



1 Rue de la Chebarde, Complexe des Cézeaux, BP 125 – 63178 Aubière

Rapport de projet de première année :

**Formulaire « de style » avec HTML(5)  
et CSS(3)**

Elèves :

**Nahel CHAZOT  
Juliette NAUDIN**

Durée : 20 heures

Encadrant : David HILL

Février à Juin 2018

## Remerciements

Nous souhaitons remercier M. David HILL, notre encadrant, pour son soutien, ses conseils et son suivi.

## Table des figures

Figure II.1 : Mockup du formulaire .....	10
Figure III.1 : Diagramme de Gantt du projet .....	11
Figure III.2 : Fonctionnement d'un logiciel de gestion de version.....	12
Figure IV.1 : Fichier d'extension .html .....	14
Figure IV.2 : Syntaxe d'une balise de type paire .....	14
Figure IV.3 : Insertion d'une image.....	16
Figure IV.4 : Tableau Noms, Prénoms, Professions.....	17
Figure V. 1 : syntaxe d'une feuille de style .....	19
Figure V. 2 : code source HTML où sera appliqué style.css.....	19
Figure V. 3 : rendu sans CSS.....	19
Figure V. 4 : couleur rouge appliquée à tous les titres h1.....	20
Figure V. 5 : rendu après application de la couleur rouge pour les titres h1.....	20
Figure V. 6 : sélection d'une classe avec CSS .....	21
Figure V. 7 : code source HTML où sera appliqué style.css.....	21
Figure V. 8 : rendu avec CSS .....	21
Figure V. 9 : Sélection d'une balise contenue dans une autre .....	22
Figure V. 10 : Sélection d'un balise possédant un attribut où la valeur est ou n'est pas précisée ..	22
Figure V. 11 : Bordures en CSS .....	23
Figure V. 12 : Modification du style d'un élément lors d'un survol (CSS).....	24
Figure V.13 : Modification du style d'un élément lors d'un survol (Rendu).....	24
Figure V. 14 : « You are the CSS to my HTML » .....	25
Figure VI. 1 : balises principales d'une page HTML.....	26
Figure VI. 2 : L'entête <head> d'une page HTML.....	27
Figure VI. 3 : Effet de la balise <title>.....	27
Figure VI. 4 : Insertion d'une image « NousContacter.png ».....	28

Figure VI. 5 : Champ Nom du formulaire (Mockup).....	28
Figure VI. 6 : Champ Nom du formulaire (HTML) .....	28
Figure VI. 7 : Champ Mail du formulaire (Mockup).....	29
Figure VI. 8 : Champ Mail du formulaire (HTML) .....	29
Figure VI. 9 : Champ Téléphone du formulaire (Mockup).....	29
Figure VI. 10 : Champ Téléphone du formulaire (HTML) .....	30
Figure VI. 11 : Champ Catégorie du formulaire (Mockup).....	30
Figure VI. 12 : Champ Catégorie du formulaire (HTML).....	30
Figure VI. 13 : Liste déroulante obtenue .....	31
Figure VI. 14 : Champ Message du formulaire (Mockup).....	31
Figure VI. 15 : Champ Message du formulaire (HTML).....	31
Figure VI. 16 : Bouton d'envoi du formulaire (Mockup).....	32
Figure VI. 17 : Bouton d'envoi du formulaire (HTML) .....	32
Figure VI. 18 : Bas de page <footer>.....	33
Figure VI. 19 : Formulaire en HTML sans CSS.....	34
Figure VII. 1 : inclusion des polices.....	36
Figure VII. 2 : utilisation d'une police .....	36
Figure VII 3. : style du conteneur principal .....	37
Figure VII 4. : style des champs .....	37
Figure VII 5. : allure des champs avec la bordure d'un input de type text.....	38
Figure VII 6. : allure des champs sans la bordure du input text .....	38
Figure VII 7. : liste déroulante de basee, avant l'application du stylee.....	38
Figure VII 8. : liste après l'application du style.....	38
Figure VII 9. : alignement des champs email et téléphone.....	39
Figure VII 10. : style du bouton d'envoi.....	39
Figure VII 11. : Rendu de la page avec le CSS.....	40
Figure VIII. 1 : code des fonctions JS pour masquer et afficher un élément.....	42

Figure VIII. 2 : fonction JS de désactivation des messages de correction.....	43
Figure VIII. 3 : code HTML relatif au champ Nom du formulaire .....	43
Figure VIII. 4 : fonction JS de vérification du champ Nom du formulaire.....	44
Figure VIII. 5 : fonction JS indiquant si le nom est en train d'être édité .....	44
Figure VIII. 6 : code HTML relatif au champ Mail du formulaire.....	45
Figure VIII. 7 : fonction JS de vérification du champ Mail du formulaire.....	45
Figure VIII. 8 : fonction JS indiquant si le mail est en train d'être édité.....	46
Figure VIII. 9 : code HTML relatif au champ Téléphone du formulaire .....	46
Figure VIII. 10 : fonction JS de vérification du champ Téléphone du formulaire .....	46
Figure VIII. 11 : fonction JS indiquant si le téléphone est en train d'être édité .....	47
Figure VIII. 12 : code HTML relatif au champ Message du formulaire .....	47
Figure VIII. 13 : fonction JS de vérification du champ Message du formulaire .....	48
Figure VIII. 14 : code HTML relatif au champ Catégorie du formulaire.....	49
Figure VIII. 15 : fonction JS de vérification du champ Catégorie du formulaire.....	49
Figure VIII. 16 : code HTML relatif à l'envoi du formulaire.....	50
Figure VIII. 17 : fonction JS de vérification du formulaire avant envoi.....	50
Figure VIII. 18 : rendu en cas d'erreur sur chaque champ, avant envoi.....	51
Figure IX. 1: attributs de <form> relatifs au PHP.....	52
Figure IX. 2 : vérification de variables existantes et non nulles.....	53
Figure IX. 3 : récupération des données dans des variables par php .....	54
Figure IX. 4 : rendu si l'utilisateur ne passe pas par le formulaire (erreur) .....	55
Figure IX. 5: rendu lorsque l'utilisateur envoie des données correctes via le formulaire .....	55

## Table des matières

Remerciements.....	2
Table des figures.....	3
Table des matières.....	6
I. Introduction .....	8
I. 1) Préambule.....	8
I. 2) Motivation.....	8
II. Présentation du projet.....	9
II. 1) Objectifs.....	9
II. 2) Démarches.....	9
II. 3) Environnement.....	10
III. Méthodes de travail.....	11
III. 1) Word et le suivi de modifications.....	11
III. 2) Diagramme de Gantt .....	11
III. 3) Git .....	12
IV. Tutoriel HTML .....	14
IV. 1) Démarrage.....	14
IV. 2) Balises.....	14
IV. 3) Structure de base d'une page HTML.....	15
IV. 4) Un site web sans CSS ? Sans... Nahel ? .....	17
IV. 5) Sources du tutoriel.....	17
V. Tutoriel CSS.....	18
V. 1) Emplacement.....	18
V. 2) Appliquer un style.....	18
V. 3) Propriétés utiles aux formulaires.....	22
V. 4) Suite du travail.....	25
V. 5) Sources du tutoriel.....	25
VI. Le HTML.....	26
VI. 1) Fonction du HTML.....	26
VI. 2) Vue d'ensemble.....	26
.....	26
VI. 3) Entête .....	27
VI. 4) Le corps de la page .....	27
VI. 5) Le bas de page .....	33
VI. 6) Finalité .....	33

VII. Le CSS.....	35
VII. 1) Fonction du CSS .....	35
VII. 2) Organisation des fichiers.....	35
VII. 3) Police .....	35
VII. 4) Eléments du formulaire .....	36
VII. 5) Résultat final .....	40
VIII. Le JavaScript (JS).....	41
VIII. 1) Pourquoi avoir fait du JavaScript ?.....	41
VIII. 2) Masquer/afficher les messages de correction .....	42
VIII. 3) Vérification du nom.....	43
VIII. 4) Vérification du mail .....	45
VIII. 5) Vérification du téléphone.....	46
VIII. 6) Vérification du message .....	47
VIII. 7) Vérification de la catégorie .....	49
VIII. 8) Vérification avant envoi.....	50
VIII. 9) Rendu avant envoi.....	51
IX. Le PHP.....	52
IX. 1) Pourquoi avoir fait du PHP ? .....	52
IX. 2) Comment envoyer les données saisies par l'utilisateur ? .....	52
IX. 3) Une fois envoyées, comment traiter les données ? .....	53
IX. 4) Page de traitement des données : envoi.php .....	53
IX. 5) Rendu après envoi.....	55
ANNEXES.....	56
Annexe 1 : supports pour le tutoriel de HTML.....	57
Annexe 2 : arborescence du projet.....	59
Annexe 3 : code HTML.....	60
Annexe 4 : code CSS .....	63
Annexe 5 : code JavaScript.....	66
Annexe 6 : code PHP .....	69

## I. Introduction

### I. 1) Préambule

Dans le cadre de notre second semestre à l'Institut Supérieur d'Informatique, de Modélisation et de leurs Applications (ISIMA), nous avons travaillé sur un projet de développement. Ce projet a été d'une durée de 20 heures par personne, et a été encadré par M. David Hill.

### I. 2) Motivation

Issus de filières différentes (IUT informatique et classe préparatoire), nous avions à cœur de choisir un projet qui ne nécessitait pas de connaissances pré-requises, et où nous pourrions tous les deux apprendre. L'un de nous deux, ayant un bagage plus solide en informatique, possédait déjà des notions sur le sujet, mais ceci n'a pas été dérangeant. En effet, nous pouvons toujours acquérir de nouvelles compétences en développement web, tant la complexité et l'étendu du travail que peut nécessiter un site sont grands. Curieux de savoir comment une page web était construite, et d'en créer à notre tour, nous nous sommes rapidement positionnés sur ce projet qui était notre premier choix.

## II. Présentation du projet

### II. 1) Objectifs

Pour créer un site web, on doit donner des instructions à l'ordinateur. Il ne suffit pas simplement de taper le texte qui devra figurer dans le site (comme on le ferait dans un traitement de texte Word, par exemple), il faut aussi indiquer où placer ce texte, insérer des images, ou bien encore créer des formulaires.

Pour ce faire, il faut utiliser au minimum deux langages de programmation qui se complètent puisqu'ils ont des rôles différents :

- **HTML (HyperText Markup Language)** : son rôle est de gérer et organiser le contenu : paragraphes, titre, texte, des liens, des images... Il sert à dire au navigateur comment structurer les pages web que l'on visite. Le HTML 5 en est la dernière version, elle rajoute des fonctionnalités telles que la possibilité d'inclure facilement des vidéos.
- **CSS : (Cascading Style Sheets, aussi appelées Feuilles de style)** : son rôle est de gérer l'apparence de la page web (agencement, positionnement, décoration, couleurs, image de fond, taille du texte...). Le CSS 3 en est la dernière version, elle apporte des fonctionnalités telles que l'ajout de bordures arrondies et les ombres.

Notre projet consistait à **créer une page web avec un formulaire en utilisant, donc, HTML 5 et CSS 3**. Ce formulaire devait être fait de manière ergonomique (« de style »).

### II. 2) Démarches

Dans un premier temps, nous devions nous initier à la pratique du HTML et du CSS afin de découvrir ou redécouvrir ce qu'était le HTML et le rôle du CSS. Pour cela, nous avons fait un tutoriel chacun : l'un expliquait à l'autre ce qu'était le HTML et l'autre expliquait à l'un comment utiliser le CSS. Chacun devait donc se confronter à des documentations en français ou en anglais pour faire un travail de synthèse et expliquer au mieux ce dont il était nécessaire de maîtriser pour la suite du projet. Puis, nous avons effectué ensemble la mise en forme toute simple d'une page web, qui est devenue par la suite notre page de formulaire.

Une fois ces bases posées, nous sommes rentrés dans le vif du sujet en insérant notre formulaire au sein de notre page web. Nous avons donc utilisé le HTML propre aux formulaires et nous l'avons personnalisé grâce au CSS : design des champs, du bouton... Nous avons réalisé un mockup (= une maquette) du site afin d'avoir un modèle à suivre.

## Nous Contacter

NOM  
Entrez votre nom

EMAIL  
Entrez votre email

TELEPHONE  
Entrez votre numéro de téléphone

CATEGORIE SOCIOPROFESSIONNELLE  
Sélectionnez le groupe auquel vous appartenez ▾

MESSAGE  
Tapez votre message ici...

Envoyer →

Figure II.1 : Mockup du formulaire

Ce travail accompli, nous avons profité du temps qu'il nous restait pour approfondir : comment à la fois vérifier si les données recueillies par le formulaire sont celles attendues et rendre la page interactive ? Pour répondre à cette question, nous avons inséré du langage javascript dans notre page. Notre formulaire est alors devenu dynamique.

Enfin, nous nous sommes intéressés au langage PHP afin de pouvoir récupérer et utiliser les données envoyées via le formulaire.

### II. 3) Environnement

Les pages web du projet ont été testées sous les navigateurs Mozilla Firefox et Google Chrome. Elles ont été éditées sous Sublime Text, et nous avons utilisé un serveur local Wamp pour utiliser le PHP.

