

Secteur Tertiaire Informatique

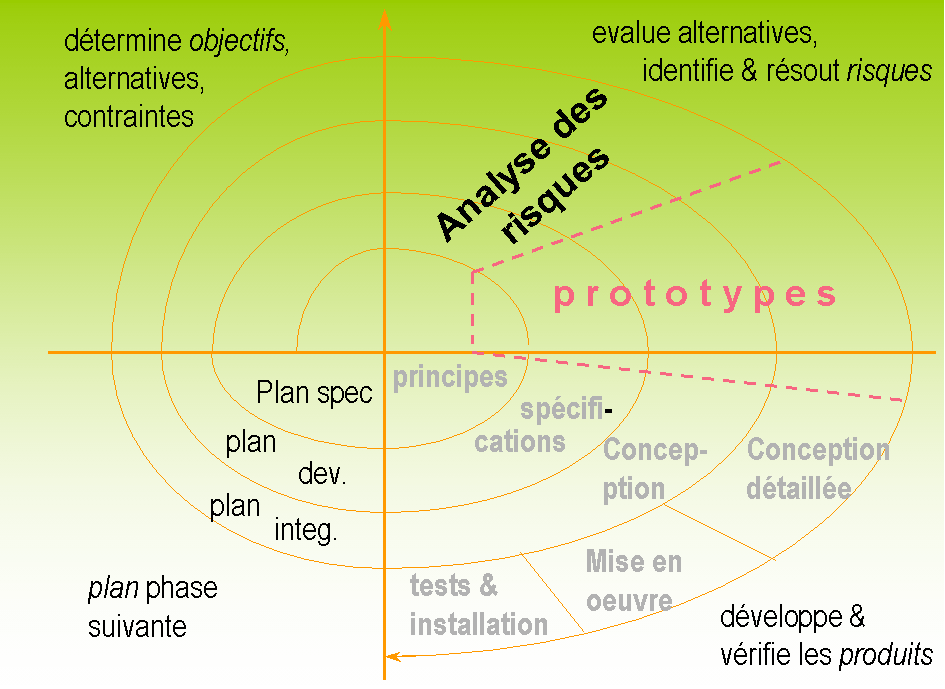
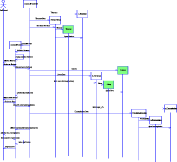
Filière étude - développement

Projet

**Dossier de Conception technique   
IHM Web/ Programmation PHP**

Equipe :

(nom et logo)



Participants :

-

-

-

-

**SOMMAIRE**

[I CONTEXTE 4](#_Toc471405245)

[I.1.1 Reformulation du besoin et User Stories développées 4](#_Toc471405246)

[I.1.2 Plateforme de développement 4](#_Toc471405247)

[I.1.3 La base de données utilisée 4](#_Toc471405248)

[II LA CONCEPTION IHM 5](#_Toc471405249)

[II.1.1 Graphe du dialogue 5](#_Toc471405250)

[III CONCEPTION et développement DES COMPOSANTS 5](#_Toc471405251)

[III.1.1 Les fonctionnalités 5](#_Toc471405252)

[III.1.2 Maquettes 5](#_Toc471405253)

[III.1.3 Les ressources utilisées et particularités de la Story 5](#_Toc471405254)

[III.1.4 Code développé 5](#_Toc471405255)

# CONTEXTE

Reformulation du besoin et User Stories développées

Vous placez ici la reformulation du besoin et les User Stories choisies par votre équipe en début de mission.

Plateforme de développement

Vous citez dans ce tableau les principaux composants logiciels de la plateforme de développement utilisée.

|  |  |  |
| --- | --- | --- |
| **Niveau** | **Produit** | **Remarques** |
| Système d’exploitation | Windows |  |
| SGBD | MySQL version… |  |
| EDI | Visual Studio Code |  |
| Langage(s) | HTML5/CSS3/JavaScript/PHP5 |  |
| Autres | xampp | Inclut Apache version… et PHP version… |

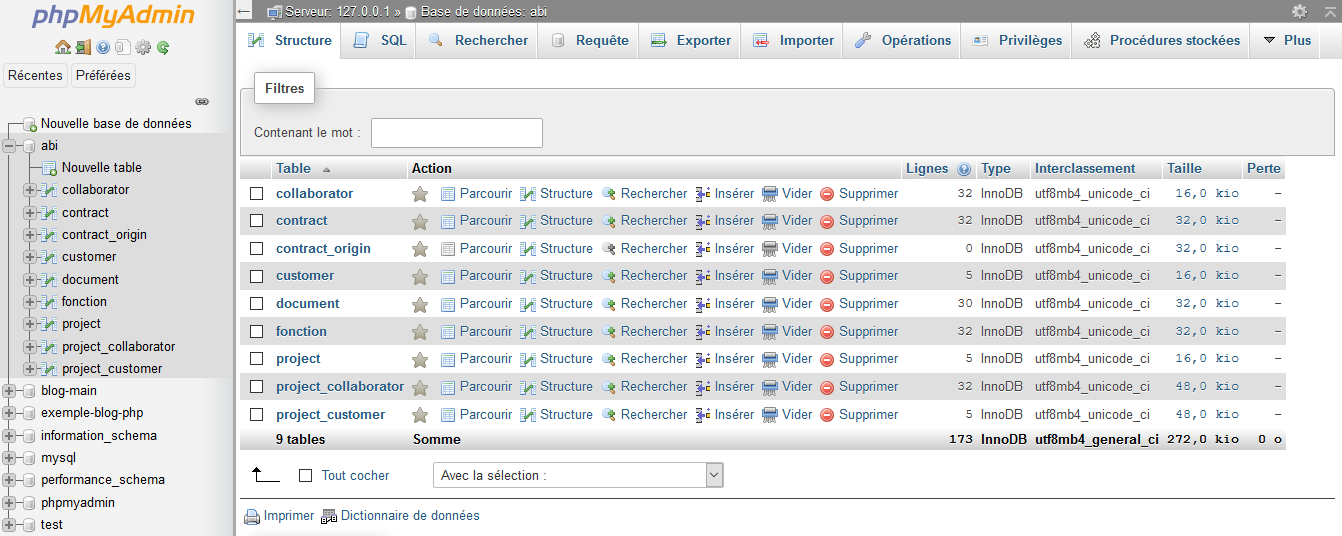
La base de données utilisée

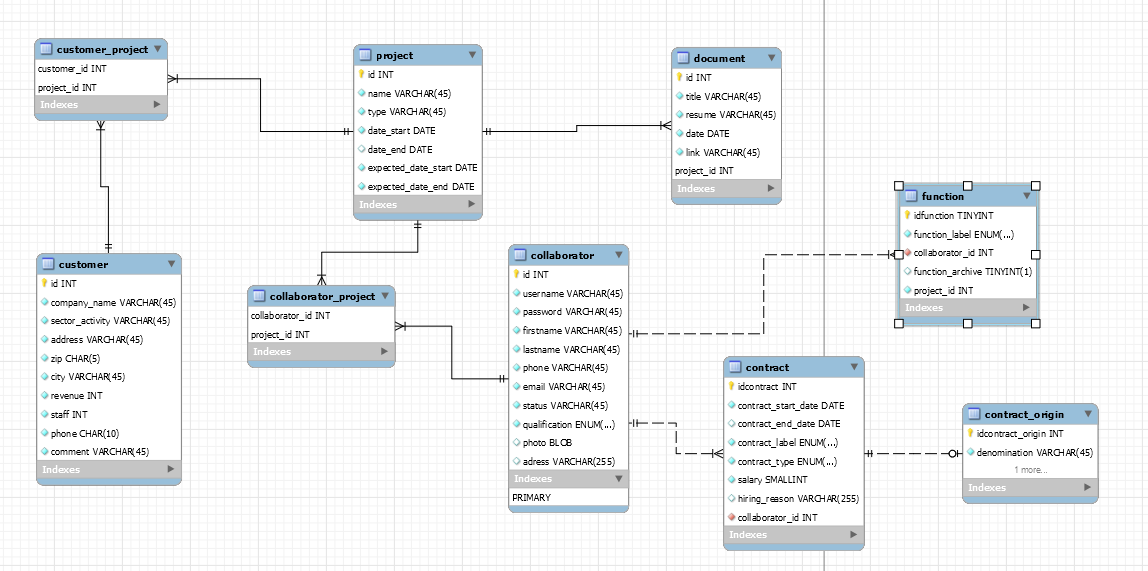
Vous mentionnerez ici le ***nom de la base de données*** utilisée, le ***serveur*** qui l’héberge, les ***modalités de connexion*** à ce serveur...

La base

de donnée est nommé abi

Vous insérerez ici des ***copies d’écrans montrant les tables implémentées*** et votre ***jeu d’essai.***





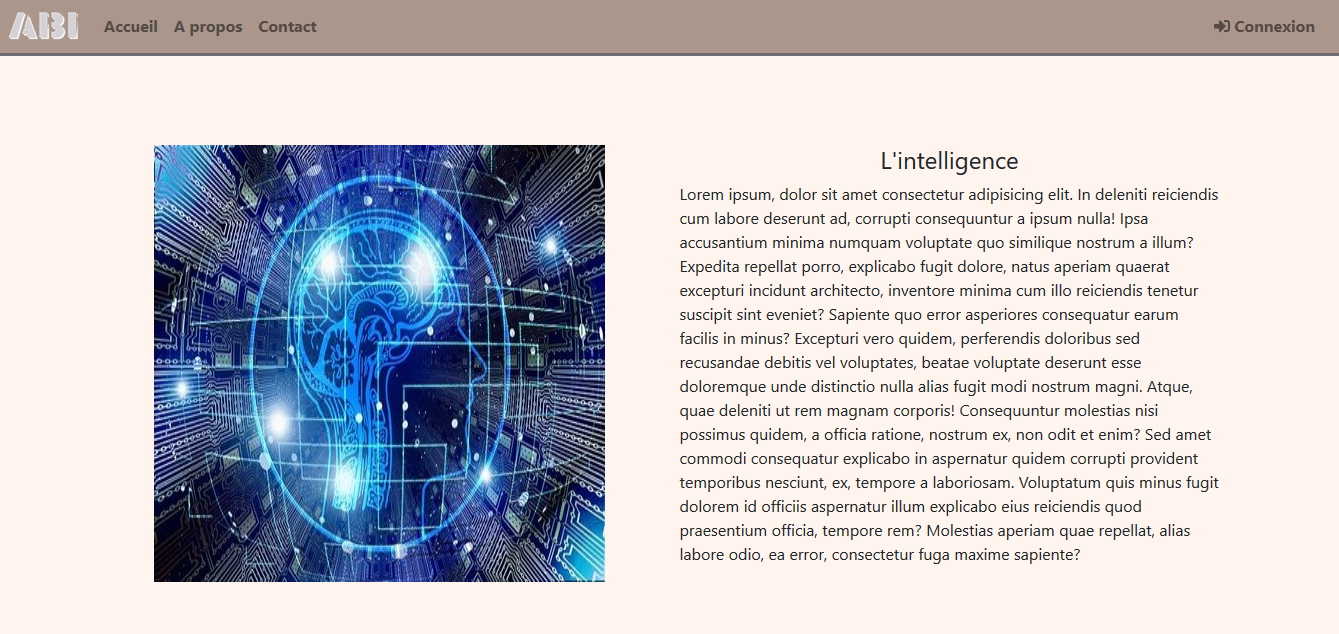
# LA CONCEPTION IHM

Graphe du dialogue

Vous placez ici le ***graphe du dialogue*** de l’application établi à partir des ***images écrans miniatures*** et de ***flèches*** précisant la navigation entre les pages.

