de l’utilisateur
* La génération Automatique de plusieurs types de captcha selon l’achat du client
* Une vérification intrusive des actions de l’utilisateur dès lors que le captcha est généré

## Choix des Technologies

Parmi la multitude de technologies permettant des réaliser une API, nous nous sommes arrêté sur 2 notamment :

* Python : qui grâce à la librairie Flask permet de generer des API de haute qualité. Un autre atout de python est sa diversité de bibliothèque qui lui permettent aisément de pouvoir s’intégrer dans n’importe quel projet multiplateforme.
* JavaScript : L’arme absolue en terme de réactivité et de dynamisme, Il ne fait plus aucun doute sur l’efficacité de ce langage purement adapté pour le développement web.

Le couplage de ces 2 technologies dans notre api lui confère une grande réactivité assurée par des opérations gérées avec souplesse en back end.

## Base de données

Comme technologie de base de données, nous sommes partis sur le No SQL de GOOGLE FIREBASE. Firebase est un outil très répandu dans le monde du développement de solutions informatiques aujourd’hui. Grace à sa large gamme de produits, il a pu offrir à notre projet la satisfaction dont il avait besoin pour les opérations sur les données.

* Firestore : est un produit de Firebase permettant de créer des bases de données non relationnelle basées sur la relation « collection-document »
* Storage : permet de stocker les fichiers dans Firebase et d’y accéder très facilement via un lien http pour chaque fichier. C’est ici que sont hébergés tous les fichiers CSS, JS et images de l’API.

## Le Font-End

Pour le Design UI, nous nous sommes aidés des Template Bootstrap suivants :

* **Progressus Template** : pour le décor du site. Qui est un Template léger fait pour la présentation de software.
* **StartBoostrap shop Home Page Template** : Pour la partie présentation des produits à vendre.

## Le Back-end

Pour le back end nous avons axé les fonctionnements suivants :

* La vérification des Tokens du client :
* JavaScript :
* function testKeys(){
* console.log('testing keys ...');
* var xhr = new XMLHttpRequest();
* let url = new URL('http://127.0.0.1:5000/testKeys');
* //url.searchParams.set('q', 'test me!');
* xhr.open("GET", url + '?userKey=' + Captcha\_UIT\_Token.UserKey + '&APIkeys='
* + Captcha\_UIT\_Token.APIKey, true);
* xhr.setRequestHeader('Content-Type', 'application/json; charset=utf-8');
* xhr.onreadystatechange = () => { // Call a function when the state changes.
* if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
* if (xhr.responseText != "False") {
* type=xhr.responseText;
* if(xhr.responseText != "soft"){
* let im=captcha\_get\_images(xhr.responseText);
* }
* else if(xhr.responseText == "soft"){
* draw\_captchas(xhr.responseText);
* }
* }
* else{
* alert("Impossible de charger vos Captchas, Veuillez réessayer ou verifiez vos Clés d'accès");
* }
* }
* }
* xhr.send();
* }
* Python :
* @app.route('/testKeys', methods=['GET'])
* def testKeys():
* userKey = request.args.get('userKey')
* APIkeys = request.args.get('APIkeys')
* users = db.collection('Sales').where('user\_key','==',userKey).where('API\_key','==',APIkeys).stream()
* for user in users:
* datas=user.to\_dict()
* return datas['type']
* return "False"
* La génération des Captchas :

La génération des captchas se fait par téléchargement aléatoires des captchas enregistrés préalablement dans Firebase Storage.

* Python :
* def generate\_captcha(type\_,N):
* n=random.randint(0, N+1)
* print('position: ' ,n)
* words = db.collection('Captchas').where('type','==',type\_).where('position','==',n).stream()
* for word in words:
* datas=word.to\_dict()
* return datas['image'],datas['solution']
* def generate\_captcha\_images():
* N=800
* positions=[]
* images=[]
* results=[]
* for i in range(9):
* positions.append(random.randint(0, N+1))
* for pos in positions:
* words = db.collection('Captchas').where('type','==','images').where('position','==',pos).stream()
* for word in words:
* datas=word.to\_dict()
* images.append(datas['image'])
* results.append(datas['solution'])
* break
* return images,results
* La vérification des résultats :
* JavaScript :
* function captcha\_testAnswer(){
* var answer='';
* var loading = document.getElementById("captcha\_"+type+"\_loading")
* if(type!='images'){
* var inp = document.getElementById("captcha\_"+type+"\_input")
* answer=inp.value;
* }
* else{
* for(var j=0;j<9;j++){
* var el='0';
* for(var i=0;i<sels.length;i++){
* if(j==sels[i]){
* el='1';
* break;
* }
* }
* answer+=el+'-';
* }
* }
* console.log(answer);
* loading.style.display="inline-block";
* var xhr = new XMLHttpRequest();
* xhr.open("GET", 'http://127.0.0.1:5000/testAnswer' + '?answer=' + answer
* + '&times=' + captcha\_times\_catcher(type) + '&type=' + type, true);
* xhr.setRequestHeader('Content-Type', 'application/json; charset=utf-8');
* xhr.onreadystatechange = () => { // Call a function when the state changes.
* if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200){
* loading.style.display="none";
* if (xhr.responseText == "OK") {
* Captcha\_UIT\_Token.result="Passed";
* captcha\_iuc\_onsuccess();
* }
* else{
* Captcha\_UIT\_Token.result="Failed";
* captcha\_iuc\_onfailure();
* //draw\_captchas(type);
* alert("Echec de test Captcha, Veuillez recharger la page pour réessayer");
* }
* }
* }
* xhr.send();
* }
* Python :
* @app.route('/testAnswer', methods=['GET'])
* def testAnswer():
* answer = request.args.get('answer')
* times = request.args.get('times')
* type\_ = request.args.get('type')
* print(get\_result() +' \*\*\*\*\*\*\*\*\* '+ answer)
* if answer==get\_result()+"" and Evaluate\_times(times,type\_):
* return "OK"
* else:
* return "False"

## Implémentation

Dès le lancement de notre API sur Python (en local) ; L’utilisateur peut alors accéder à l’api suivant les réglages suivants :

<html>

<head>

<link rel="stylesheet" href="https://storage.googleapis.com/recaptcha-uic.appspot.com/Recaptcha\_js/css/bootstrap-theme.css" />

<link rel="stylesheet" href="https://storage.googleapis.com/recaptcha-uic.appspot.com/Recaptcha\_js/css/bootstrap.min.css" />

</head>

<body>

<div id="captcha\_iuc">

</div>

</body>

<script >

var captcha\_result="";

var Captcha\_UIT\_Token={

UserKey: ‘userkey’,

APIKey: ‘apiKey’

};

function captcha\_iuc\_onsuccess(){

captcha\_result="success";

... Ecrivez votre code de succes ici

}

function captcha\_iuc\_onfailure(){

captcha\_result="failure";

... Ecrivez votre code d'echec ici

}

function captcha\_iuc\_onerror(){

... Ecrivez votre code de gestion d'erreur ici

}

</script >

<script src="http://netdna.bootstrapcdn.com/bootstrap/3.0.0/js/bootstrap.min.js" > </script >

<script src="https://storage.googleapis.com/recaptcha-uic.appspot.com/Recaptcha\_js/core/captcha\_IUC\_API\_distant.js" > </script >

</html >