### III. Méthodes de travail

#### III. 1) Word et le suivi de modifications

Nous avons rédigé nos tutoriels sous Word, et nous avons effectué une relecture croisée de nos tutoriels en utilisant le suivi de modification que propose Word. Le suivi de modification permet de connaître qui a apporté des modifications au document, de choisir les modifications à accepter ou refuser, et aussi d'ajouter des commentaires.

#### III. 2) Diagramme de Gantt

Le diagramme de Gantt est un outil d'ordonnancement de projet. Il nous a permis de visualiser la liste des tâches à effectuer et d'évaluer le temps à accorder à chacune.

A la fin de notre projet, nous avons obtenu le diagramme de Gantt suivant :

#### Diagramme de Gantt

Intitulé du projet :

Formulaire HTML/CSS

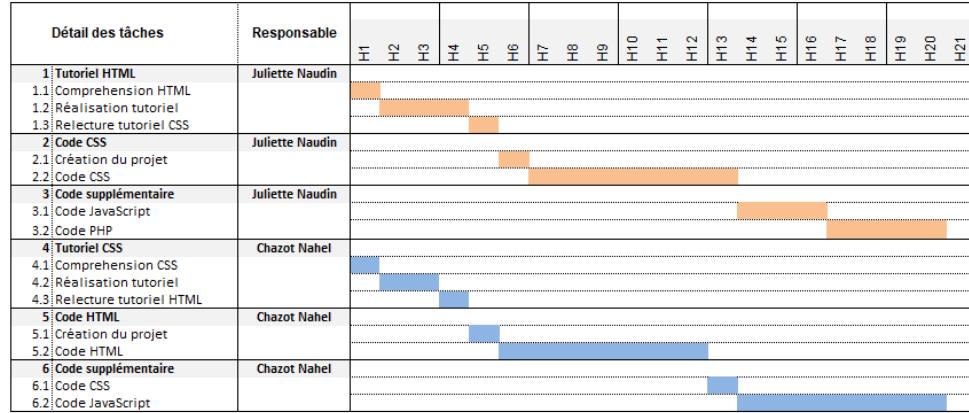


Figure III.1 : Diagramme de Gantt du projet

Les tâches n°1, n°2, n°4 et n°5 représentent l'objectif de base du projet. Après les avoir terminées, il nous restait une douzaine d'heures au total. Nous avons alors décidé, comme expliqué précédemment, de rajouter du code supplémentaire, à savoir du JavaScript et du PHP (tâches n°3 et n°6).

### III. 3) Git

#### III. 3) a. Fonctionnement

Git est un logiciel de gestion de version. Le principe est simple, on crée un dépôt distant sur un serveur sur lequel sera stocké notre projet. Dès que l'un des contributeurs veut ajouter ou modifier un élément du projet, il récupère le projet sur sa machine et peut travailler dessus en local.

Quand il a fini, il valide ses modifications en indiquant les fichiers ajoutés/modifiés accompagnés d'un commentaire expliquant ce qui a été fait.

Le contributeur renvoie ensuite le projet au serveur qui s'occupe de faire une mise-à-jour.

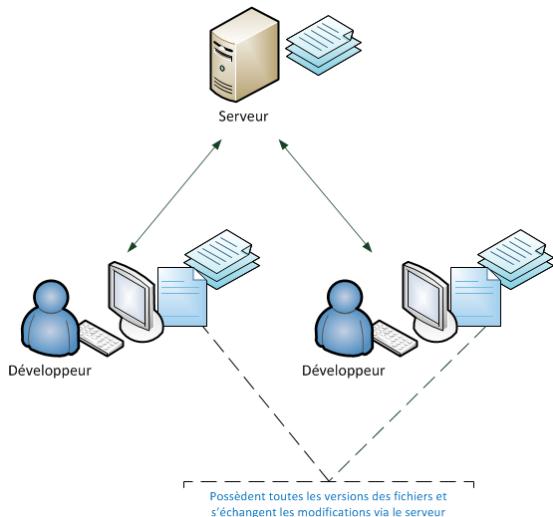


Figure III.2 : Fonctionnement d'un logiciel de gestion de version

#### III. 3) b. Quelques avantages

- Git enregistre la liste de chaque modification, chacune appelée « commit ». Nous pouvons alors revenir à toute ancienne version du projet. Par exemple si un bug apparaît, il est simple de trouver d'où il provient en s'aidant des précédentes modifications.
- Git enregistre l'émetteur de chaque commit. Si un contributeur ne comprend pas une ligne de code, il peut savoir qui l'a ajouté et lui demander de l'aide.
- Un message est attaché à chaque commit. En regardant la liste des commits, nous pouvons suivre le déroulement du projet.

Dans le cadre de notre projet, nous avons utilisé GitHub comme service d'hébergement.

L'adresse du dépôt est la suivante : [https://github.com/nalchaz/formulaire\\_ZZ1](https://github.com/nalchaz/formulaire_ZZ1).

## IV. Tutoriel HTML

réalisé par Juliette Naudin pour Nahel Chazot

Ce tutoriel fait référence à l'annexe 1.

### IV. 1) Démarrage

On utilise un éditeur de texte (ici Sublime Text) pour créer un fichier ayant l'extension.html (ici : accueil.html). Ce sera notre première page web. Ce fichier peut être ouvert dans le navigateur web.



Figure IV.1 : Fichier d'extension .html

### IV. 2) Balises

À l'intérieur du fichier, nous écrirons le contenu de notre page, accompagné de balises HTML. Elles servent à décrire correctement la structure du document. Le navigateur n'affiche pas les balises telles quelles : lorsqu'un utilisateur visite une page web, son navigateur analyse le document et l'interprète afin d'afficher la page web correctement.

#### IV. 2) a. Type

Les balises peuvent être de deux types :

- <balise> </balise> : celles qui fonctionnent par paire (s'ouvrent et se ferment)
- <balise /> : les orphelines

#### IV. 2) b. Syntaxe



Figure IV.2 : Syntaxe d'une balise de type paire

*Les balises doivent être fermées dans le sens inverse de leur ouverture :*



<html><body></html></body>	✓
<html><body></body></html>	✗

#### IV. 2) c. Liste des balises HTML ?

Comme il existe de nombreuses balises HTML, les liens ci-dessous ne répertorient que les balises les plus utilisées en pratique :

- <https://openclassrooms.com/courses/apprenez-a-creer-votre-site-web-avec-html5-et-css3/memento-des-balises-html>
- <https://jaetheme.com/balises-html5/>

Notons cependant que pour écrire un commentaire il faut utiliser :

<!-- Ceci est un commentaire -->

#### IV. 3) Structure de base d'une page HTML

##### IV. 3) a. La balise <html>

Englobe tout le contenu de la page. Indique au navigateur que le contenu est le code HTML.

##### IV. 3) b. L'en-tête <head>

Donne des informations générales importantes sur la page comme son titre, l'encodage (pour la gestion des caractères spéciaux), etc.

- **L'encodage (« charset »)**

<meta charset="utf-8" /> : indique l'encodage utilisé dans accueil.html. Détermine comment les caractères spéciaux vont s'afficher (accents, idéogrammes chinois et japonais, caractères arabes, etc.). L'encodage UTF-8 précisé ici permet d'afficher sans aucun problème pratiquement tous les symboles de toutes les langues existantes.

- **Titre de la page**

<title> Prems </title> : donne un titre en haut de la page, à l'onglet du navigateur.  
Nb : c'est le titre qui apparaît aussi dans les résultats des moteurs de recherche.

J'ai appelé notre première page « Prems » ... c'est la première !

##### IV. 3) c. Corps de la page <body>

C'est cette balise qui contient le contenu de la page. C'est donc cette balise qui contient la majeure partie du code.

- **Image** <img />

`src` : (attribut obligatoire) chemin (relatif ou absolu) de l'image à insérer  
`alt` : (attribut obligatoire) texte alternatif à afficher si l'image ne peut être chargée  
`title` : rajoute une infobulle



Figure IV.3 : Insertion d'une image

En haut du body, j'ai mis l'image `projetzz.png` ci-dessus, ayant pour adresse relative à `accueil.html` : `images/projetzz.png`. Au survol de l'image, on a l'infobulle « un super projet ! » qui apparaît (comme si on ne le savait pas...). Si l'image n'est pas chargée, le texte « Logo\_Projet » est alors affiché.

- **Un peu de mise en page !**

<br/> : retour à la ligne.

<p> : créé un paragraphe. Les paragraphes sont automatiquement séparés par des sauts de lignes.

- **Lien** <a href="#" ></a>

`href` : chemin (relatif ou absolu) pouvant être externe (ie vers un autre site, on parle alors d'URL et non de chemin) du lien.

J'ai placé un lien sur "ici" dirigeant vers un autre fichier html : `formulaire.html`, où sera notre formulaire (étape 2 du travail !,... restons encore 2 minutes à l'étape 1).

Nb 1 : D'autres attributs sont bien sûr possibles comme `title` pour rajouter une infobulle, ou pour avoir un lien qui ouvre une nouvelle fenêtre du navigateur, télécharger un fichier, etc.  
Nb 2 : Le hashtag # permet de ne pas bouger de la page sur laquelle on est, même en cliquant sur le lien (➔ lien fictif).

- **Le pied de page** <footer>

À l'inverse de l'en-tête, le pied de page se trouve en général tout en bas du document.

- **Tableau <table>**

<table> </table> : indique le début et la fin d'un tableau

<tr> </tr> : indique le début et la fin d'une ligne du tableau

<td> </td> : indique le début et la fin du contenu d'une cellule

CHAZOT	Nahel	Ingénieur
MOROUX	Christine	Dentiste
ELMALEH	Gad	Humoriste

Figure IV.4 : Tableau Noms, Prénoms, Professions

J'ai donc fait un pied de page avec nos coordonnées : une ligne chacun avec nom, prénom et mail ISIMA.

#### IV. 4) Un site web sans CSS ? Sans... Nahel ?

On peut très bien créer un site web uniquement en HTML, mais celui-ci ne sera pas très beau : l'information apparaîtra « brute ». On peut tout de même styliser le contenu brut par des balises HTML : comme dans cette première page par exemple, j'ai utilisé <i> pour mettre en italique, <u> pour mettre en gras. Néanmoins l'aspect de cette page laisse toujours à désirer... C'est pour cela que le langage CSS vient toujours le compléter : à toi Nahel !

#### IV. 5) Sources du tutoriel

Les balises :

[https://developer.mozilla.org/fr/Apprendre/HTML/Balises\\_HTML](https://developer.mozilla.org/fr/Apprendre/HTML/Balises_HTML)

<http://www.commentcamarche.com/contenus/488-balise-html>

Structure :

[https://developer.mozilla.org/fr/Apprendre/HTML/Comment/Cr%C3%A9er\\_un\\_document\\_HTML\\_simple](https://developer.mozilla.org/fr/Apprendre/HTML/Comment/Cr%C3%A9er_un_document_HTML_simple)

Tableaux :

<https://openclassrooms.com/courses/apprenez-a-creer-votre-site-web-avec-html5-et-css3/les-tableaux-1>

Complément de structure (ex : pied de page) :

<https://openclassrooms.com/courses/apprenez-a-creer-votre-site-web-avec-html5-et-css3/structurer-sa-page>

## V. Tutoriel CSS

réalisé par Nahel Chazot pour Juliette Naudin

Ce tutoriel fait référence à l'annexe 4.

### V. 1) Emplacement

Le CSS peut s'écrire à différents endroits :

- Directement dans le fichier HTML au sein de balise (attribut « style »)
- Dans l'en-tête `<head>` du fichier HTML
- Dans un fichier `.css`

La solution la plus propre est d'utiliser un fichier `.css` qui permet de séparer le code CSS du code HTML pour plus de lisibilité.

Pour ajouter une feuille de style à un fichier HTML il faut lui indiquer où la trouver grâce à la balise `<link>`. Il faut indiquer l'option « `rel` » avec la valeur « `stylesheet` » ainsi que l'option « `href` » avec la localisation du fichier de style.

Par exemple :

```
<link rel="stylesheet" href="style.css" />
```

### V. 2) Appliquer un style

#### V. 2) a. Syntaxe

Pour donner un style on a besoin de trois informations :

- Le **nom** de la balise ou du type de balise
- Les **propriétés** CSS à appliquer
- Les **valeurs** attribuées à ces propriétés

Ainsi, une feuille de CSS ressemble à ça :

```

1 nomBalise1{
2     propriete1 : valeur1 ;
3     propriete2 : valeur2 ;
4 }
5
6 nomBalise2{
7     propriete1 : valeur1 ;
8     propriete2 : valeur2 ;
9     propriete3 : valeur3 ;
10}

```

Figure V. 1 : syntaxe d'une feuille de style

*Note : Pour ajouter un commentaire dans un fichiers CSS il faut l'inserer entre /\* et \*/*

## V. 2) b. Sélecteurs de balises

La feuille de style « style.css » sera appliquée sur le code HTML suivant :

```

1 <html>
2
3     <h1> Projet ZZ1 </h1>
4     <link rel="stylesheet" href="style.css" />
5     <body>
6
7         Cette page d'accueil est la toute première page web que nous
8         créons.
9
10    </body>
11 </html>

```

Figure V. 2 : code source HTML où sera appliquée style.css

Qui rend comme ceci :



Figure V. 3 : rendu sans CSS

Pour appliquer un style à un type de balise il faut indiquer le nom du type. Par exemple pour appliquer une couleur à tous les grands titres on indique le nom `h1` :

```
1  h1{  
2      color : red ;  
3  }
```

Figure V. 4 : couleur rouge appliquée à tous les titres `h1`

On obtient alors la page suivante :



Figure V. 5 : rendu après application de la couleur rouge pour les titres `h1`

Cependant ceci implique que toute les balises `h1` soit écrite en rouge. Comment éviter cela et être plus sélectif ?