Vous pouvez lister les principes éventuels de design spécifiques (couleurs, logos, mise en page, messages d’erreur...) et de navigation (liens hypertexte, retours, droit à l’erreur, messages d’erreur...) que vous avez adoptés.

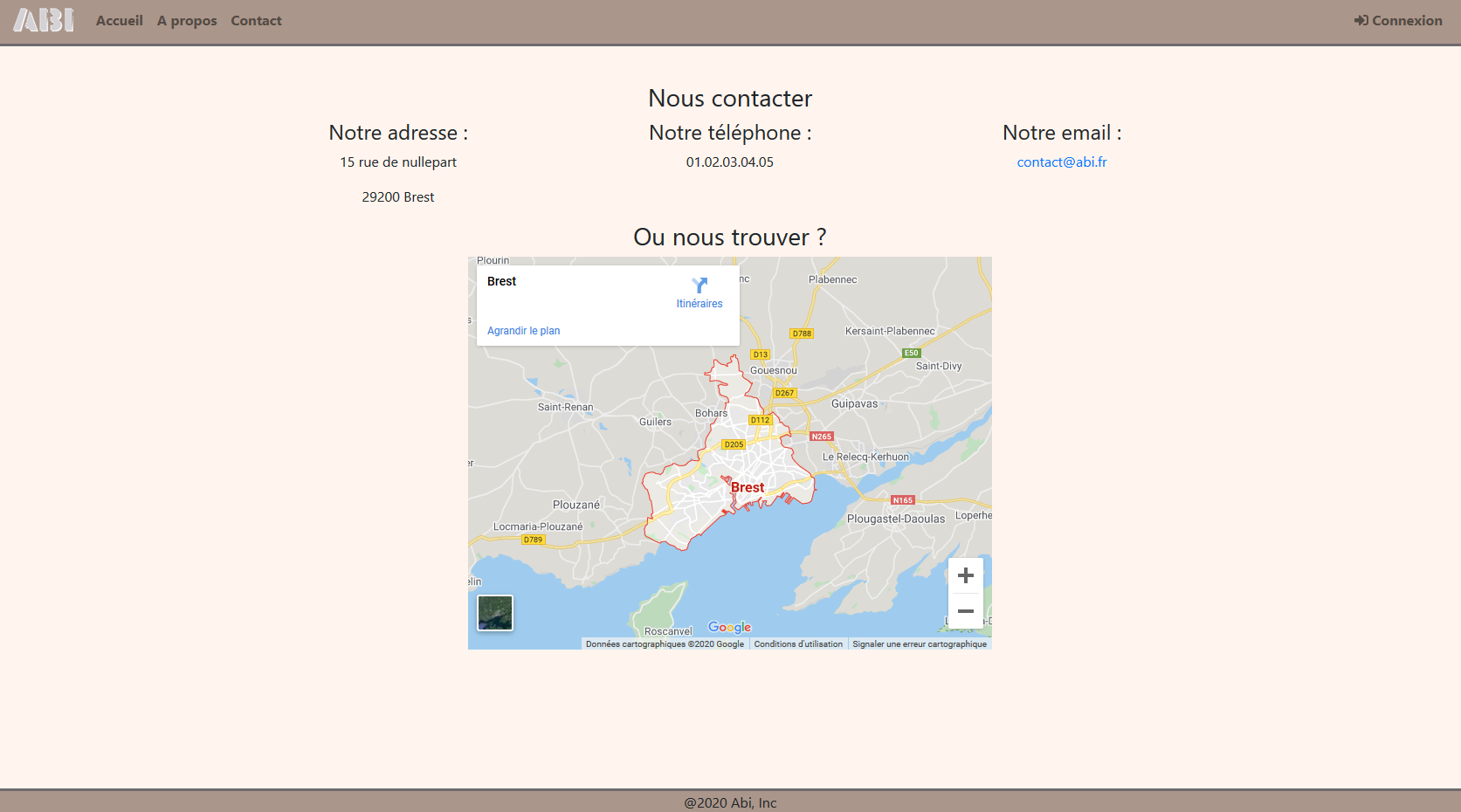
Nous avons créé une page d’accueil très standards permettant : de naviguer vers d’autre pages (A propos et contact) disponible pour chaque visiteur. Ou alors pour les personnels de l’entreprise de se connecter dans une zone membre.



Voici une photo de notre page « A propos » qui est une page donnant des informations générales sur l’entreprise.

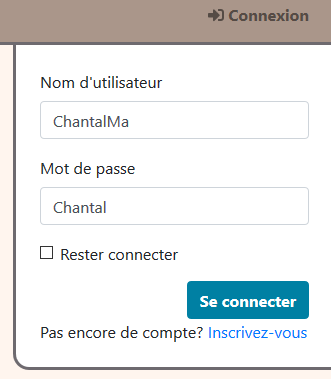


Voici une photo de notre page « Contact » qui est une page donnant des informations sur comment nous contacter et où nous trouver.



Il est possible de se connecter à l’espace membre grâce à vos identifiants :

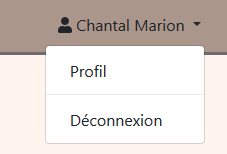
(Ici Chantal Marion) Des profils de connexion ont été créé et permet selon le profil d’avoir accès a des pages ou non.



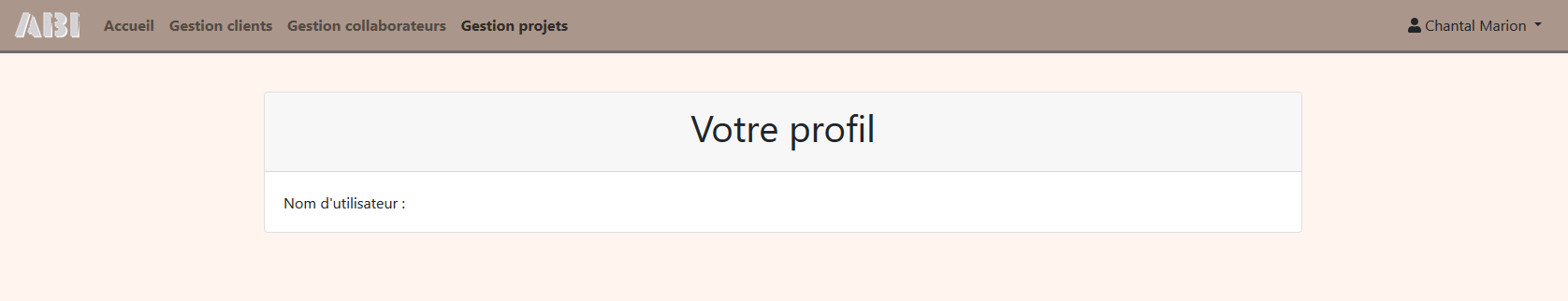
Lorsque nous sommes connectés nous sommes dirigés sur la page d’accueil du site. Mais de nouveaux onglets sont disponible dans la navbar du header, qui sont : Gestion clients, Gestion collaborateurs et Gestion projets. Il est aussi possible de voir votre profil de connexion et se déconnecter.



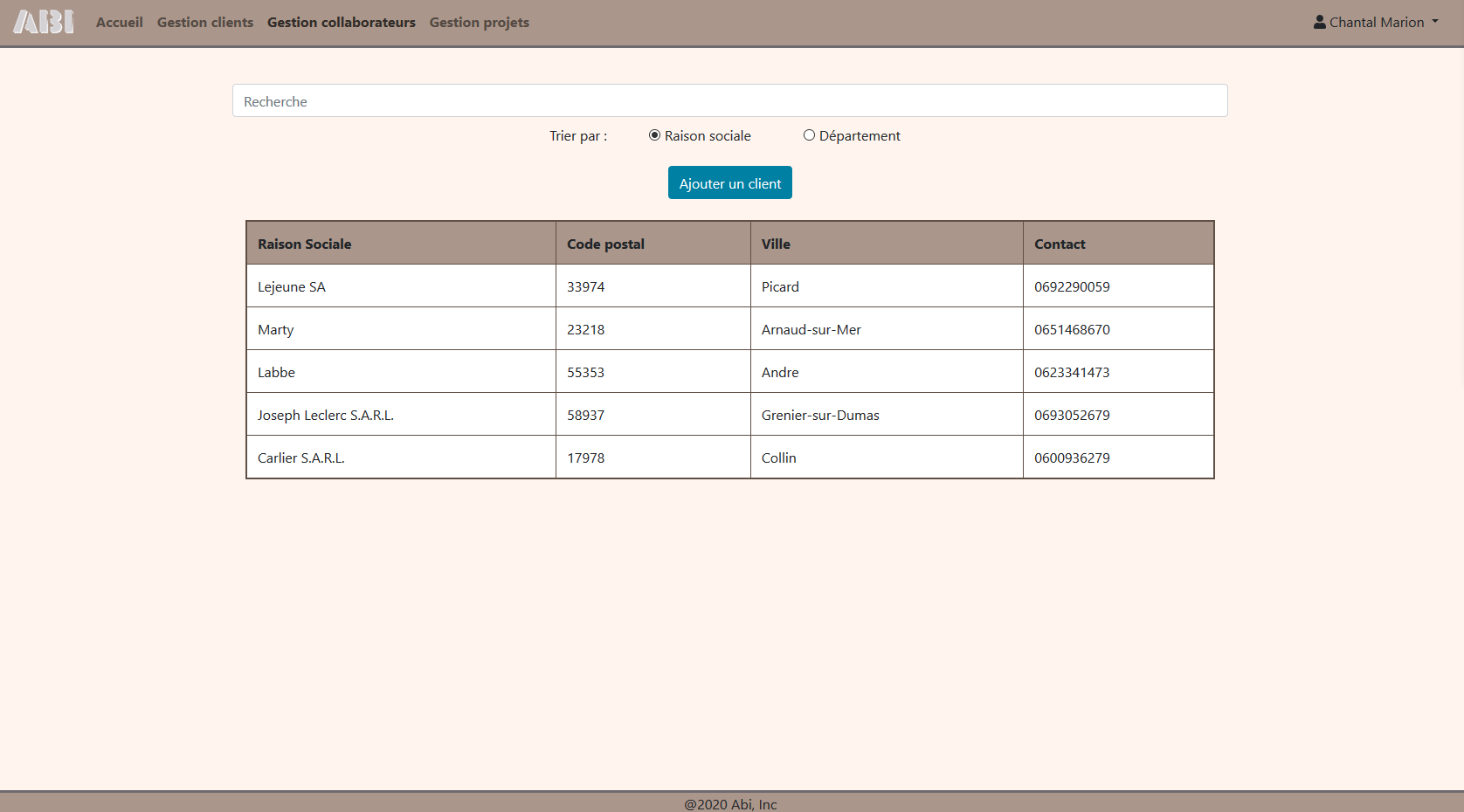
Ici il est possible de se déconnecter ou voir son profil.



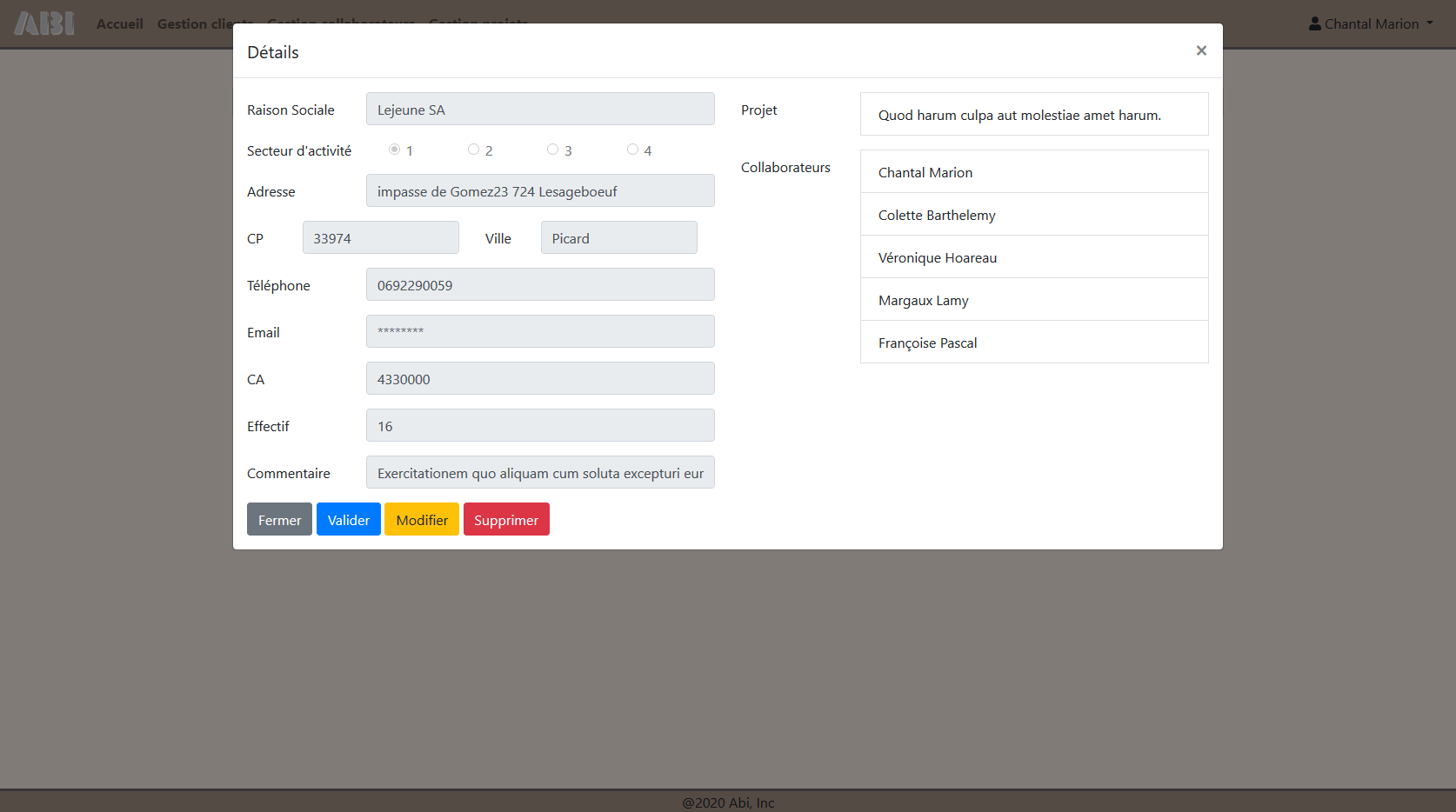
Nous avons choisi de ne pas travailler la page profil pour ce concentré sur d’autre fonctionnalité.



Voici la page « gestion de clients ». Nous pouvons voir les clients de l’entreprise dans un tableau et si nous cliquons dessus un modal s’ouvre, affichant plus d’information avec la possibilité de les modifier.