## V. 2) c. Class et id

Dans les balises HTML, on peut utiliser deux attributs afin de différencier les balises de même type :

- L'attribut `class`
- L'attribut `id`

Ces deux attributs sont quasiment identiques, seulement l'attribut `class` peut être attribué à **plusieurs** balises tandis que l'attribut `id` ne peut être utilisé qu'**une seule fois** dans le code.

Pour sélectionner une classe dans le code CSS, il faut indiquer le nom de la classe en commençant par un point comme ceci :

```
1 .texteRouge{  
2 |   color : red ;  
3 }
```

Figure V. 6 : sélection d'une classe avec CSS

Ainsi pour le code HTML suivant :

```
1 <html>  
2  
3   <h1> Projet ZZ1 </h1>  
4   <link rel="stylesheet" href="style.css" />  
5   <body>  
6  
7     Cette page d'accueil est la toute première page web que nous  
8     créons.  
9  
10    <p class="texteRouge"> Texte à écrire en rouge <p>  
11    <p> Texte à laisser en noir <p>  
12  </body>  
13 </html>
```

Figure V. 7 : code source HTML où sera appliqué style.css

On obtient la page web :



Figure V. 8 : rendu avec CSS

Pour sélectionner un `id` il faut indiquer le nom de l'`id` en commençant par un « # ». Le fonctionnement est identique.

## V. 2) d. Sélecteurs spéciaux

Il existe d'autres façons de sélectionner des balises, en voici quelques-unes importantes :

- \* : le sélecteur universel, il sélectionne toutes les balises sans exception.
- A B : sélectionne toutes les balises B contenu dans une balise A.

Exemple :

```
1  h1 a{
```

Sélectionne toutes les balises a  
contenu dans une balise h1.

Figure V. 9 : Sélection d'une balise contenue dans une autre

- A[attribut] ou A[attribut= "Valeur" ] : sélectionne toutes les balises A possédant l'attribut entre crochet ou plus précisément l'attribut et sa valeur.

Exemple :

```
1  a[title]{
```

La première ligne selectionne toutes les balises a possédant un titre.  
La cinquième ligne selectionne toutes les balises a possédant le titre « titre1 ».

**Commenté [JN1]:** Les balises « a » dont tu parles là font référence à des liens ?

**Commenté [JN2R1]:** Non c'est une balise générique « A » c'd n'importe quel balise/classe/id

Figure V. 10 : Sélection d'un balise possédant un attribut où la valeur est ou n'est pas précisée

Beaucoup d'autres selecteurs existent, ils sont tous listés à la source, le [site du W3C](#).

## V. 3) Propriétés utiles aux formulaires

L'objectif étant de créer un formulaire, intéressons-nous aux propriétés intéressantes afin de le mettre en forme.

### V. 3) a. Le texte

Il y a énormément de possibilités concernant la mise en forme du texte. Voici simplement les plus importants :

- La police : font-family : nomPolice;
- La taille de la police : font-size : 10px;

- Le style de la police : `font-style : italic; /* Police en italique */  
font-weight : bold; /* Police en gras */`
- L'alignement : `text-align : left/center/right;`

### V. 3) b. Couleur et fond

- Couleur du texte : `color : blue/#FFFFFF ;`
- Fond : `Background-color : red/#EEEEEE;`

### V. 3) c. Bordures

Pour les bordures il existe une super-propriété dans laquelle on peut indiquer trois valeurs :

- La largeur (en pixel px)
- La couleur
- Le type de bordure :
  - o `solid` : un trait simple
  - o `dotted` : pointillés
  - o `dashed` : tirets
  - o `double` : bordure double
  - o `groove` : en relief
  - o `ridge` : autre effet relief
  - o `inset` : effet 3D global enfoncé
  - o `outset` : effet 3D global surélevé

Elle s'utilise de la manière suivante :

```

1  nomBalise
2  {
3    border: taille couleur type;
4  }
5
6
7  h1
8  {
9    border: 3px blue dashed;
10 }
```

Figure V. 11 : Bordures en CSS

Il est aussi possible d'arrondir les angles des bordures (heureusement) grâce à la propriété `border-radius`.

Il suffit d'indiquer la taille (« l'importance ») de cet arrondi en pixel :  
`border-radius : 8px ;`

**Commenté [JN3]:** Les bordures concernent quel genre d'objet HTML ? Les tableaux ? Le texte simple n'est pas concerné ?

**Commenté [JN4R3]:** Non une bordure peut être ajoutée à n'importe quelle balise

### V. 3) d. Les apparences dynamiques

Il est possible de modifier l'apparence d'un élément lors de certains évènements, après que la page ait été chargée.

L'évènement le plus connu étant le « survol » c'est-à-dire quand l'utilisateur passe la souris au-dessus de l'élément.

Pour modifier l'apparence d'un élément lors d'un survol, il faut rajouter l'information :`hover` après le nom de la balise puis indiquer les propriétés à changer.

Exemple :

```
1  h1
2  {
3      color: blue;
4  }
5
6
7  h1:hover
8  {
9      background-color: grey;
10 }
```

Figure V. 12 : Modification du style d'un élément lors d'un survol (CSS)



Figure V.13 : Modification du style d'un élément lors d'un survol (Rendu)

Il existe d'autres évènements, la liste de toutes ces pseudo-classes est disponible sur [ce lien](#).

#### V. 4) Suite du travail

Pour la suite, je prépare une page formulaire en HTML et Juliette l'embellira grâce à sa maîtrise du CSS.



Figure V. 14 : « You are the CSS to my HTML »

#### V. 5) Sources du tutoriel

Documentation :

<https://developer.mozilla.org/fr/docs/Web/CSS>

Utilisation du CSS :

<https://openclassrooms.com/courses/apprenez-a-creer-votre-site-web-avec-html5-et-css3/creation-d-apparences-dynamiques>

Sélecteurs d'éléments :

<https://www.w3.org/Style/css3-selectors-updates/WD-css3-selectors-20010126.fr.html#selectors>

Pseudo-classes :

<https://developer.mozilla.org/fr/docs/Web/CSS/Pseudo-classes>

## VI. Le HTML

Cette partie fait référence à l'annexe 3.

### VI. 1) Fonction du HTML

Le HTML est la base de notre formulaire. C'est ce qui crée la page, l'entête, le bas de page et le corps de la page.

D'après notre mockup, nous avons décidé d'organiser notre page de la manière suivante :

- L'entête : possède des informations concernant la page web.
- Le corps : possède 5 champs :
  1. Nom
  2. Mail
  3. Téléphone
  4. Catégorie
  5. Message
- Le bas de page : Nos noms, prénoms ainsi que nos adresses mail.

### VI. 2) Vue d'ensemble

Une page HTML est lue de façon linéaire, à savoir les premières lignes seront les éléments placés en premier. Les éléments doivent être écrits dans l'ordre dans lequel nous souhaitons les voir s'afficher.

La page HTML est donc divisée en trois parties successives, le « head » suivi du « body » puis du « footer ».

```
1 <html>
2
3   <head> ***
9   </head>
10
11  <body> ***
87  </body>
88
89  <footer> ***
103 </footer>
104
105 </html>
```

The diagram illustrates the structure of an HTML document. It shows the code with line numbers on the left. Three curly braces on the right side group the code into sections: a red brace groups the first three lines (the opening `<html>` tag and the `<head>` tag with its content), labeled 'Entête'; a green brace groups the next two lines (the closing `</head>` tag and the opening `<body>` tag with its content), labeled 'Corps'; and a blue brace groups the last three lines (the closing `</body>` tag and the opening and closing `<footer>` tag with its content), labeled 'Bas de page'.

Figure VI. 1 : balises principales d'une page HTML

### VI. 3) Entête

L'entête possède 3 éléments :

- L'information concernant l'encodage des caractères
- Les liens vers les feuilles de style
- Le titre affiché sur l'onglet quand la page web est ouverte

```
<head>
  <meta charset="utf-8" />
  <link rel="stylesheet" type="text/css" href="../css/main.css">
  <link rel="stylesheet" type="text/css" href="../css/police.css">
  <title> Contact </title>
</head>
```

Figure VI. 2 : L'entête `<head>` d'une page HTML

L'encodage des caractères est donné dans la balise « `meta` », on utilise l'encodage UTF-8 (Universal Character Set Transformation Format - 8 bits) qui permet d'encoder les caractères spéciaux tels que les accents.

Le lien vers chacune de nos pages de style est faite via la balise « `link` » à laquelle on donne le type de page (ici une page css) ainsi que le chemin vers la page donnée par l'option « `href` ».

La dernière balise, « `title` », permet d'indiquer ce qui apparaîtra dans l'onglet quand la page web est ouverte, ici on obtient ceci :

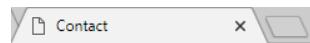


Figure VI. 3 : Effet de la balise `<title>`

### VI. 4) Le corps de la page

Le corps possède la quasi-totalité de ce qui apparaît dans la page. Chaque partie est encapsulée dans une division (balise « `div` ») pour avoir une bonne visibilité de chaque élément et pour pouvoir les manipuler par la suite dans le CSS. De plus, la plupart des balises possèdent une classe ou/et un identifiant. Ils seront eux aussi utiles pour le CSS, mais sont à ignorer pour le moment.

#### VI. 4) a. Titre

Nous Contacter

Dans notre maquette, le titre apparaît centré en haut de la page avec une police basique.

Nous avons choisi d'utiliser une image dans notre HTML.

```
<center></center>
```

Figure VI. 4 : Insertion d'une image « NousContacter.png »

La balise « center » permet de centrer tout ce qui se trouve à l'intérieur. Nous avons donc placé la balise « img » contenant l'image dedans. L'option « alt » de la balise image permet d'afficher du texte de remplacement si l'image n'arrive pas à s'afficher. Ici, si l'image ne charge pas, « Nous contacter » s'affichera à la place.

#### VI. 4) b. Nom



NOM  
Entrez votre nom

Figure VI. 5 : Champ Nom du formulaire (Mockup)

Le champ du nom se situe juste en dessous du titre et possède un cadre, un label ainsi qu'un champ dans lequel écrire.

```
<!-- Nom -->
<div class="entry" id="champ_nom">
  <span id="nomLabel" class="labelChamp">NOM</span>
  <input name="Nom" id="nomEntry" type="text" placeholder="Entrez votre nom" size="30" onblur="checkNom()" onkeydown="editionNom()">
  <div class="hiddenMessage" id="messageNom">
    Un nom doit faire plus de 2 caractères et ne contient que des lettres
  </div>
</div>
```

Figure VI. 6 : Champ Nom du formulaire (HTML)

Le champ est enveloppé dans une division, qui nous permettra de rajouter la bordure par la suite.

La division d'identifiant « messageNom » est à ignorer pour le moment, elle interviendra dans le JavaScript. Ici on s'intéresse à la balise « span » ainsi qu'à la balise « input ».

La balise « span » est un conteneur générique pour les contenus phrasés. Nous avons choisi de l'utiliser pour le label, en écrivant dedans « NOM ».

La balise « input » est un champ d'interaction avec l'utilisateur. Ici, on lui donne le type « text » indiquant qu'il s'agira d'une zone dans laquelle l'utilisateur pourra écrire du texte.

Les autres options sont :

- « placeholder », c'est le texte de substitution qui s'affichera dans le champ tant qu'il sera vide.
- « size », c'est le nombre de caractère maximum du champ.
- « onblur », « onkeydown » qui sont liés au JavaScript.

#### VI. 4) c. Mail

Figure VI. 7 : Champ Mail du formulaire (Mockup)

Le champ de l'adresse email est placé à la même hauteur que le champ téléphone dans notre mockup, et prend la moitié de l'espace.

```
<!-- Email -->
<div class="entry" id="champ_email">
  <span id="emailLabel" class="labelChamp">E-MAIL</span>
  <br/>
  <input name="Email" id="emailEntry" type="text" placeholder="Entrez votre e-mail" size="30" onblur="checkMail()" onkeydown="editionMail()">
  <div class="hiddenMessage" id="messageMail">
    Entrez un e-mail valide
  </div>
</div>
```

Figure VI. 8 : Champ Mail du formulaire (HTML)

Comme pour le nom, la division d'identifiant « messageMail » est à ignorer. La balise « span » est identique, on change simplement le texte. La balise « input » possède les mêmes options que pour le champ nom.