Permet de créer un modal, pour enregistrer un nouveau client



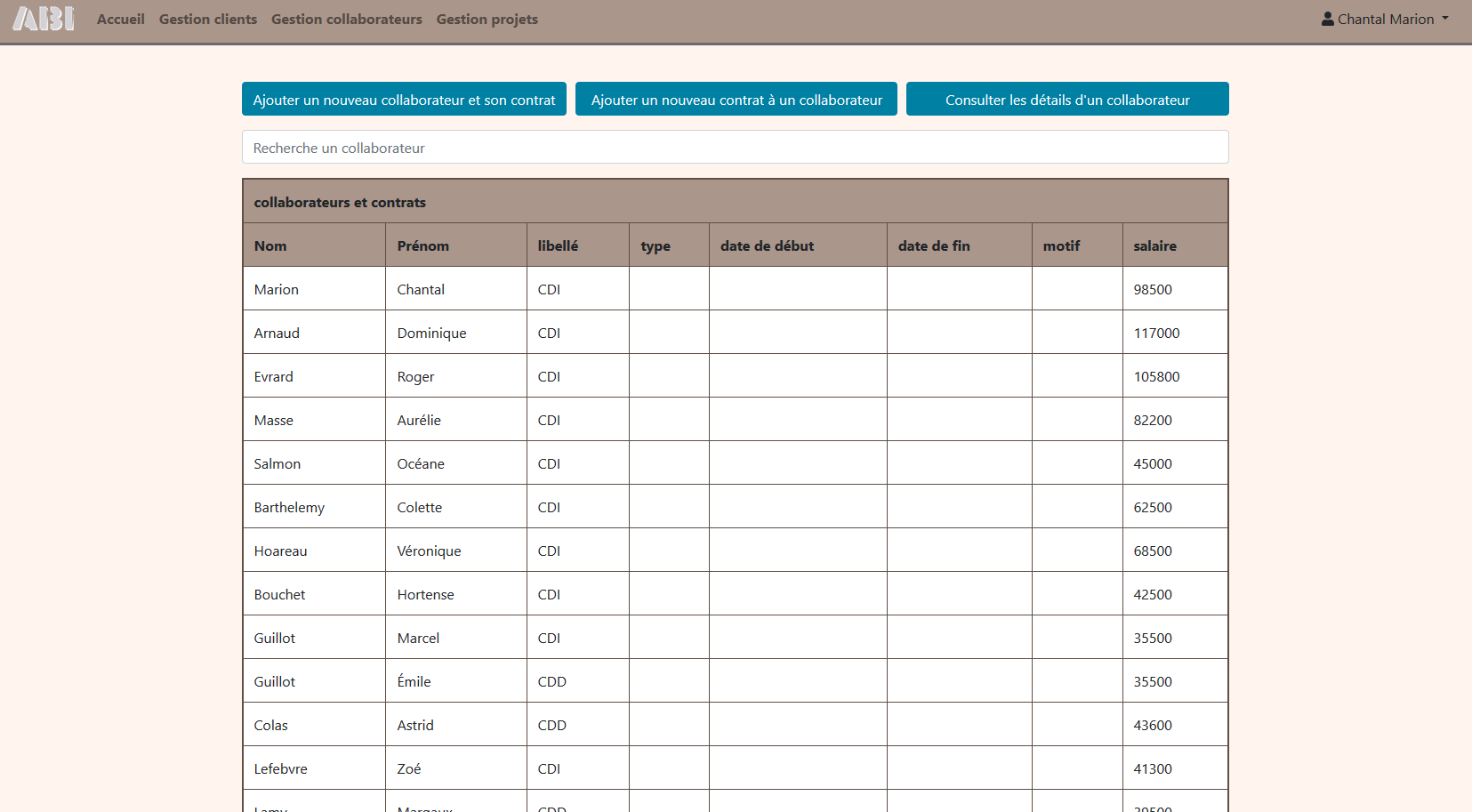
Affichage des collaborateurs

Valider les modifications du client

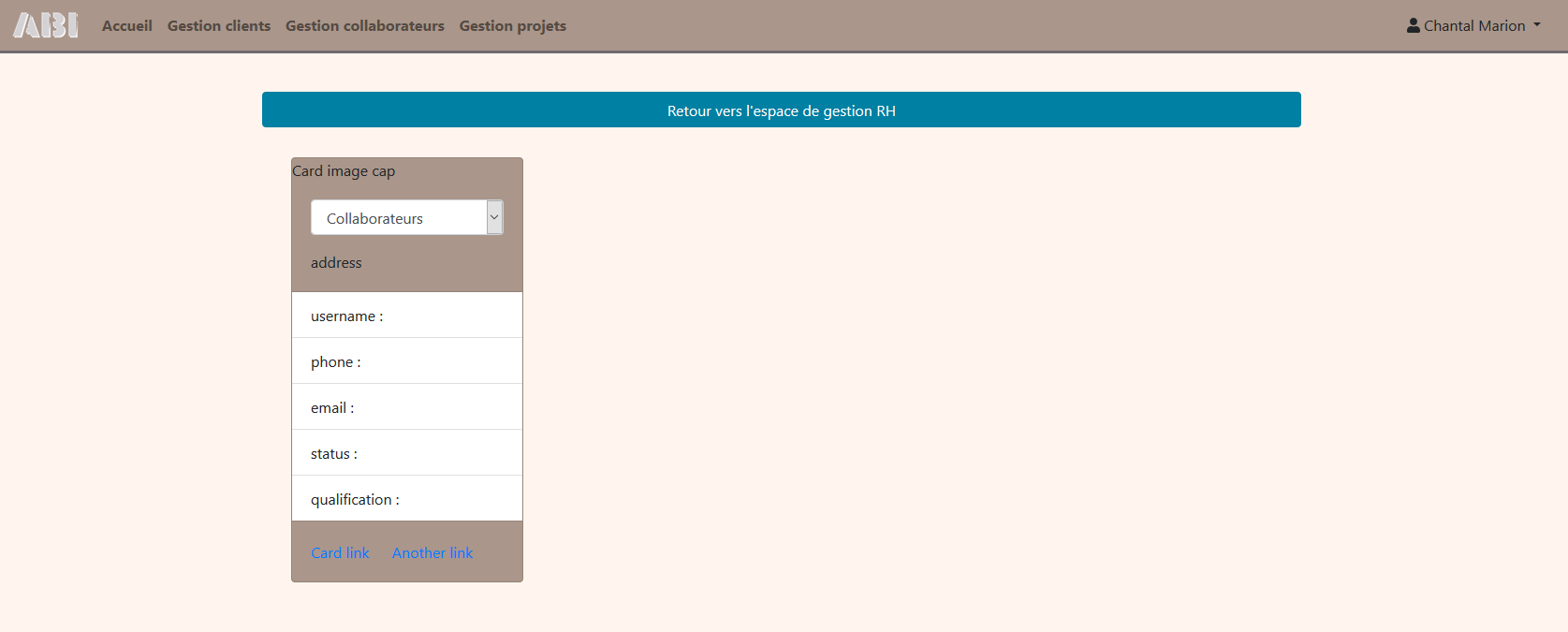
Modifier le client

Supprimer le client

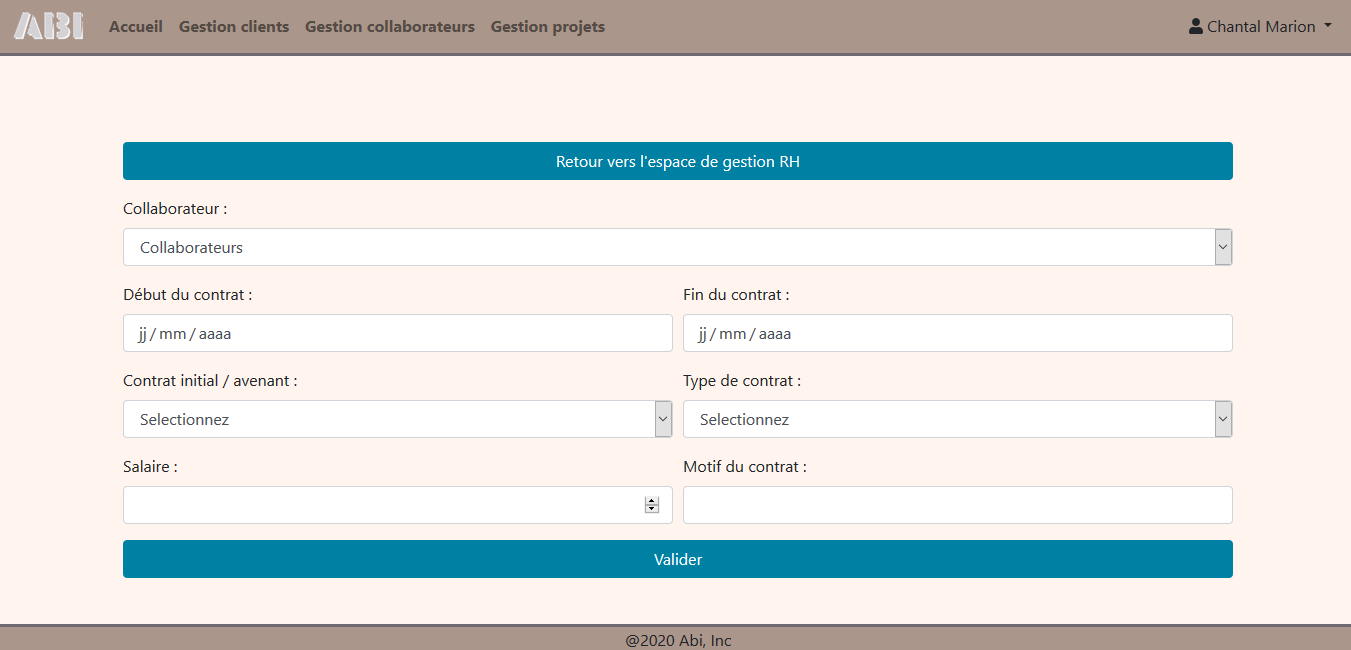
Voici la page « Gestion collaborateurs ». Cette page liste tous les collaborateurs de l’entreprise ainsi que leurs salaires et le type de contrat qu’ils possèdent. Depuis cette page il est possible d’aller vers 3 autres pages permettant de Consulter les détails d’un collaborateur, d’Ajouter un nouveau contrat à un collaborateur et Ajouter un nouveau collaborateur et son contrat.



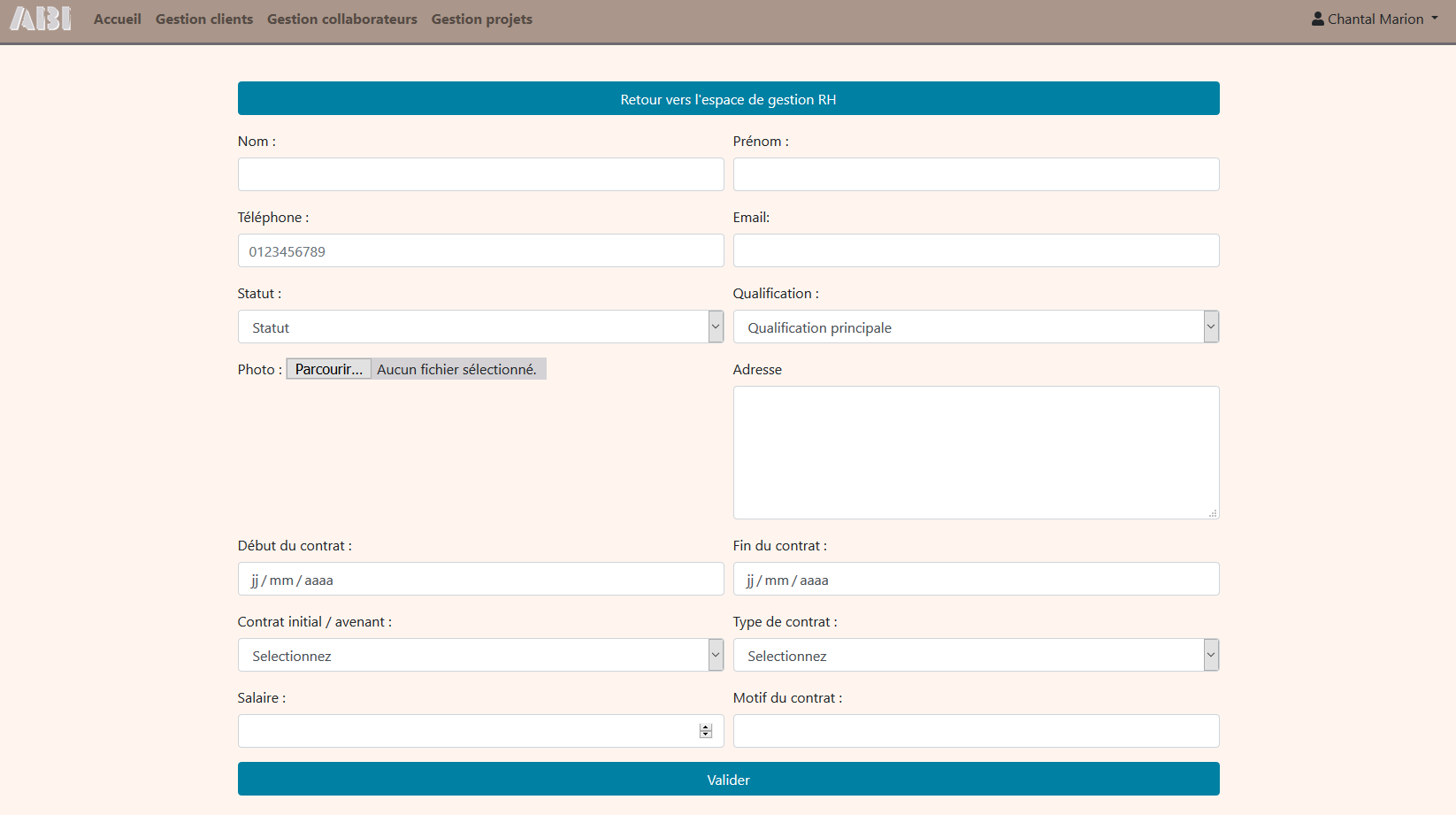
Consulter les détails d’un collaborateur