D'après notre mockup, la balise doit être à la même hauteur que le téléphone. Pourtant, remarquons qu'il n'y a aucun changement de fait par rapport au nom. Cela est dû au fait que nous n'avons pas géré ça par le HTML, mais en utilisant le CSS.

#### VI. 4) d. Téléphone

Figure VI. 9 : Champ Téléphone du formulaire (Mockup)

Le champ est en tout point identique au champ email.

```
<!-- Phone -->
<div class="entry" id="champ_phone">
  <span id="phoneLabel" class="labelChamp">TELEPHONE</span>
  <br/>
  <input name="Telephone" id="phoneEntry" type="text" placeholder="Entrez votre numéro de téléphone" size="10" onblur="checkTel()" onkeydown="editionTel()">
  <div class="hiddenMessage" id="messageTel">
    Entrer un numéro de téléphone valide
  </div>
</div>
```

Figure VI. 10 : Champ Téléphone du formulaire (HTML)

Une seule différence notable : l'option « `size` » a ici la valeur 10 pour correspondre à la taille d'un numéro de téléphone. (On admettra qu'un numéro de téléphone ne fait que 10 caractères).

#### VI. 4) e. Catégorie

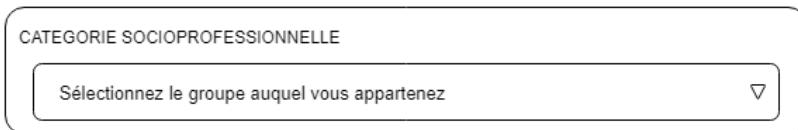


Figure VI. 11 : Champ Catégorie du formulaire (Mockup)

Le champ catégorie est différent des autres : il s'agit d'une liste déroulante dans laquelle l'utilisateur choisit sa catégorie.

```
<!-- Needed services -->
<div class="entry">
  <span id="catLabel" class="labelChamp">CATEGORIE SOCIOPROFESSIONNELLE</span>
  <br/>
  <select name="Categorie" id="catEntry" onchange="checkCat()">
    <option value="0">Sélectionnez le groupe auquel vous appartenez</option>
    <option value="1">1. Agriculteurs exploitants</option>
    <option value="2">2. Artisans, commerçants, chefs d'entreprises</option>
    <option value="3">3. Cadres et professions intellectuelles supérieures</option>
    <option value="4">4. Professions intermédiaires</option>
    <option value="5">5. Employés</option>
    <option value="6">6. Ouvriers</option>
    <option value="7">7. Retraitées</option>
    <option value="8">8. Ouvrières</option>
    <option value="9">9. Sans activité</option>
  </select>
  <div class="hiddenMessage" id="messageCat">
    Vous devez sélectionner votre catégorie socioprofessionnelle
  </div>
</div>
```

Figure VI. 12 : Champ Catégorie du formulaire (HTML)

La balise « span » ne change pas des autres champs. Nous n'utilisons plus de balise « input », elle est remplacée par la balise « select ». La balise « select » permet de créer une liste déroulante. Les éléments à afficher sont dans les balises « option ».

On obtient avec ce code la liste suivante :

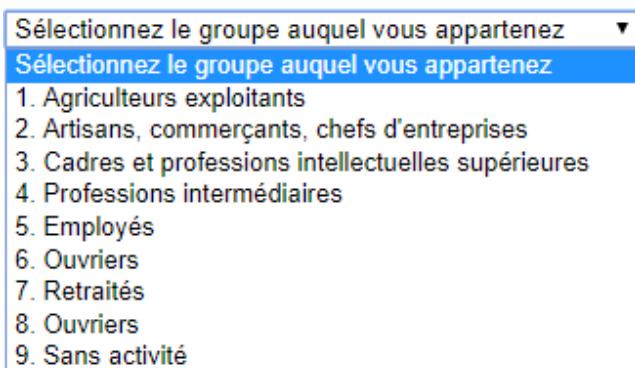


Figure VI. 13 : Liste déroulante obtenue

#### VI. 4) f. Message



Figure VI. 14 : Champ Message du formulaire (Mockup)

Le message est une grande zone de texte située en bas du formulaire.

```
<!-- Message -->


<span id="messageLabel" class="labelChamp">MESSAGE</span>
  <br/>
  <textarea name="Message" id="messageEntry" cols="100" rows="2" placeholder="Tapez votre message ici..." onfocusout="checkMessage()"></textarea>
  <div class="hiddenMessage" id="messageMessage">
    Votre message doit contenir entre 20 et 1200 caractères
  </div>

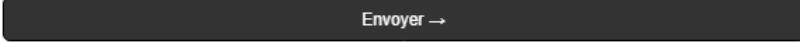

```

Figure VI. 15 : Champ Message du formulaire (HTML)

Comme pour tous les champs, on retrouve la balise « span » contenant le label.

Ici on utilise une balise « `textarea` ». Il s'agit d'une balise permettant d'éditer du texte sur plusieurs lignes. On retrouve les options utilisées dans les `input text` accompagnés de deux autres options : « `rows` » et « `cols` ». Ces options servent à spécifier la largeur et la hauteur du `textarea`.

#### VI. 4) g. Bouton



Envoyer →

Figure VI. 16 : Bouton d'envoi du formulaire (Mockup)

Le bouton d'envoi se situe tout en bas du formulaire.

```
<!-- Button -->
<div class="submitArea2">
|   <input name="submit" type="submit" value="Envoyer →" id="submitButton" class="submitArea">
</div>
```

Figure VI. 17 : Bouton d'envoi du formulaire (HTML)

Un bouton s'implémente de deux façons :

- La balise « `button` »
- La balise « `input` » avec un type « `submit` »

Nous avons choisi d'utiliser l'`input` qui nous sera utile par la suite, pour le PHP.  
L'option « `value` » indique le texte écrit dans le bouton.

## VI. 5) Le bas de page

Le bas de page indique nos noms, prénoms ainsi que nos adresses email. Pour les afficher, nous avons choisi d'utiliser un tableau HTML.

```
<footer>
    <br/>
    <table>
        <tr><!-- Début 1ere ligne -->
        <td> CHAZOT </td>
        <td> Nahel </td>
        <td> nahel.chazot@etu.isima.fr </td>
        </tr><!-- Fin 1ere ligne -->
        <tr><!-- Début 2eme ligne -->
        <td> NAUDIN </td>
        <td> Juliette </td>
        <td> juliette.naudin@etu.uca.fr </td>
        </tr><!-- Fin 2eme ligne -->
    </table>
</footer>
```

Figure VI. 18 : Bas de page <footer>

La balise « `table` » indique le début et la fin du tableau. Les lignes sont indiquées par les balises « `tr` » et les colonnes par les balises « `td` ».

## VI. 6) Finalité

On obtient la page web suivante :

The screenshot shows a contact form with the following fields:

- NOM**: Input field labeled "Entrez votre nom".
- E-MAIL**: Input field labeled "Entrez votre e-mail".
- TELEPHONE**: Input field labeled "Entrez votre num".
- CATEGORIE SOCIOPROFESSIONNELLE**: A dropdown menu labeled "Sélectionnez le groupe auquel vous appartenez".
- MESSAGE**: A large input area labeled "Tapez votre message ici...".
- Envoyer →**: A button labeled "Envoyer →".

At the bottom of the form, there is a footer with two names and their email addresses:

CHAZOT Nahel nahel.chazot@etu.isima.fr  
NAUDIN Juliette juliette.naudin@etu.uca.fr

Figure VI. 19 : Formulaire en HTML sans CSS

Il y a encore du travail !

## VII. Le CSS

Cette partie fait référence à l'annexe 4.

### VII. 1) Fonction du CSS

Toute l'utilité du CSS réside dans la mise en page du formulaire. Après avoir obtenu tous les champs « brute » grâce au HTML, nous nous concentrerons sur l'affichage de ceux-ci, le but étant d'obtenir un site ergonomique.

### VII. 2) Organisation des fichiers

Nous avons créé deux feuilles de style, « main.css » et « police.css ». Le rôle du « main » étant de gérer l'apparence global de la page tandis que le fichier « police » gère ce qui est en rapport avec la police.

### VII. 3) Police

Nous avons choisi d'utiliser une famille de police en particulier, les « Montserrat ». Pourquoi le choix de cette police ? Nous ne voulions pas utiliser une police générique disponible sur n'importe quelle machine, mais une police moins répandue afin de comprendre comment l'ajouter au site web et en faire profiter tous les utilisateurs.

Nous avons utilisé trois polices de la famille des Montserrat :

- Montserrat-Medium
- Montserrat-SemiBold
- Montserrat-Black

#### VII. 3) a. Ajout au projet

Pour pouvoir les utiliser, nous devions les télécharger puis les ajouter au projet. Elles sont placées dans un répertoire nommé « fonts » et sont au format « ttf » (TrueType). Nous avons tout de même rajouté deux autres formats de fichier pour chaque police (« woff » et « woff2 ») qui seront utiles si le navigateur de l'utilisateur ne sait pas lire le format « ttf ».

#### VII. 3) b. Inclusion et utilisation

Après avoir ajouté les polices, il faut les inclure dans le projet afin de pouvoir les utiliser. Nous l'avons fait dans le fichier nommé « police.css ».

```

1 @font-face{
2   font-family: "Montserrat-Black";
3   src: url("../fonts/montserrat/Montserrat-Black.woff2") format('woff2'),
4       url("../fonts/montserrat/Montserrat-Black.woff") format('woff');
5 }
6
7 @font-face{
8   font-family: "Montserrat-SemiBold";
9   src: url("../fonts/montserrat/Montserrat-Semibold.woff2") format('woff2'),
10      url("../fonts/montserrat/Montserrat-Semibold.woff") format('woff');
11 }
12
13 @font-face{
14   font-family: "Montserrat-Medium";
15   src: url("../fonts/montserrat/Montserrat-Medium.ttf") format("opentype"),
16       url("../fonts/montserrat/Montserrat-Medium.woff2") format('woff2'),
17       url("../fonts/montserrat/Montserrat-Medium.woff") format('woff');
18 }
19
20

```

Figure VII. 1 : inclusion des polices

Chaque police est déclarée dans une balise « `@font-face` ». Cette balise permet de donner un nom à une police, ainsi que d'indiquer le chemin vers le fichier de police. Le nom est dans la propriété « `font-family` » et le chemin ainsi que le format du fichier dans la propriété « `src` ».

Les polices étant maintenant incluses dans le projet, il ne restait plus qu'à les utiliser. Pour le faire, il suffit d'indiquer dans la propriété « `font-family` » d'un élément le nom de la police. Par exemple pour le texte de substitution, nous voulions utiliser la police Montserrat-Semibold :

```

32 input::placeholder, textarea::placeholder{
33   font-size: 16px;
34   font-family: "Montserrat-Semibold", Arial;
35   color: #ACACAC;
36 }

```

Figure VII. 2 : utilisation d'une police

Nous avons néanmoins rajouté la police Arial dans le cas où le navigateur n'arriverait pas à charger la police Montserrat.

## VII. 4) Eléments du formulaire

### VII. 4) a. Conteneur principal

Nous cherchions à obtenir un formulaire ne prenant pas tout l'espace visible disponible, mais seulement la partie centrale de l'écran.

Tout le code HTML concernant le formulaire est englobé dans une balise « `div` » possédant l'identifiant « `main-container` ». C'est sur cette balise que nous modifions le style.

```
41 #mainContainer{
42   width: 65%;
43   margin: auto;
44   padding: 20px 40px 60px 40px;
45   background-color: white;
46   border-radius: 10px;
47 }
```

Figure VII 3. : style du conteneur principal

Nous lui avons donné une largeur de 65% de la taille totale (`width : 65%`), un fond blanc et un arrondissement des angles de sa bordure. La valeur « `auto` » associée à la propriété « `margin` » permet de centrer l'élément. La propriété « `padding` » nous a permis d'ajouter des marges internes.

La couleur de fond blanche permet de dissocier le conteneur principal du reste, qui possède un fond gris clair. (cf. propriété « `background-color` » de l'élément « `body` »).

#### VII. 4) b. Apparence des champs

Nous souhaitions obtenir des champs possédant un fond gris très clair afin de les dissocier du conteneur, avec des angles arrondis. Chaque champ était inclus dans une division possédant la classe « `entry` ».

```
33 .entry, select {
34   background-color: #F7F7F7;
35   padding: 15px;
36   border: 1px solid #E6E6E6;
37   border-radius: 15px;
38   margin: 10px 0px 15px 0px;
39 }
```

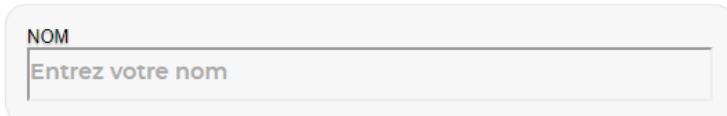
Figure VII 4. : style des champs

Nous retrouvons les mêmes propriétés que pour le conteneur principal, à savoir « `border-radius` » ainsi que « `background-color` ». La couleur `#F7F7F7` correspond au gris clair.