Ajouter un nouveau contrat à un collaborateur

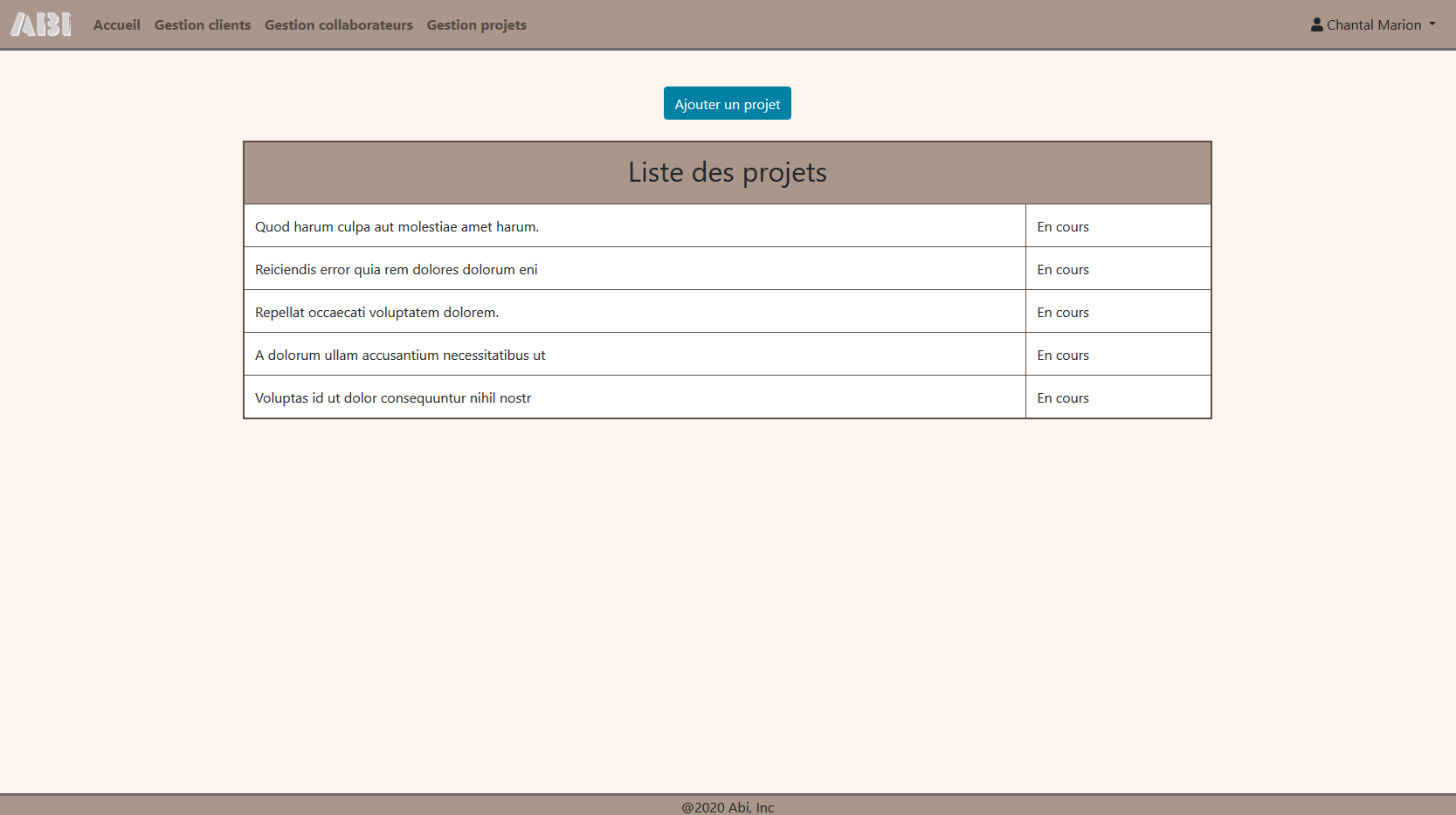


Ajouter un nouveau collaborateur et son contrat



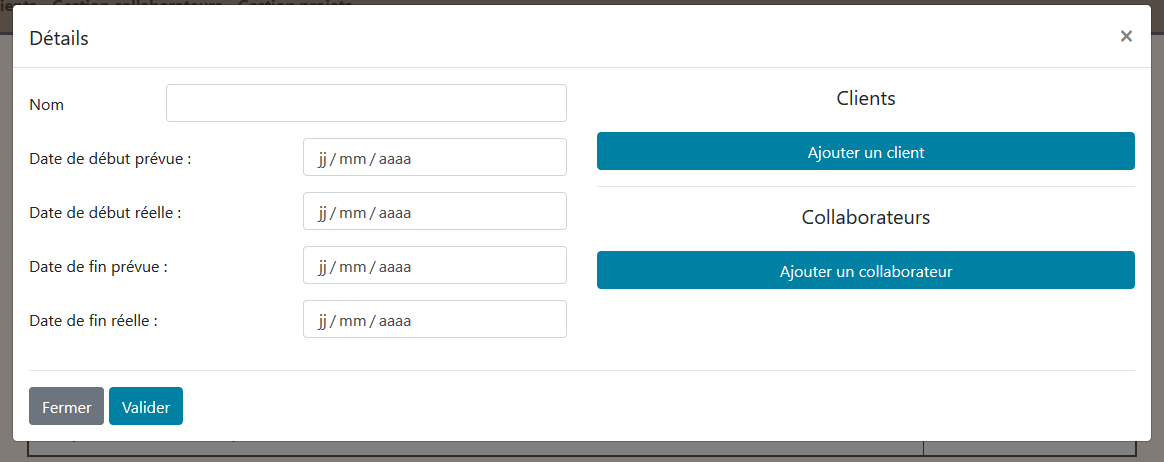
Voici la page « Gestion projets ». Elle permet de créer des projets, afficher les collaborateurs, les clients, le nom du projet ainsi que les dates de début et fin du projet.

En cliquant dans la liste des projets vous aurez plus d’informations et pourrez modifiez certaines informations sur les projets.

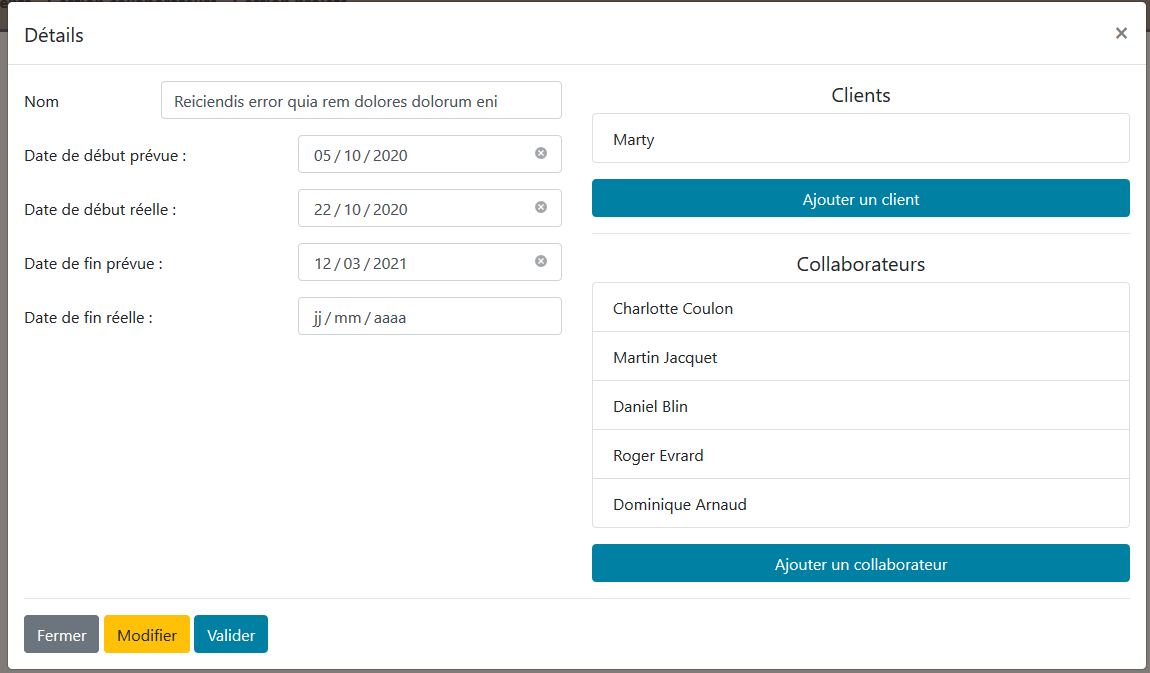


Il est possible de créer un nouveau projet avec une ouverture de modal en cliquant ici

« Ajouter un projet »



« Modifier un projet »



# CONCEPTION et développement DES COMPOSANTS

***Vous détaillez ce chapitre pour chaque User Story développée.***

## Lister tous les clients / Lister les contacts d’un client / Lister les projets d’un client

Les fonctionnalités

Vous énoncez le libellé de la User Story*.*

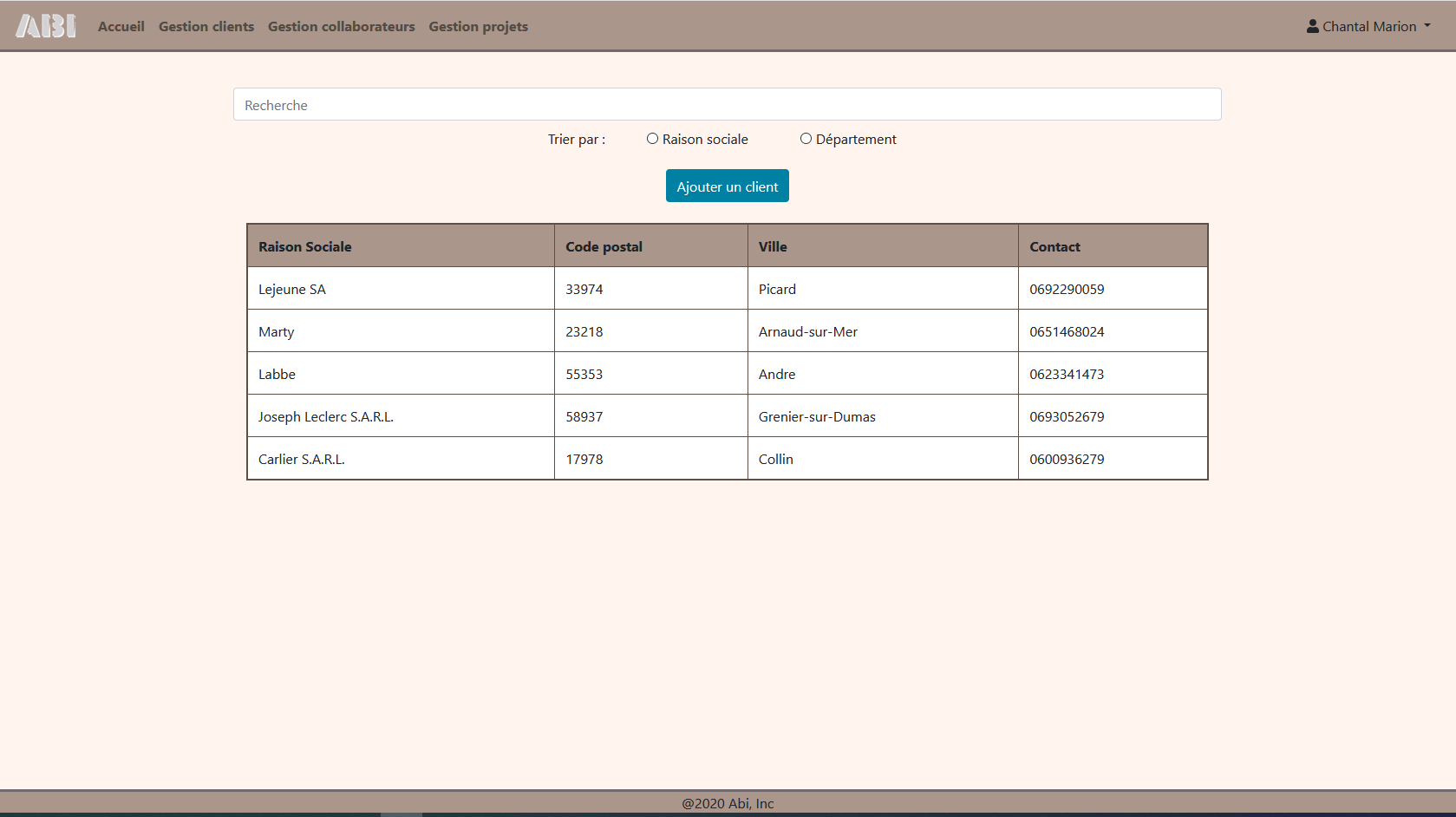
Vous précisez l’état de la User Story au moment de la livraison (terminée, tâches en cours).