Nous avons rajouté quelques propriétés :

- « `margin` » : Nous a permis d'ajouter une marge extérieure pour que les champs ne se touchent pas.
- « `padding` » : Ajout d'une marge intérieure pour que le texte ne touche pas les bordures.
- « `border` » : Pour changer l'apparence de la bordure.

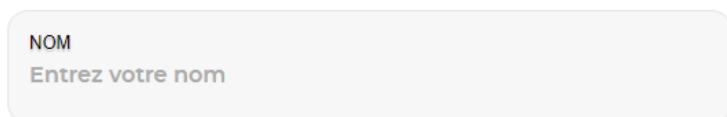
On obtient des champs ressemblant au champ suivant :



NOM  
Entrez votre nom

Figure VII 5. : allure des champs avec la bordure d'un input de type text

Dernier problème ; la bordure de base des input text donne un affichage moins agréable. Pour la supprimer nous avons appliqué la valeur « hidden » (= caché) à la propriété « border » des input.



NOM  
Entrez votre nom

Figure VII 6. : allure des champs sans la bordure du input text

Pour la liste déroulante, celle de base n'étant pas très ergonomique, nous avons choisi de la styliser aussi. Ne trouvant pas d'autres idées, nous avons décidé d'utiliser le même style que pour les champs. C'est pourquoi toutes ces propriétés ont aussi été appliquées à élément « select » de la liste déroulante du champ catégorie.



Figure VII 7. : liste déroulante de basee, avant l'application du stylee



Figure VII 8. : liste après l'application du style

## VII. 4) c. Placement des champs e-mail et téléphone

Les champs e-mail et téléphone devaient être l'un à côté de l'autre, sur la même ligne. Il a donc fallu rajouter quelques lignes de code.

Encore une fois, nous avons encapsulé les champs e-mail et téléphone dans des divisions ayant pour identifiant « `champ_email` » et « `champ_phone` ».

```

70  #champ_email, #champ_phone
71  {
72      display: inline-block;
73      width: 49%;
74  }
75
76 #champ_phone{
77     float: right;
78 }
```

Figure VII 9. : alignement des champs email et téléphone

Pour les aligner nous avons utilisé trois propriétés :

- « `display` » : sur les deux champs, la valeur « `inline-block` » indique à la page que ces divisions peuvent se placer sur la même ligne.
- « `width` » : la valeur « `49%` » donne une taille d'à peine la moitié du conteneur principal à chacun des champs. Puisque l'addition de leur taille ne dépasse pas la taille totale de la ligne, ils s'afficheront sur une seule ligne.
- « `float` » : la valeur « `right` » impose au champ de s'aligner à droite.

## VII. 4) f. Bouton envoyer

Pour le bouton envoyer, nous souhaitions qu'il prenne presque toute la largeur du conteneur principal, et qu'il affiche un dégradé amenant un changement de couleur lors du survol (le bouton passe progressivement du noir au vert).

L'input de type « `button` » possède la classe « `submitArea` » dans le HTML.

```

90 .submitArea{
91     border: none;
92     padding: 15px 70px 15px 70px;
93     background-color: #333333;
94     color: white;
95     border-radius: 20px;
96     width: 100%;
97
98     transition: background-color 0.4s;
99 }
100
101 .submitArea:hover{
102     background-color: #00AD5F;
103 }
```

Figure VII 10. : style du bouton d'envoi

Le code CSS du bouton se divise en deux parties :

- L'affichage du bouton en général
- L'affichage du bouton quand l'utilisateur a son curseur sur celui-ci

Le code de l'affichage général utilise les mêmes propriétés que les éléments précédents, à savoir un « border-radius » pour avoir les angles arrondis, un « background-color » pour obtenir le fond noir, puis une modification de la taille et des marges à l'aide de « width » et « padding ».

Ce qui est intéressant ici est la propriété « transition ». Elle permet de contrôler la vitesse d'animation lorsque les propriétés CSS sont modifiées. Ici, on indique que : lorsque la propriété « background-color » change, il doit y avoir une transition de 0,4 seconde. Grâce à cela, on obtient un dégradé du noir vers le vert quand l'utilisateur survole le bouton, et du vert vers le noir quand l'utilisateur enlève son curseur du bouton.

## VII. 5) Résultat final

Après avoir appliqué le CSS au HTML, nous obtenons la page suivante :

The screenshot shows a contact form titled "Nous contacter" in a stylized font at the top. The form consists of several input fields:

- A large input field for "NOM" (Name) with placeholder text "Entrez votre nom".
- A two-column layout for "E-MAIL" (Email) and "TELEPHONE" (Phone) with placeholder text "Entrez votre e-mail" and "Entrez votre numéro de téléphone" respectively.
- A dropdown menu for "CATEGORIE SOCIOPROFESSIONNELLE" (Professional Category) with placeholder text "Sélectionnez le groupe auquel vous appartenez".
- A large input field for "MESSAGE" (Message) with placeholder text "Tapez votre message ici...".
- A black button at the bottom labeled "Envoyer →" (Send).

Figure VII 11. : Rendu de la page avec le CSS

## VIII. Le JavaScript (JS)

Cette partie fait référence à l'annexe 5.

### VIII. 1) Pourquoi avoir fait du JavaScript ?

#### VII. 1) a. Avantages

Grâce à JavaScript, nous pouvons manipuler le formulaire et ses données directement côté navigateur client, avant éventuellement d'envoyer ces données vers un serveur externe. Ainsi, on peut avertir immédiatement l'utilisateur en cas de saisie erronée. Si c'est le cas, le formulaire n'est pas envoyé et on reste sur la page du formulaire, ce qui permet donc :

- d'éviter les redirections, et donc les temps de chargements qui vont avec (qui peuvent être longs si le serveur est saturé) ;
- de ne pas être redirigé sur un formulaire vierge, et donc ne pas avoir à tout retaper

Par ailleurs, le JS permet de prévenir l'utilisateur qu'un champ est mal rempli dès que qu'il a fini de le remplir, et non pas lorsqu'il clique sur le bouton « Envoyer » du formulaire.

Ainsi, le JS a l'avantage d'être très agréable pour l'utilisateur, et c'est pourquoi nous avons décidé de l'incorporer au sein de notre formulaire.

#### VIII. 1) b. ... Mais le JavaScript n'est pas fiable

Une vérification en JS n'est pas fiable, pour la simple et bonne raison qu'il est très facile de désactiver le JS. En effet, le JavaScript est un langage qui s'exécute côté client, or le client peut très bien modifier son comportement ou bien même le désactiver, ce qui annulerait les vérifications.

Par conséquent, le test du contenu d'un formulaire en Javascript doit être seulement considéré comme un plus pour le confort de l'utilisation de la page, afin d'éviter un envoi et un recharge du formulaire si son contenu est invalide. Il ne remplace en aucun cas une vérification côté serveur, qui est indispensable et doit être systématique afin de palier le cas où le Javascript n'est pas activé, donc au cas où le formulaire n'est pas vérifié côté client.

## VIII. 2) Masquer/afficher les messages de correction

### VIII. 2) a. Fonctions pour masquer et afficher un élément

```
1 // Fonction qui masque l'élément
2 function cache(elementStyle){
3     elementStyle.display = "none" ;
4 }
5
6 // Fonction qui affiche l'élément
7 function affiche(elementStyle){
8     elementStyle.display = "";
9 }
```

Figure VIII. 1 : code des fonctions JS pour masquer et afficher un élément

Principe : La fonction `cache()`, resp. `affiche()`, prend en argument `elementStyle` qui est le `.style` d'un élément. Cette fonction modifie la propriété `display` à « `none` », resp. « `» », afin de masquer, resp. afficher, cet élément.`

Notons qu'une autre option est possible pour masquer des éléments ; c'est `visibility: hidden;`. Néanmoins, `display: none;` masque totalement l'élément et annule des propriétés telles que `margin`, `padding`, `width`, `height...` tandis que `visibility: hidden;` masque seulement l'élément, ce qui peut laisser des espaces vides. Nous avons donc choisi `display: none;` afin de ne pas avoir d'espaces vides.

### VIII. 2) b. Cacher les messages de correction

Dans notre code HTML nous avons créé des balises `<div>` avec la classe `.hiddenMessage`. Comme le nom de la classe l'indique, ce sont des messages de correction cachés qui seront dévoilés seulement pour informer l'utilisateur du format attendu de chaque champ du formulaire, si jamais ce qu'il rentre n'est pas correct. Seulement, ces balises sont affichées par défaut et il nous faut donc les cacher par le biais du JavaScript.

Nous avons décidé d'afficher par défaut et masquer grâce au JavaScript et pas l'inverse : nous n'avons pas caché par défaut ces balises pour les afficher via le Javascript. En effet, si nous avions procédé ainsi, nous prenions le risque qu'un utilisateur ayant désactivé le JavaScript ne puisse pas voir les messages de correction.

Voici le code de la fonction `desactivHidden()` contenu dans main.js :

```

11 // Fonction de désactivation de l'affichage des messages de correction
12 // ("hiddenMessage" class)
13 function desactivHidden() {
14
15     var hiddenMessage = document.querySelectorAll('.hiddenMessage'),
16         hiddenMessageLength = hiddenMessage.length;
17
18     for (var i = 0; i < hiddenMessageLength; i++) {
19         cache(hiddenMessage[i].style);
20     }
21 }
```

Figure VIII. 2 : fonction JS de désactivation des messages de correction

Principe : Une variable `hiddenMessage` est un tableau contenant tous les objets de classe `.hiddenMessage` contenu dans le document HTML. Pour chaque objet de ce tableau, on modifie la propriété `display` à « `none` » grâce à la fonction `cache()` définie précédemment.

Cette fonction est appelée après le chargement de la page, afin de cacher les messages avant que l'utilisateur ne commence à remplir le formulaire, d'où son appellation directement dans le main.js, qui lui-même est appelé à la fin du `<body>`. En effet, une page Web est lue par le navigateur de façon linéaire, c'est-à-dire qu'il lit d'abord le `<head>`, puis les éléments du `<body>` les uns à la suite des autres. Ainsi, les éléments `<script>` seront appellés après le chargement des éléments de la page.

### VIII. 3) Vérification du nom

#### VIII. 3) a. Code HTML

```

<!-- Nom -->
<div class="entry">
    <span id="nomLabel" class="labelChamp">NOM</span>
    <input name="Nom" id="nomEntry" type="text" placeholder="Entrez votre nom" size="30"
        onblur="checkNom()" onkeydown="editionNom()">
    <div class="hiddenMessage" id="messageNom">
        Un nom doit faire plus de 2 caractères et ne contient que des lettres
    </div>
</div>
```

Figure VIII. 3 : code HTML relatif au champ Nom du formulaire

L'évènement `onblur` fait appel à la fonction JS `checkNom()` lorsque le champ relatif au nom sur lequel il est appliqué perd le focus, i.e. quand l'utilisateur quitte le champ.

L'évènement `onkeydown` fait appel à la fonction JS `editionNom()` dès que l'utilisateur frappe une touche du clavier dans le champ relatif au nom.

### VIII. 3) b. Code JS : checkNom()

```

29 function checkNom() {
30     var nomInput = document.getElementById('nomEntry'), //L'input
31     nomMessageStyle = document.getElementById('messageNom').style; //Le style du message
32
33     if (nomInput.value.length > 2 && /^[A-Za-z]+$/i.test(nomInput.value) ) {
34         nomInput.className = 'OK'; // changement de class
35         cache(nomMessageStyle); // on masque le message de correction
36         return 1;
37     } else {
38         nomInput.className = 'messageHidden';
39         affiche(nomMessageStyle); //on affiche le message de correction
40     }
41     return 0;
42 }
```

Figure VIII. 4 : fonction JS de vérification du champ Nom du formulaire

Principe : La variable `nomInput` récupère l'input relatif au champ Nom du formulaire et la variable `nomMessageStyle` récupère le style associé. On vérifie si le nom fait plus de 2 caractères et grâce à une regex on vérifie que ces caractères ne soient que des lettres majuscules ou minuscules. Si c'est le cas, i.e. si le nom rentré vérifie ces critères, alors la classe de `nomInput` change et l'on masque le message de correction relatif au champ Nom grâce à la fonction `cache()`. Si ce n'est pas le cas, la classe de `nomInput` change et on affiche le message de correction relatif au champ Nom grâce à la fonction `affiche()`.

### VIII. 3) c. Code JS : editionNom()

```

24 function editionNom() {
25     var nomInput = document.getElementById('nomEntry');
26     nomInput.className = 'edition';
27 }
```

Figure VIII. 5 : fonction JS indiquant si le nom est en train d'être édité

Cette fonction permet de savoir si le champ nom est en cours d'édition, afin de changer l'affichage pendant la modification. Si l'utilisateur rentre un nom invalide, il s'affichera en rouge. Quand il revient dans le champ « Nom » pour le modifier, on ne veut pas que ce qu'il tape reste en rouge pendant la modification, mais qu'il devienne noir tant qu'il n'a pas fini d'édition.