**Volet GS : Lister tous les clients (User story terminée), Lister les contacts d’un client (User story terminée), Lister les projets d’un client (User story terminée)**

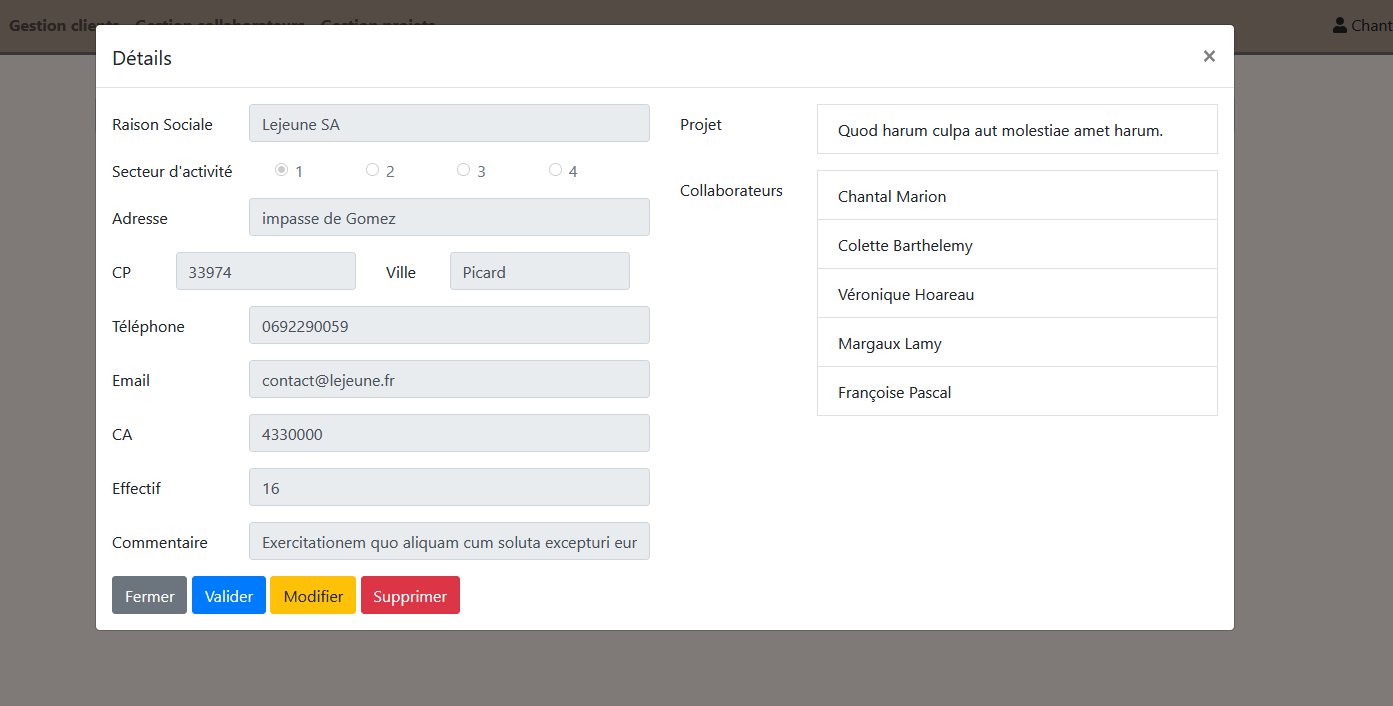
Maquettes

Vous insérez ici une copie de chaque écran développé pour cette User Story.

Page Gestion des Clients avec tableau contenant les clients de la base de donnée.



Nous avons choisi d’afficher sur ce tableau le nom du client, le code postal, la ville et le numéro de téléphone. Au clic sur une ligne, les détails du client apparaissent avec ses projets associés ainsi que les collaborateurs.



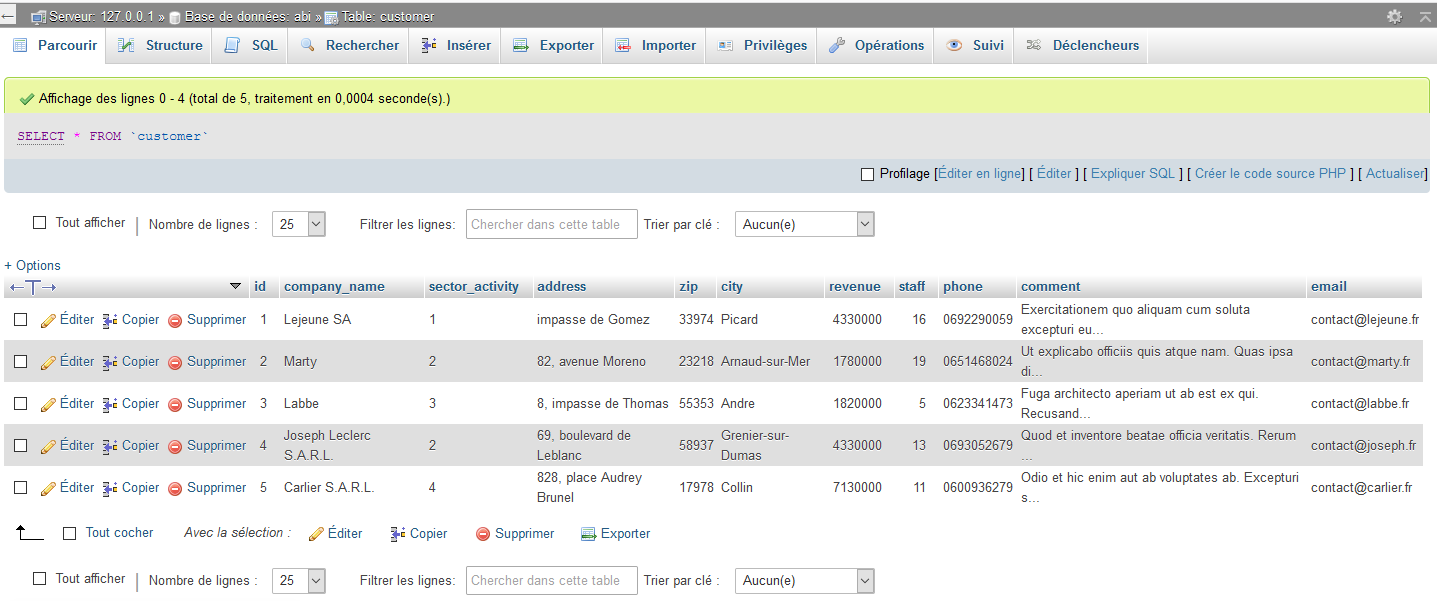
Les ressources utilisées et particularités de la Story

Le développement des fonctionnalités utilise des ressources diverses extérieures : tables de base de données, classes Métier ou d’accès à la BDD, bibliothèques de code... Le lien avec ces ressources doit pouvoir être retrouvé aisément par le développeur en situation de maintenance.

Si le travail de développement est bien fait, chaque écran a fait l’objet d’un maquettage validé puis d’une ***programmation modulaire dûment commentée, en particulier par le ‘cartouche’ indiquant en début de script l’auteur et les relations de ce script avec les autres ressources***.

Vous pouvez préciser ou rappeler ici certains **algorithmes** ou **règles de gestion** particulièrement importants ou spécifiques.

Image de la table Customer de la base donnée contenant la liste des clients, ainsi que toutes les informations les concernant.



La classe de connexion à la base de donnée utilisé est la même pour chaque User Story développée. C’est la class Dbconfig qui contient le DSN, le Username et le Password de connexion.

<?php

namespace App\Config;

/\*\*

 \* Class de configuration pour l'accès à la base de donnée

 \* @author Aymeric

 \* @author Doryan

 \* @author Simon

 \* 19/10/20

 \*/

class DbConfig

{

    const DSN = "mysql:host=localhost;dbname=abi";

    const USERNAME = "root";

    const PASSWORD = "";

}

La class Database renseigne ensuite les informations nécessaires à la connexion dans le PDO avec la méthode connect qui retourne le PDO rempli.

<?php

/\*\*

 \* @author Aymeric

 \*/

namespace App;

use \PDO;

use App\Config\DbConfig;

class Database

{

    /\*\*

     \* Retourne une instance de PDO

     \* @return PDO

     \*/

    public static function connect(): PDO

    {

        return new PDO(DbConfig::DSN, DbConfig::USERNAME, DbConfig::PASSWORD,[

            PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION

        ]);

    }

}

La class Repository est utilisé pour ses méthodes find et findAll qui permet de récupérer les clients de la table customer.

<?php

/\*\*

 \* @author Doryan

 \* @author Yann

 \* @author Simon

 \* @author Aymeric

 \* date 19/10/20

 \*/

namespace App\Repository;

use App\Config\DbConfig;

use App\Database;

use \PDO;

class Repository

{

    protected PDO $pdo;

    private string $table;

    private string $classname;

    public function \_\_construct(string $classname)

    {

        $this->pdo = Database::connect();

        $this->classname = $classname;

        $this->table = strtolower($classname);

    }

    /\*\*

     \* @method find

     \* Retourne les valeurs correspondantes dans la table que l'on recherche dans la base de donnée selon l'id

     \* @param int id

     \* @return mixed

     \*/

    public function find(int $id)

    {

        $sql = "SELECT \* FROM $this->table WHERE id =?";

        $query = $this->pdo->prepare($sql);

        $query->execute([$id]);

        $query->setFetchMode(PDO::FETCH\_CLASS, "App\Entity\\" . $this->classname);

        return $query->fetch();

    }

    /\*\*

     \* @method findAll

     \* Retourne les valeurs correspondantes dans la table que l'on recherche

     \* @return array

     \*/

    public function findAll(): array

    {

        $sql = "SELECT \* FROM $this->table";

        $query = $this->pdo->prepare($sql);

        $query->execute();

        return $query->fetchAll(PDO::FETCH\_CLASS, "App\Entity\\" . $this->classname);

    }

}

On l’utilise grâce à la class CustomerRepository qui est un enfant de cette classe.