## VIII. 4) Vérification du mail

### VIII. 4) a. Code HTML

```
<!-- Email -->
<div class="entry" id="champ_email">
  <span id="emailLabel" class="labelChamp">E-MAIL</span>
  <br/>
  <input name="Email" id="emailEntry" type="text" placeholder="Entrez votre e-mail" size="30"
    | onblur="checkMail()" onkeydown="editionMail()">
  <div class="hiddenMessage" id="messageMail">
    Entrez un e-mail valide
  </div>
</div>
```

Figure VIII. 6 : code HTML relatif au champ Mail du formulaire

Le code HTML est similaire à la section précédente, avec l'événement `onblur` qui fait appel à la fonction JS `checkMail()` et l'événement `onkeydown` qui fait appel à la fonction `editionMail()`.

### VIII. 4) b. Code JS : checkMail()

```
50 function checkMail() {
51   var mailInput = document.getElementById('emailEntry'), //L'input
52     mailMessageStyle = document.getElementById('messageMail').style; //Le style du message
53
54   if (/^[_a-z0-9_-]+@[_a-z0-9._-]+\.[a-z]{2,6}$/ .test(mailInput.value)) {
55     mailInput.className = 'OK'; // changement de class
56     cache(mailMessageStyle); // on masque le message de correction
57     return 1;
58   } else {
59     mailInput.className = 'messageHidden';
60     affiche(mailMessageStyle); //on affiche le message de correction
61   }
62   return 0;
63 }
```

Figure VIII. 7 : fonction JS de vérification du champ Mail du formulaire

Le principe est le même que la fonction de vérification précédente, hormis le fait que cette fois on vérifie si le contenu correspond à la forme d'un mail. On vérifie donc par une regex qu'il y ait bien :

- Tout d'abord un pseudonyme (au minimum une lettre) constitué uniquement de lettres minuscules, chiffres, points, tirets et des underscores.
- Ensuite, un arrobase.
- Puis, un nom de domaine, ayant les mêmes règles que le pseudonyme hormis qu'il y a cette fois au minimum deux caractères.
- Enfin, une extension (comme « .fr »), qui comporte un point, suivi de 2 à 4 lettres minuscules. (Il existe « .es », « .de », mais aussi « .com », « .net », « .info », etc.)

### VIII. 4) c. Code JS : editionMail()

```

44  function editionMail() {
45      var mailInput = document.getElementById('emailEntry');
46      mailInput.className = 'edition';
47  }
48

```

Figure VIII. 8 : fonction JS indiquant si le mail est en train d'être édité

Fonctionnement identique au nom.

### VIII. 5) Vérification du téléphone

#### VIII. 5) a. Code HTML

```

<!-- Phone -->
<div class="entry" id="champ_phone">
  <span id="phoneLabel" class="labelChamp">TELEPHONE</span>
  <br/>
  <input name="Telephone" id="phoneEntry" type="text" placeholder="Entrez votre numéro de téléphone"
         size="50" onblur="checkTel()" onkeydown="editionTel()">
  <div class="hiddenMessage" id="messageTel">
    Entrez un numéro de téléphone valide
  </div>
</div>

```

Figure VIII. 9 : code HTML relatif au champ Téléphone du formulaire

Le code HTML est similaire à la section précédente, avec l'événement `onblur` qui fait appel à la fonction JS `checkTel()` et l'événement `onkeydown` qui fait appel à la fonction `editionTel()`.

#### VIII. 5) b. Code JS : checkTel()

```

71  function checkTel() {
72      var telInput = document.getElementById('phoneEntry'), //L'input
73          telMessageStyle = document.getElementById('messageTel').style; //Le style du message
74
75      if (/^0[1-68]([.- ]?[0-9]{2}){4}$/.test(telInput.value)) {
76          telInput.className = 'OK'; // changement de class
77          cache(telMessageStyle); // on masque le message de correction
78          return 1;
79      } else {
80          telInput.className = 'messageHidden';
81          affiche(telMessageStyle); //on affiche le message de correction
82      }
83      return 0;
84  }

```

Figure VIII. 10 : fonction JS de vérification du champ Téléphone du formulaire

Le principe est le même que la fonction de vérification précédente, hormis le fait que cette fois on vérifie si le contenu correspond à la forme d'un numéro de téléphone français. On vérifie donc par une regex qu'il y ait bien :

- le premier chiffre qui est toujours 0 ;
- le second chiffre va de 1 à 8 : 1 à 5 pour les fixes de différentes régions, 6 et 7 pour les téléphones portables, mais il y a aussi le 8 pour les numéros spéciaux ;
- ensuite viennent les 8 chiffres restants allant de 0 à 9.
- Il y a possibilité d'avoir un tiret, un point, un espace ou rien entre chaque paire de chiffres.

### VIII. 5) c. Code JS : editionTel()

```
65  function editionTel() {  
66      var telInput = document.getElementById('phoneEntry');  
67      telInput.className = 'edition';  
68  }
```

Figure VIII. 11 : fonction JS indiquant si le téléphone est en train d'être édité

Fonctionnement identique au nom/mail.

### VIII. 6) Vérification du message

#### VIII. 6) a. Code HTML

```
<!-- Message -->  
<div id="champ_message">  
    <span id="messageLabel" class="labelChamp">MESSAGE</span>  
    <br/>  
    <textarea name="Message" id="messageEntry" cols="100" rows="2"  
        placeholder="Tapez votre message ici..." onblur="checkMessage()"></br>  
    </textarea>  
    <div class="hiddenMessage" id="messageMessage">  
        Votre message doit contenir entre 20 et 1200 caractères  
    </div>  
</div>
```

Figure VIII. 12 : code HTML relatif au champ Message du formulaire

Le code HTML est similaire à la section précédente, avec l'événement `onblur` qui fait appel à la fonction JS `checkMessage ()`.

## VIII. 6) b. Code JS : checkMessage()

```
87 function checkMessage() {
88     var messageInput = document.getElementById('messageEntry'), //L'input
89         messageMessageStyle = document.getElementById('messageMessage').style, //Le style du message
90         Longueur = messageInput.value.length; //longueur du texte tapé
91
92     if (Longueur >= 20 && Longueur <= 1200) {
93         messageInput.className = ''; // changement de class
94         cache(messageMessageStyle); // on masque le message de correction
95         return 1;
96     } else {
97         messageInput.className = 'messageHidden';
98         affiche(messageMessageStyle); //on affiche le message de correction
99     }
100    return 0;
101 }
```

Figure VIII. 13 : fonction JS de vérification du champ Message du formulaire

Le principe est le même que la fonction de vérification précédente, hormis le fait que cette fois on vérifie si le message fait au minimum 20 caractères et au maximum 1200.

De plus, on ne fait pas appel à une fonction « editionMessage » puisqu'on ne veut pas effectuer la vérification à chaque fois que l'utilisateur tape un caractère, mais seulement quand il a fini son message. La vérification est donc faite à la fin, quand l'utilisateur change de champs.

### VIII. 7) Vérification de la catégorie

#### VIII. 7) a. Code HTML

```
<div class="entry">
    <span id="catLabel" class="labelChamp">CATÉGORIE SOCIOPROFESSIONNELLE</span>
    <br/>
    <select name="Categorie" id="catEntry" onchange="checkCat()">
        <option value="0">Sélectionnez le groupe auquel vous appartenez</option>
        <option value="1">1. Agriculteurs exploitants</option>
        <option value="2">2. Artisans, commerçants, chefs d'entreprises</option>
        <option value="3">3. Cadres et professions intellectuelles supérieures</option>
        <option value="4">4. Professions intermédiaires</option>
        <option value="5">5. Employés</option>
        <option value="6">6. Ouvriers</option>
        <option value="7">7. Retraités</option>
        <option value="8">8. Ouvriers</option>
        <option value="9">9. Sans activité</option>
    </select>
    <div class="hiddenMessage" id="messageCat">
        Vous devez sélectionner votre catégorie socioprofessionnelle
    </div>
</div>
```

Figure VIII. 14 : code HTML relatif au champ Catégorie du formulaire

L'évènement `onchange` fait appel à la fonction JS `checkCat()` lorsque l'utilisateur fait un choix de catégorie.

#### VIII. 7) b. Code JS : checkCat()

```
103 function checkCat() {
104     var catInput = document.getElementById('catEntry'), //L'input
105         messageCatStyle = document.getElementById('messageCat').style; //Le style du message
106
107     if (catInput.value != "0") {
108         cache(messageCatStyle); // on masque le message de correction
109         return 1;
110     } else {
111         catInput.className = 'messageHidden';
112         affiche(messageCatStyle); //on affiche le message de correction
113     }
114     return 0;
115 }
```

Figure VIII. 15 : fonction JS de vérification du champ Catégorie du formulaire

Le principe est le même que la fonction de vérification précédente, hormis le fait que cette fois on vérifie si la catégorie est une autre que « Sélectionnez le groupe auquel vous appartenez », i.e. que l'utilisateur a bien choisi une catégorie.

Comme pour le message, on ne veut pas afficher l'erreur à chaque fois que l'utilisateur ne sélectionne aucune catégorie, mais seulement quand il n'en sélectionne pas et passe à un autre champ. C'est pourquoi il n'y a pas de fonction « `editionCategorie` ».

## VIII. 8) Vérification avant envoi

### VIII. 8) a. Code HTML

```
<p class="hiddenMessage" id="messageFormulaire">Des champs sont invalides</p>
<form id="formulaire" onsubmit="return Envoi()" method="post" action="envoi.php">
```

Figure VIII. 16 : code HTML relatif à l'envoi du formulaire

L'évènement `onsubmit` fait appel à la fonction `Envoi()` lorsque le formulaire est soumis et avant l'exécution de l'action, donc avant que l'on ne charge `envoi.php`. Il peut permettre de déterminer si l'action doit être exécutée (si `Envoi()` retourne `True`) ou non (si `Envoi()` retourne `False`).

### VIII. 8) b. Code JS : `Envoi()`

```
117 function Envoi(){
118     var messageFormulaireStyle = document.getElementById('messageFormulaire').style;
119
120     if(checkCat() == 1 & checkNom() == 1 & checkMail() == 1 & checkTel() == 1 & checkMessage() == 1){
121         cache(messageFormulaireStyle);
122         alert('Formulaire bien rempli');
123         return true;
124     }else{
125         affiche(messageFormulaireStyle);
126         return false;
127     }
128 }
```

Figure VIII. 17 : fonction JS de vérification du formulaire avant envoi

Principe: Si toutes les vérifications précédentes sont bonnes, alors le message d'erreur du formulaire est masqué grâce à la fonction `cache()`, une fenêtre avertit l'utilisateur que le formulaire est bien rempli, et `Envoi()` retourne `True`. Sinon, on affiche le message d'erreur grâce à `affiche()` et `Envoi()` retourne `False`.

### VIII. 9) Rendu avant envoi

Si la vérification n'est pas bonne à chaque champ du formulaire, voici un aperçu de ce qu'affiche la page, et ce, avant l'envoi :

The screenshot shows a contact form titled "Nous contacter". The form includes fields for Nom, E-mail, Téléphone, Catégorie socioprofessionnelle, and Message. Each field has an error message displayed below it in red text.

- NOM:** The input "bj" is invalid. The error message is: "Un nom doit faire plus de 2 caractères et ne contient que des lettres".
- E-MAIL:** The input "jeanPrl@hotmail.com" is invalid. The error message is: "Entrez un e-mail valide".
- TELEPHONE:** The input is empty. The error message is: "Entrez votre numéro de téléphone".
- CATEGORIE SOCIOPROFESSIONNELLE:** The dropdown menu is empty. The error message is: "Vous devez sélectionner votre catégorie socioprofessionnelle".
- MESSAGE:** The input "Projet de ZZ" is invalid. The error message is: "Votre message doit contenir entre 20 et 1200 caractères".

At the bottom right of the form area, there is a dark button labeled "Envoyer →". Below the form, the names and emails of the authors are listed:

CHAZOT Nahel nahel.chazot@etu.isima.fr  
NAUDIN Juliette juliette.naudin@etu.uca.fr

Figure VIII. 18 : rendu en cas d'erreur sur chaque champ, avant envoi

## IX. Le PHP

Cette partie fait référence à l'annexe 6.

### IX. 1) Pourquoi avoir fait du PHP ?

#### IX. 1) a. Définition

PHP est un autre langage informatique utilisé sur l'internet. C'est l'abréviation, en anglais, de « PHP Hypertext Preprocessor ». A la différence des langages HTML, CSS et JavaScript qui s'exécutaient côté client, c'est à dire dans le navigateur même des visiteurs, le PHP va s'exécuter côté serveur.

#### IX. 2) a. Rôle

Le PHP permet de créer des sites dynamiques et interactifs. Par exemple, nous pouvons charger une page différente pour chaque visiteur demandant l'accès à une URL.

Dans ce projet, le PHP va nous permettre de récupérer les données (si elles ont été saisies correctement) rentrées par l'utilisateur dans le formulaire, et d'afficher ses données sur une autre page.

#### IX. 3) Prérequis

Le PHP étant interprété par un serveur, un serveur est nécessaire pour développer en PHP. Pour ce projet, nous avons installé un serveur WAMP local, que l'on accède en tapant l'adresse <http://localhost/> ou <http://127.0.0.1/>.

La page du formulaire utilisée, `formulaire.php`, est la même que `formulaire.html` précédente, hormis le fait que son extension est donc maintenant `.php`.

### IX. 2) Comment envoyer les données saisies par l'utilisateur ?

```
<form id="formulaire" onsubmit="return Envoi()" method="post" action="envoi.php">
```

Figure IX. 1 : attributs de `<form>` relatifs au PHP

L'attribut `method` de `<form>` indique par quel moyen les données vont être envoyées. Il peut prendre deux valeurs : "get" ou "post". La différence entre ces deux méthodes est que GET fait transiter les données saisies dans le formulaire par la barre

d'adresse, contrairement à POST. Les données du formulaire pouvant être considérées comme sensibles, nous avons naturellement choisi la méthode POST.

### IX. 3) Une fois envoyées, comment traiter les données ?

L'attribut `action` de `<form>` sert à définir la page appelée par le formulaire, ici `envoi.php`. C'est cette page qui recevra les données du formulaire et qui sera chargée de les traiter. Comme expliqué précédemment, avant l'envoi, c'est `onsubmit` qui fait appel à la fonction `Envoi()` et qui permet à l'attribut `action` de rediriger vers `envoi.php` si la fonction renvoie `True`.

### IX. 4) Page de traitement des données : `envoi.php`

#### IX. 4) a. Vérification de l'existence et du caractère non nul des données

```

10 <?php
11 // S'il y des données de postées
12 if(!empty($_POST['submit'])) {
13
14     if (!empty($_POST['Nom']) && !empty($_POST['Email'])
15         && !empty($_POST['Telephone']) && !empty($_POST['Categorie'])
16         && !empty($_POST['Message']))
17     { ...
18     }else{
19         echo "Les données saisies sont incorrectes. Veuillez réessayer.";
20     }
21     }else{
22         echo "Une erreur est survenue. Veuillez réessayer.";
23     }
24
25 ?>

```

Figure IX. 2 : vérification de variables existantes et non nulles

Premièrement, il faut vérifier que l'utilisateur arrive sur cette page `envoi.php` suite à l'envoi du formulaire par `formulaire.php`, et non pas qu'il ait tapé lui-même l'adresse dans la barre d'adresse du navigateur ".../envoi.php". Pour cela, vérifie s'il y a bien eu un clic sur le bouton d'envoi du formulaire (d'attribut `name="submit"`) grâce à la variable `$_POST['submit']` et la fonction `empty()`. `Empty()` permet de vérifier si la variable est vide, nulle, ou non définie. Si ce n'est pas le cas, la page web affiche un message d'erreur.

Une fois passé ce test, toujours avec `empty()`, on vérifie que les champs du formulaires ont été remplis. Cette vérification est nécessaire dans le cas où l'utilisateur n'utiliserait pas JavaScript. Si les champs du formulaire ne sont pas remplis intégralement, la page web affiche un message d'erreur.

**NB :** Nous nous sommes contentés ici de ne vérifier que l'existence et le caractère non nul des données, mais nous avons conscience que cela n'est pas suffisant pour un vrai site web. Il faudrait également vérifier la conformité des données, i.e. faire le même travail qu'en JavaScript pour chaque champ du formulaire, mais cette fois-ci côté serveur, en PHP.

#### IX. 4) b. Récupération des données

```
// Récupération des variables
$nom = htmlentities($_POST['Nom']);
$email = htmlentities($_POST['Email']);
$phone = htmlentities($_POST['Telephone']);
$categorie = htmlentities($_POST['Categorie']);
$message = htmlentities($_POST['Message']);

echo "Vos données ont bien été prises en compte.";
echo "</br></br>";
echo "<center>Résultat du questionnaire :</center>";
echo "<br/>";
echo "<u>Nom</u> : ".$nom ;
echo "<br/>";
echo "<u>E-mail</u> : ".$email ;
echo "<br/>";
echo "<u>Téléphone</u> : ".$phone ;
echo "<br/>";
echo "<u>Catégorie socioprofessionnelle</u> : n°".$categorie ;
echo "<br/>";
echo "<br/>";
echo "Votre message :";
echo "<br/>";
echo $message;
```

Figure IX. 3 : récupération des données dans des variables par php

Après avoir vérifié que des données conformes avaient bien été envoyées, nous les récupérons en les stockant dans différentes variables : \$nom, [...], \$message.

Nous utilisons la fonction `htmlentities()` pour se prémunir des attaques qui consistent à injecter du code générant des comportements non prévus sur notre site web. On appelle ce type d'attaque « Attaque XSS » ou « Cross-Site Scripting ». La fonction `htmlentities()` décode les caractères qui ont des significations spéciales en HTML. Par exemple, `htmlentities('Un apostrophe en <strong>gras</strong>')` affichera « Un apostrophe en &lt;strong&gt;gras&lt;/strong&gt; ».

Nous affichons ensuite sur la page les données que l'utilisateur a rentré dans le formulaire.

## IX. 5) Rendu après envoi

IX. 5) a. Si l'utilisateur se rend sur `envoi.php` sans passer par le formulaire

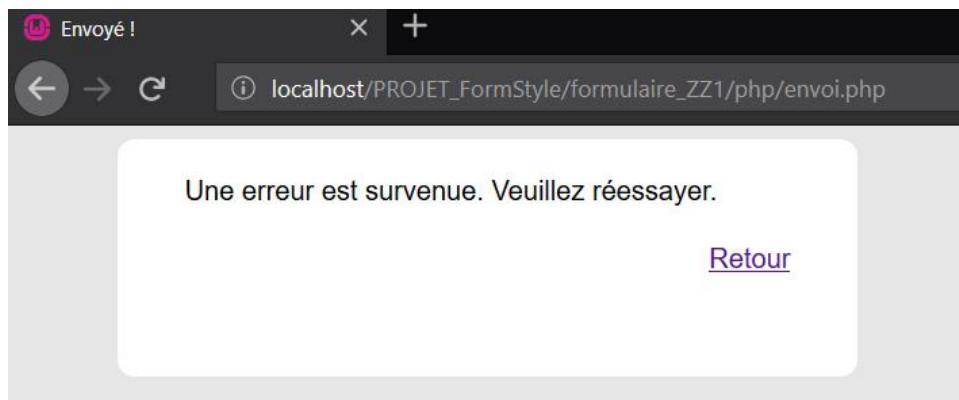


Figure IX. 4 : rendu si l'utilisateur ne passe pas par le formulaire (erreur)

IX. 5) b. Si l'utilisateur envoie des données correctes via le formulaire

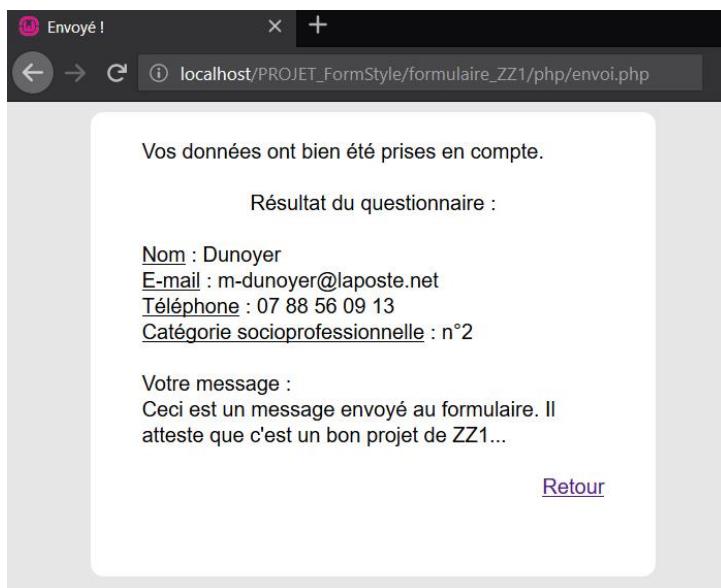


Figure IX. 5 : rendu lorsque l'utilisateur envoie des données correctes via le formulaire

## ANNEXES

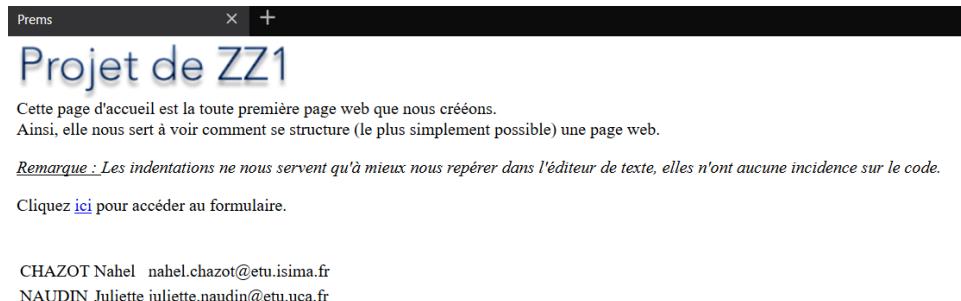
Annexe 1 : supports pour le tutoriel de HTML.....	57
Annexe 2 : arborescence du projet .....	59
Annexe 3 : code HTML.....	60
Annexe 4 : code CSS.....	63
Annexe 5 : code JavaScript .....	66
Annexe 6 : code PHP.....	69

## Annexe 1 : supports pour le tutoriel de HTML

accueil.html :

```
accueil.html
1 <html>
2   <head>
3     <!-- En-tête de la page -->
4     <meta charset="utf-8" />
5     <title> Prem's </title>
6   </head>
7
8   <body> <!-- Corps de la page -->
9
10    
11    <br/>
12    Cette page d'accueil est la toute première page web que nous créons.
13    <br/>
14    Ainsi, elle nous sert à voir comment se structure (le plus simplement possible) une page web.
15
16    <p><i>
17      <u> Remarque : </u> Les indentations ne nous servent qu'à mieux nous repérer dans l'éditeur
18      de texte, elles n'ont aucune incidence sur le code.
19    </i></p>
20
21    Cliquez <a href="formulaire.html">ici</a> pour accéder au formulaire.
22
23    <footer> <!-- Pied de page -->
24      <br/><br/>
25      <table>
26        <tr> <!-- Début 1ere ligne -->
27          <td> CHAZOT </td>
28          <td> Nahel </td>
29          <td> nahel.chazot@etu.isiima.fr </td>
30      </tr><!-- Fin 1ere ligne -->
31      <tr> <!-- Début 2eme ligne -->
32          <td> NAUDIN </td>
33          <td> Juliette </td>
34          <td> juliette.naudin@etu.uca.fr </td>
35      </tr><!-- Fin 2eme ligne -->
36    </table>
37
38  </footer>
39
40  </body>
41
42
43 </html>
```

Rendu sur navigateur de accueil.html :



## Annexe 2 : arborescence du projet

```
| Rapport.docx
| README.md

+---css
|   main.css
|   police.css

+---fonts
|   \---montserrat
|       Montserrat-Black.ttf
|       Montserrat-Black.woff.woff
|       Montserrat-Black.woff2
|       Montserrat-Medium.ttf
|       Montserrat-Medium.woff
|       Montserrat-Medium.woff2
|       Montserrat-SemiBold.ttf
|       Montserrat-Semibold.woff
|       Montserrat-Semibold.woff2

+---html
|   formulaire.html

+---images
|   NousContacter.png
|   projetZZ.png

+---js
|   main.js

\---php
    envoi.php
    formulaire.php
```

### Annexe 3 : code HTML

Header et corps de la page (formulaire.html) :

```
1 <html>
2
3     <head>
4         <!-- En-tête de la page -->
5         <meta charset="utf-8" />
6         <link rel="stylesheet" type="text/css" href="../css/main.css">
7         <link rel="stylesheet" type="text/css" href="../css/police.css">
8         <title> Contact </title>
9
10    </head>
11
12
13    <body>
14        <div id="mainContainer">
15            <center></center>
16            <p class="hiddenMessage" id="messageFormulaire">Des champs sont invalides</p>
17
18        <form id="formulaire" onsubmit="return Envoi()" method="post" action="envoi.php">
19            <!-- Nom -->
20            <div class="entry">
21                <span id="nomLabel" class="labelChamp">NOM</span>
22                <input name="Nom" id="nomEntry" type="text" placeholder="Entrez votre nom" size="30"
23                    onblur="checkNom()" onkeydown="editionNom()">
24                <div class="hiddenMessage" id="messageNom">
25                    Un nom doit faire plus de 2 caractères et ne contient que des lettres
26                </div>
27            </div>
28
29            <div id="line2">
30                <!-- Email -->
31                <div class="entry" id="champ_email">
32                    <span id="emailLabel" class="labelChamp">E-MAIL</span>
33                    <br/>
34                    <input name="Email" id="emailEntry" type="text" placeholder="Entrez votre e-mail" size
35                        ="30" onblur="checkMail()" onkeydown="editionMail()">
36                    <div class="hiddenMessage" id="messageMail">
37                        Entrez un e-mail valide
38                    </div>
39                </div>
40            </div>
41        </form>
42    </body>
43
```

```
38
39      <!-- Phone -->
40      <div class="entry" id="champ_phone">
41          <span id="phoneLabel" class="labelChamp">TELEPHONE</span>
42          <br/>
43          <input name="Telephone" id="phoneEntry" type="text" placeholder="Entrez votre numéro
44              de téléphone" size="50" onblur="checkTel()" onkeydown="editionTel()"/>
45          <div class="hiddenMessage" id="messageTel">
46              Entrez un numéro de téléphone valide
47          </div>
48      </div>
49
50      <!-- Needed services -->
51      <div class="entry">
52          <span id="catLabel" class="labelChamp">CATÉGORIE SOCIOPROFESSIONNELLE</span>
53          <br/>
54          <select name="Categorie" id="catEntry" onchange="checkCat()">
55              <option value="0">Sélectionnez le groupe auquel vous appartenez</option>
56              <option value="1">1. Agriculteurs, exploitants</option>
57              <option value="2">2. Artisans, commerçants, chefs d'entreprises</option>
58              <option value="3">3. Cadres et professions intellectuelles supérieures</option>
59              <option value="4">4. Professions intermédiaires</option>
60              <option value="5">5. Employés</option>
61              <option value="6">6. Ouvriers</option>
62              <option value="7">7. Retraités</option>
63              <option value="8">8. Ouvriers</option>
64              <option value="9">9. Sans activité</option>
65          </select>
66          <div class="hiddenMessage" id="messageCat">
67              Vous devez sélectionner votre catégorie socioprofessionnelle
68          </div>
69      </div>
70
71      <!-- Message -->
72      <div id="champ_message">
73          <span id="messageLabel" class="labelChamp">MESSAGE</span>
74          <br/>
75          <textarea name="Message" id="messageEntry" cols="100" rows="2" placeholder="Tapez
76              votre message ici..." onfocusout="checkMessage()"/>
77          <div class="hiddenMessage" id="messageMessage">
78              Votre message doit contenir entre 20 et 1200 caractères
79          </div>
80      </div>
81
82      <!-- Button -->
83      <div class="submitArea2">
84          <input name="submit" type="submit" value="Envoyer" id="submitButton" class=""
85              submitArea">
86      </div>
87
88  </form>
89  </div>
```