<?php

/\*\*

 \* @author Doryan

 \* @author Aymeric

 \* @author Simon

 \* 21/10/20

 \*/

namespace App\Repository;

use App\Repository\Repository;

use App\Entity\Project;

use App\Entity\Customer;

use \PDO;

class CustomerRepository extends Repository

{

    public function \_\_construct()

    {

        parent::\_\_construct("Customer");

    }

    /\*\*

     \* @method findByProject

     \* Retourne tous les Customer qui ont une relation avec le Project passé en paramètre

     \* @param Project $project

     \* @param int $nb

     \* @return array

     \*/

    public function findByProject( Project $project, $nb = 1): array

    {

        $idProject = $project->getId();

        $query = $this->pdo->prepare("SELECT customer.\* FROM customer JOIN project\_customer AS pc ON customer.id = pc.customer\_id WHERE pc.project\_id = ?");

        $query->execute([$idProject]);

        return $query->fetchAll(PDO::FETCH\_CLASS, Customer::class,[$nb]);

    }

}

Code développé

Vous insérez ***ici tout le*** ***code HTML/CSS/JavaScript/PHP -commenté- développé pour chacun des composants***.

Pour des scripts communs à plusieurs User-Stories (feuille CSS, script d’accès à la base…), copiez une seule fois le code correspondant dans votre dossier.

La méthode findAll s’utilise dans le fichier Controller. On créer la variable customers qui contient la liste des clients grâce à la méthode appliquée au CustomerRepository, qui est une extension de la class Repository.

public function gestionclientsController()

    {

        ob\_start();

        session\_start();

        $customerRepository = new CustomerRepository();

        $customers = $customerRepository->findAll();

        $accepted = ["Scrum master", "Secrétaire technique", "Directeur général", "Directeur administratif", "Directeur financier", "Responsable des ressources humaines", "Secrétaire administratif", "Commercial"];

        if (in\_array($\_SESSION['fonction'], $accepted)) {

            include '../templates/gestionclients.php';

        } else {

            include '../templates/accessdenied.php';

        }

        ob\_end\_flush();

    }

L’affichage du tableau se fait directement à l’arrivée sur la page. Il est rempli automatiquement grâce aux balises php utilisé. Chaque ligne est cliquable.

<table class="table table-hover table-bordered" id="tableCustomer">

  <thead>

    <tr>

      <th scope="col">Raison Sociale</th>

      <th scope="col">Code postal</th>

      <th scope="col">Ville</th>

      <th scope="col">Contact</th>

    </tr>

  </thead>

  <tbody>

    <?php foreach ($customers as $customer) : ?>

      <tr id="<?= $customer->getId() ?>" class="row\_customer">

        <td scope="row" class="company\_name"><?= $customer->getCompanyName() ?></th>

        <td class="zip"><?= $customer->getZip() ?></td>

        <td class="city"><?= $customer->getCity() ?></td>

        <td class="phone"><?= $customer->getPhone() ?></td>

      </tr>

    <?php endforeach; ?>

  </tbody>

</table>

Le css utilisé est celui fourni par Bootstrap avec quelque modification sur les couleurs et bordures.

thead {

    background-color: #AA968A;

}

.table-bordered {

    border: 2px solid #5F4F45;

}

.table-bordered>thead>tr>th {

    border: 1px solid #5F4F45;

}

.table-bordered>tbody>tr>td {

    border: 1px solid #5F4F45;

}

.table {

    background-color: white;

}

Le modal de détail du client est appelé aussi par le Controller.

public function modalController(string $classname)

    {

        $id = $\_POST["id"];

        $repositoryName = 'App\Repository\\' . $classname . 'Repository';

        $repository = new $repositoryName();

        $entity = $repository->find($id);

        echo $entity->toJSON();

    }

Il est rempli lors du clic sur une ligne en JavaScript.

/\*\*

 \* @author Doryan

 \* @author Aymeric

 \* @author Simon

 \* 22/10/20

 \* Permet d'ouvrir un modal client avec toutes les info de la BDD et du php

 \*/

$(".row\_customer").click(function () {

  $("input").attr("disabled", true);

  $("#modalCustomer input").removeClass(["is-valid", "is-invalid"]);

  $.post("/modalCustomer", { id: this.id }, function (data) {

    let customer = JSON.parse(data);

    for (property in customer) {

      $("#modalCustomer #" + property).val(customer[property]);

    }

    $("#modalCustomer input[name=sector\_activity][value=" + customer.sector\_activity + "]").prop("checked", true);

    $("#projects").empty();

    $("#collaborators").empty();

    $(customer.projects).each(function (key, project) {

      $("#projects").append('<li class="list-group-item d-flex justify-content-between align-items-center">' + project.name + "</li>");

      $(project.collaborators).each(function (key, collaborator) {

        $("#collaborators").append('<li class="list-group-item d-flex justify-content-between align-item-center">' + collaborator.firstname + " " + collaborator.lastname + "</li>");

      });

    });

  });

  $("#modalCustomer").modal("toggle");

});

## Rechercher un client

Les fonctionnalités

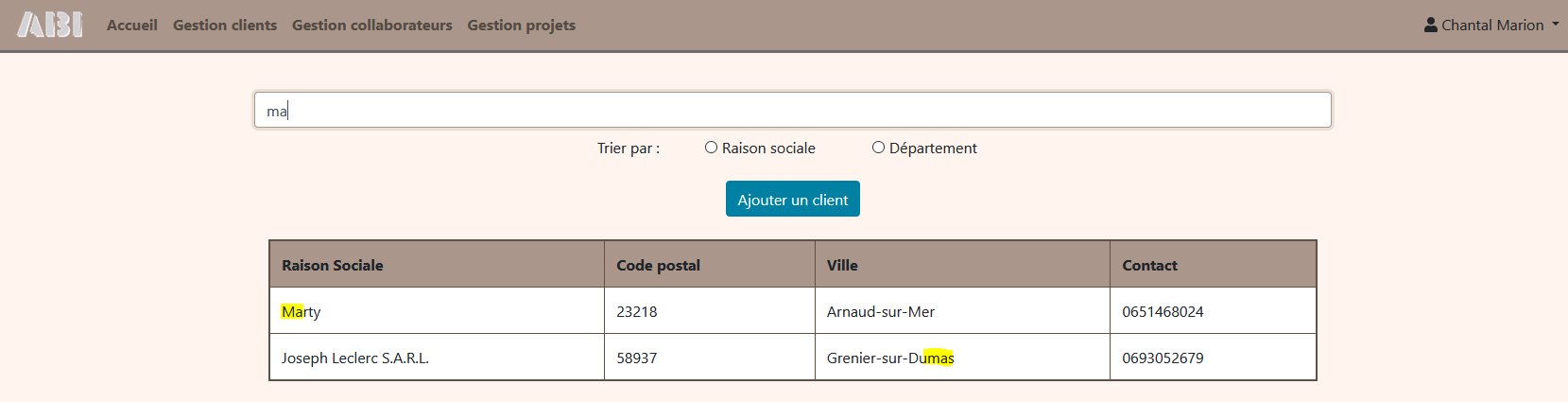
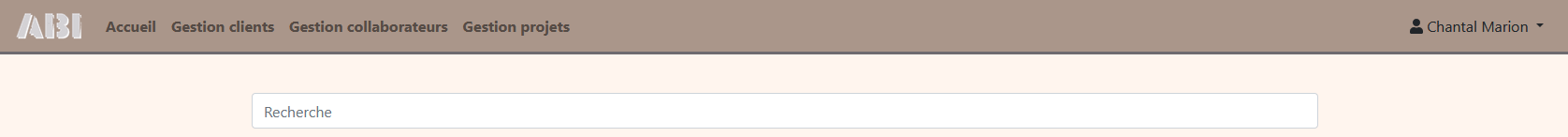
Vous énoncez le libellé de la User Story*.*

Vous précisez l’état de la User Story au moment de la livraison (terminée, tâches en cours).

**Volet GS : Rechercher un client (User story en cours)**

Maquettes

Vous insérez ici une copie de chaque écran développé pour cette User Story.



Les ressources utilisées et particularités de la Story

Le développement des fonctionnalités utilise des ressources diverses extérieures : tables de base de données, classes Métier ou d’accès à la BDD, bibliothèques de code... Le lien avec ces ressources doit pouvoir être retrouvé aisément par le développeur en situation de maintenance.

Si le travail de développement est bien fait, chaque écran a fait l’objet d’un maquettage validé puis d’une ***programmation modulaire dûment commentée, en particulier par le ‘cartouche’ indiquant en début de script l’auteur et les relations de ce script avec les autres ressources***.

Vous pouvez préciser ou rappeler ici certains **algorithmes** ou **règles de gestion** particulièrement importants ou spécifiques.

Pour cette User Story, aucun accès a la base de donnée n’est nécessaire, la recherche s’effectue sur le tableau directement.

Code développé

Vous insérez ***ici tout le*** ***code HTML/CSS/JavaScript/PHP -commenté- développé pour chacun des composants***.

Pour des scripts communs à plusieurs User-Stories (feuille CSS, script d’accès à la base…), copiez une seule fois le code correspondant dans votre dossier.

On utilise le JavaScript afin de rechercher dans notre tableau de client les correspondances avec ce que l’on écrit.

/\*\*

 \* Fonction Recherche

 \* @author Simon

 \* 21/10/20

 \*/

$(document).ready(function () {

  $("#tableSearch").on("keyup", function () {

    var value = $(this).val().toLowerCase();

    $("#tableCustomer tbody tr").filter(function () {

      $(this).toggle($(this).text().toLowerCase().indexOf(value) > -1);

    });

  });

});

La barre de recherche en HTML est un input avec un id que l’on récupère en Javascript.

<div class="row">

  <input class="form-control mb-2" id="tableSearch" type="text" placeholder="Recherche">

</div>

Aucun contrôle de saisie n’a été mis en place pour cette fonction.