Footer (formulaire.html):

```
89     <footer> <!-- Pied de page -->
90         <br/>
91         <table>
92             <tr> <!-- Début 1ere ligne -->
93                 <td> CHAZOT </td>
94                 <td> Nahel </td>
95                 <td> nahel.chazot@etu.isima.fr </td>
96             </tr><!-- Fin 1ere ligne -->
97             <tr> <!-- Début 2eme ligne -->
98                 <td> NAUDIN </td>
99                 <td> Juliette </td>
100                <td> juliette.naudin@etu.uca.fr </td>
101            </tr><!-- Fin 2eme ligne -->
102        </table>
103
104    </footer>
105
106
107    <script src="../js/main.js"></script>
108 </body>
109 </html>
```

#### Annexe 4 : code CSS

police.css :

```
1  @font-face{
2    font-family: "Montserrat-Black";
3    src: url("../fonts/montserrat/Montserrat-Black.woff2") format('woff2'),
4         url("../fonts/montserrat/Montserrat-Black.woff") format('woff');
5  }
6
7  @font-face{
8    font-family: "Montserrat-SemiBold";
9    src: url("../fonts/montserrat/Montserrat-Semibold.woff2") format('woff2'),
10       url("../fonts/montserrat/Montserrat-Semibold.woff") format('woff');
11  }
12
13  @font-face{
14    font-family: "Montserrat-Medium";
15    src: url("../fonts/montserrat/Montserrat-Medium.ttf") format("opentype"),
16         url("../fonts/montserrat/Montserrat-Medium.woff2") format('woff2'),
17         url("../fonts/montserrat/Montserrat-Medium.woff") format('woff');
18  }
19
20
21 body{
22   background-color: #E6E6E6;
23   font-family: Arial;
24 }
25
26 footer{
27   font-size: 8px;
28   text-align: center;
29 }
30
31
32 input::placeholder, textarea::placeholder{
33   font-size: 16px;
34   font-family: "Montserrat-Semibold", Arial;
35   color: #ACACAC;
36 }
37
38 input[type=text], #messageEntry{
39   font-size: 14px;
40   font-family: "Montserrat-SemiBold", Tahoma;
41   color: grey;
42 }
43
44 #catEntry{
45   font-family: "Montserrat-Medium", Arial;
46 }
47
48 .labelChamp{
49   font-size: 13px;
50   text-shadow: 0px 2px 1px #C3C3C3;
51 }
52
53 .hiddenMessage{
54   font-size: 14px;
55 }
56
57 .OK, .NOTOK{
58   font-size: 14px;
59 }
60
61 .OK{
62   color: green;
63 }
64
65 .NOTOK, .hiddenMessage{
66   color: red;
67 }
68
69 #messageCat{
70   color: red;
71 }
```

main.css :

```
1 footer table{
2     margin: auto;
3 }
4
5 footer td{
6     padding: 0px 10px 0px 0px ; /* marge intérieures : haut droite bas gauche */
7 }
8
9 input:focus, textarea:focus, select:focus{
10     outline: 0;
11 }
12
13 textarea {
14     resize: none;
15 }
16
17 input {
18     border: hidden;
19     background-color: #F7F7F7;
20 }
21
22 textarea{
23     background-color: white;
24     padding: 15px;
25     border: none;
26     margin: 10px 0px 15px 0px;
27 }
28
29 textarea::placeholder{
30     width: 100%;
31 }
32
33 .entry, select {
34     background-color: #F7F7F7;
35     padding: 15px;
36     border: 1px solid #E6E6E6;
37     border-radius:15px;
38     margin: 10px 0px 15px 0px;
39 }
40
41 #mainContainer{
42     width: 65%;
43     margin: auto;
44     padding: 20px 40px 60px 40px ;
45     background-color: white;
46     border-radius: 10px;
47 }
```

```
49 .entry input {
50   Line-Height: 25pt;
51   width: 90%;
52 }
53 #catEntry{
54   width: 95%;
55 }
56
57 #nomEntry{
58   width: 100%;
59 }
60
61 #messageEntry{
62   width: 95%;
63 }
64
65 #emailEntry, #phoneEntry, #emailLabel, #phoneLabel, #messageTel, #messageMail{
66   margin-left: 15pt;
67 }
68
69 #champ_email, #champ_phone
70 {
71   display: inline-block;
72   width:49%;
73   padding-left: 0pt;
74   padding-right: 0pt;
75 }
76
77 #champ_phone{
78   float: right;
79 }
80
81 #champ_message{
82   background-color: white;
83   padding: 15px;
84   border: 1px solid #E6E6E6;
85   margin: 10px 0px 15px 0px;
86   border-radius:15px;
87 }
88
89 .submitArea{
90   border: none;
91   padding: 15px 70px 15px 70px;
92   background-color: #333333;
93   color : white;
94   border-radius: 20px;
95   width : 100%;
96
97   transition: background-color 0.4s;
98 }
99
100 .submitArea:hover{
101   background-color: #00AD5F;
102 }
103 }
```

## Annexe 5 : code JavaScript

main.js :

```
1 // Fonction qui masque l'élément
2 function cache(elementStyle){
3     elementStyle.display = "none" ;
4 }
5
6 // Fonction qui affiche l'élément
7 function affiche(elementStyle){
8     elementStyle.display = "";
9 }
10
11 // Fonction de désactivation de l'affichage des messages de correction
12 // ("hiddenMessage" class)
13 function desactivHidden() {
14
15     var hiddenMessage = document.querySelectorAll('.hiddenMessage'),
16         hiddenMessageLength = hiddenMessage.length;
17
18     for (var i = 0; i < hiddenMessageLength; i++) {
19         cache(hiddenMessage[i].style);
20     }
21 }
22
23
24 function editionNom() {
25     var nomInput = document.getElementById('nomEntry');
26     nomInput.className = 'edition';
27 }
28
29 function checkNom() {
30     var nomInput = document.getElementById('nomEntry'), //L'input
31         nomMessageStyle = document.getElementById('messageNom').style; //Le style du message
32
33     if (nomInput.value.length > 2 && /^[A-Za-z]+$/ .test(nomInput.value) ) {
34         nomInput.className = 'OK'; // changement de class
35         cache(nomMessageStyle); // on masque le message de correction
36         return 1;
37     } else {
38         nomInput.className = 'messageHidden';
39         affiche(nomMessageStyle); //on affiche le message de correction
40     }
41     return 0;
42 }
```

```
50 function checkMail() {
51     var mailInput = document.getElementById('emailEntry'), //L'input
52         mailMessageStyle = document.getElementById('messageMail').style; //Le style du message
53
54     if (/^@[a-z0-9._-]+@[a-z0-9._-]+\.[a-z]{2,6}$/.test(mailInput.value)) {
55         mailInput.className = 'OK'; // changement de class
56         cache(mailMessageStyle); // on masque le message de correction
57         return 1;
58     } else {
59         mailInput.className = 'messageHidden';
60         affiche(mailMessageStyle); //on affiche le message de correction
61     }
62     return 0;
63 }
64
65 function editionTel() {
66     var telInput = document.getElementById('phoneEntry');
67     telInput.className = 'edition';
68 }
69
70
71 function checkTel() {
72     var telInput = document.getElementById('phoneEntry'), //L'input
73         telMessageStyle = document.getElementById('messageTel').style; //Le style du message
74
75     if (/^0[1-8]([-. ]?[0-9]{2}){4}$/.test(telInput.value)) {
76         telInput.className = 'OK'; // changement de class
77         cache(telMessageStyle); // on masque le message de correction
78         return 1;
79     } else {
80         telInput.className = 'messageHidden';
81         affiche(telMessageStyle); //on affiche le message de correction
82     }
83     return 0;
84 }
85 }
```

```
86 function checkMessage() {
87     var messageInput = document.getElementById('messageEntry'), //L'input
88     messageMessageStyle = document.getElementById('messageMessage').style, //Le style du message
89     Longueur = messageInput.value.length; //longueur du texte tapé
90
91     if (Longueur >= 20 && Longueur <= 1200) {
92         messageInput.className = ''; // changement de class
93         cache(messageMessageStyle); // on masque le message de correction
94         return 1;
95     } else {
96         messageInput.className = 'messageHidden';
97         affiche(messageMessageStyle); //on affiche le message de correction
98     }
99     return 0;
100 }
101
102 function checkCat() {
103     var catInput = document.getElementById('catEntry'), //L'input
104     messageCatStyle = document.getElementById('messageCat').style; //Le style du message
105
106     if (catInput.value != "0") {
107         cache(messageCatStyle); // on masque le message de correction
108         return 1;
109     } else {
110         catInput.className = 'messageHidden';
111         affiche(messageCatStyle); //on affiche le message de correction
112     }
113     return 0;
114 }
115
116 function Envoi(){
117     var messageFormulaireStyle = document.getElementById('messageFormulaire').style;
118     if(checkCat() == 1 & checkNom() == 1 & checkMail() == 1 & checkTel() == 1 & checkMessage() == 1){
119         cache(messageFormulaireStyle);
120         alert('Formulaire bien rempli');
121         return true;
122     }else{
123         affiche(messageFormulaireStyle);
124         return false;
125     }
126 }
127 desactivHidden();
```

## Annexe 6 : code PHP

- formulaire.php utilisé est exactement le même que formulaire.html précédemment utilisé, hormis son extension.

- envoi.php :

```

1  <html>
2    <head>
3      <!-- En-tête de la page -->
4      <meta charset="utf-8" />
5      <link rel="stylesheet" type="text/css" href="../css/main.css">
6      <link rel="stylesheet" type="text/css" href="../css/police.css">
7      <title> Envoyé ! </title>
8  </head>
9  <body>
10 <?php
11 // S'il y des données de postées
12 if(!empty($_POST['submit'])) {
13
14     if (!empty($_POST['Nom']) && !empty($_POST['Email'])
15       && !empty($_POST['Telephone']) && !empty($_POST['Categorie'])
16       && !empty($_POST['Message']))
17     {
18
19         // Récupération des variables
20         $nom    = htmlentities($_POST['Nom']);
21         $email  = htmlentities($_POST['Email']);
22         $phone  = htmlentities($_POST['Telephone']);
23         $categorie = htmlentities($_POST['Categorie']);
24         $message = htmlentities($_POST['Message']);
25
26         echo "Vos données ont bien été prises en compte.";
27         echo "<br><br>";
28         echo "<center>Résultat du questionnaire :</center>";
29         echo "<br/>";
30         echo "<u>Nom</u> : ". $nom ;
31         echo "<br/>";
32         echo "<u>E-mail</u> : ". $email ;
33         echo "<br/>";
34         echo "<u>Téléphone</u> : ". $phone ;
35         echo "<br/>";
36         echo "<u>Catégorie socioprofessionnelle</u> : n°". $categorie ;
37         echo "<br/>";
38         echo "<br/>";
39         echo "Votre message :";
40         echo "<br/>";
41         echo "<br/>";
42         echo $message;
43     }else{
44         echo "Les données saisies sont incorrectes. Veuillez réessayer.";
45     }
46 }else{
47     echo "Une erreur est survenue. Veuillez réessayer.";
48 }
49 ?>
50 <br><br/>
51 <div align="right">
52   <a href="formulaire.php">Retour</a>
53 </div>
54 </div>
55 </body>
56 </html>
```