## Mettre a jour les données générales d’un client / Supprimer un client / Ajouter un nouveau client et ses contacts déjà connus

Les fonctionnalités

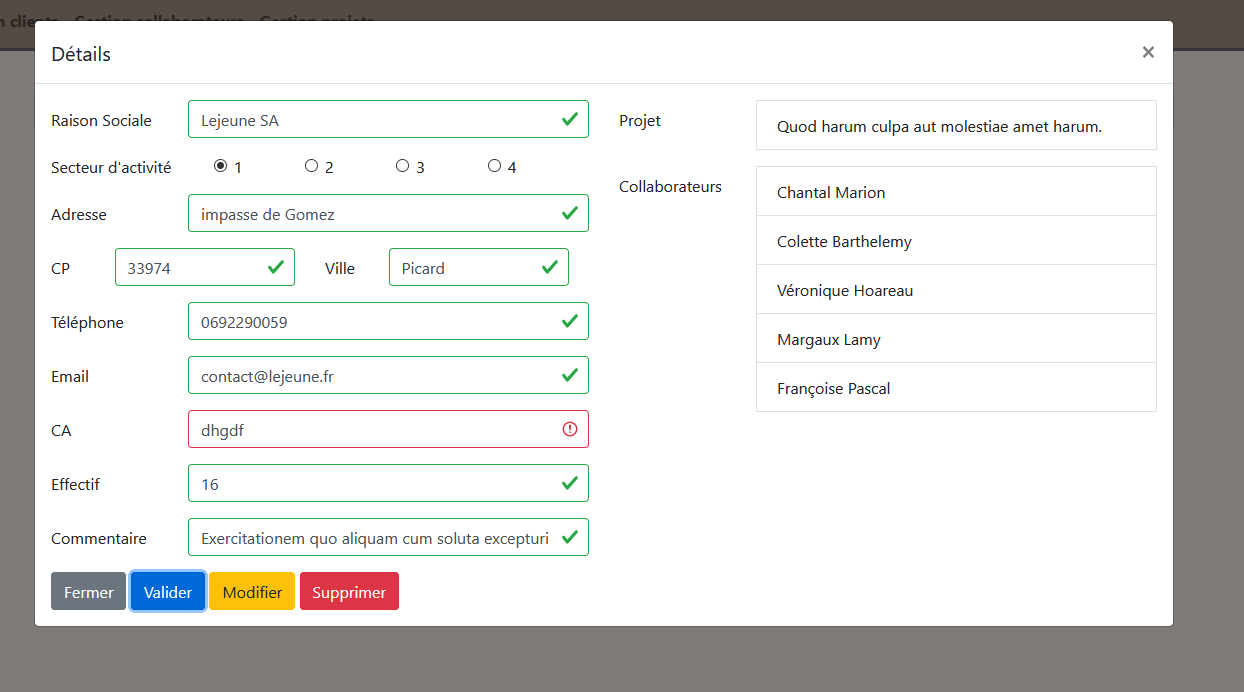
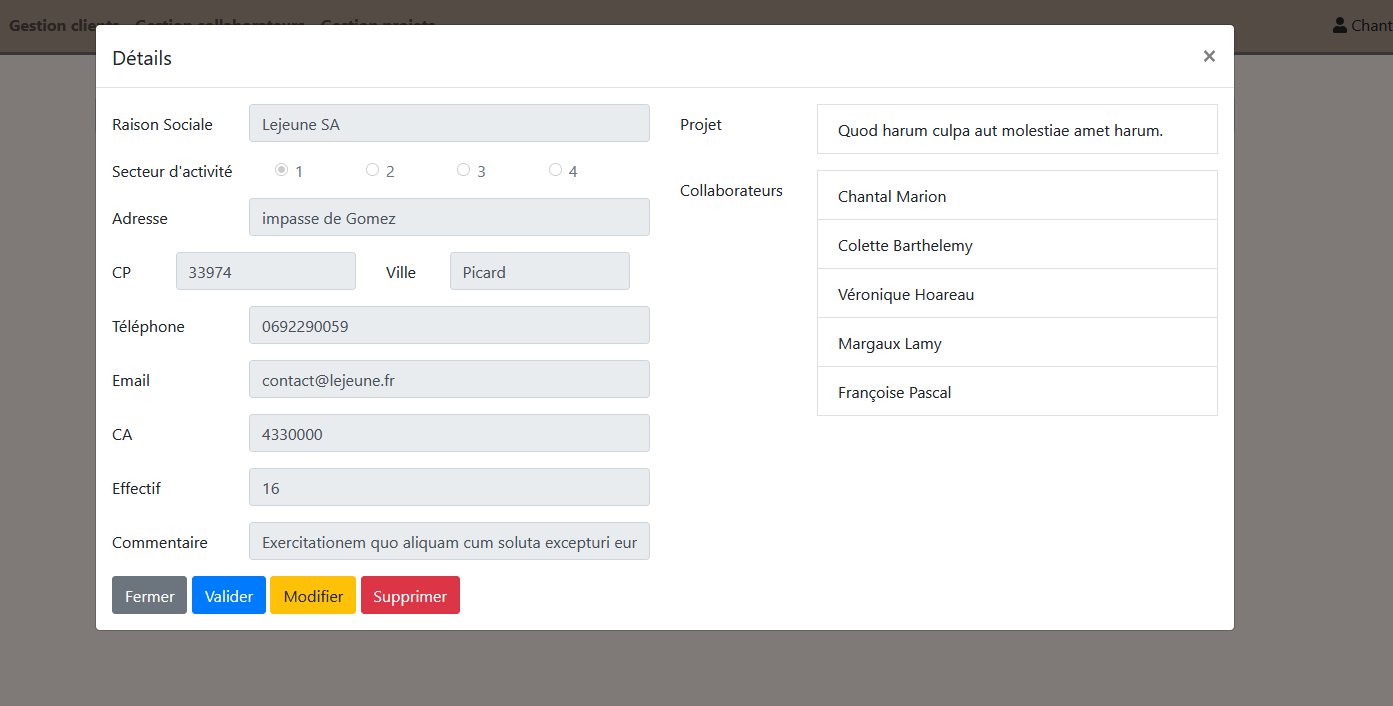
Vous énoncez le libellé de la User Story*.*

Vous précisez l’état de la User Story au moment de la livraison (terminée, tâches en cours).

**Volet GS : Mettre à jour les données générales d’un client (User story terminée), Supprimer un client (User story terminée), Ajouter un nouveau client et ses contacts déjà connus (User story en cours),**

Maquettes

Vous insérez ici une copie de chaque écran développé pour cette User Story.



Les ressources utilisées et particularités de la Story

Le développement des fonctionnalités utilise des ressources diverses extérieures : tables de base de données, classes Métier ou d’accès à la BDD, bibliothèques de code... Le lien avec ces ressources doit pouvoir être retrouvé aisément par le développeur en situation de maintenance.

Si le travail de développement est bien fait, chaque écran a fait l’objet d’un maquettage validé puis d’une ***programmation modulaire dûment commentée, en particulier par le ‘cartouche’ indiquant en début de script l’auteur et les relations de ce script avec les autres ressources***.

Vous pouvez préciser ou rappeler ici certains **algorithmes** ou **règles de gestion** particulièrement importants ou spécifiques.

Afin d’effectuer une mise à jour d’un client, nous utiliserons la class Form et la class CustomerForm qui est un des enfants. La class Form nous permet d’utilisé la méthode getFailed qui insère dans un tableau tous les champs du formulaire qui ne sont pas valide.

<?php

namespace App\Forms;

abstract class Form

{

    /\*\*

     \* @author Aymeric

     \* @author Simon

     \* 28/10/20

     \* Retourne un tableau contenant le nom de tous les champs d'un formulaire qui ne sont pas valide

     \* @return array

     \*/

    public function getFailed(): array

    {

        $objectVars = get\_object\_vars($this);

        unset($objectVars['id']);

        $failed = [];

        foreach($objectVars as $key => $value){

            $methodName = $key . 'IsValid';

            if(!$this->$methodName()){

                $failed[]= $key;

            }

        }

        return $failed;

    }

}

La class CustomerForm contient toutes les méthodes de contrôles des valeurs modifiées dans notre formulaire ainsi que les méthodes add et update.

<?php

namespace App\Forms;

use \PDO;

use App\Config\DbConfig;

use App\Entity\Customer;

use App\Repository\Repository;

class CustomerForm extends Form

{

    protected ?int $id;

    protected $company\_name;

    protected $sector\_activity;

    protected $address;

    protected $zip;

    protected $city;

    protected $revenue;

    protected $staff;

    protected $phone;

    protected $email;

    protected $comment;

    public function \_\_construct(array $post)

    {

        parent::\_\_construct($post);

    }

    /\*\*

     \* Regex controle du formulaire company\_name

     \* @author Simon

     \* 29/10/20

     \*/

    protected function company\_nameIsValid()

    {

        return preg\_match("/^[A-Za-zÀ-ÿ0-9 .-]{1,}$/", $this->company\_name);

    }

    /\*\*

     \* Controle du formulaire sector\_activity

     \* @author Simon

     \* 29/10/20

     \*/

    protected function sector\_activityIsValid()

    {

        return in\_array($this->sector\_activity, ["1", "2", "3", "4"]);

    }

!!!! CHAQUE VALEUR A SA METHODE DE CONTROLE !!!!

    /\*\*

     \* @author Doryan

     \* 23/10/20

     \* Fonction permetant d'ajouter un Client a la BDD

     \*/

    public function add()

    {

        $pdo = new PDO(DbConfig::DSN, DbConfig::USERNAME, DbConfig::PASSWORD);

        $objectVars = get\_object\_vars($this);

        unset($objectVars['id']);

        $param = array\_values($objectVars);

        $query = $pdo->prepare("INSERT INTO customer (company\_name, sector\_activity, address, zip, city, revenue, staff, phone, email, comment) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)");

        $query->execute($param);

    }

    /\*\*

     \* @author Doryan

     \* 23/10/20

     \* Fonction permetant de modifier un Client de la BDD

     \*/

    public function update()

    {

        $pdo = new PDO(DbConfig::DSN, DbConfig::USERNAME, DbConfig::PASSWORD);

        $param = get\_object\_vars($this);

        unset($param['id']);

        $param = join(", ", array\_map(fn($key, $value) => "$key = '$value'", array\_keys($param), array\_values($param)));

        $query = $pdo->prepare("UPDATE customer SET $param WHERE id = $this->id");

        $query->execute([$param]);

    }

}

Nous utilisons la class Customer afin d’utiliser la méthode delete.

    public function delete()

    {

        $pdo = Database::connect();

        $query = $pdo->prepare("DELETE FROM customer WHERE id = ?");

        $query->execute([$this->id]);

    }

Code développé

Vous insérez ***ici tout le*** ***code HTML/CSS/JavaScript/PHP -commenté- développé pour chacun des composants***.

Pour des scripts communs à plusieurs User-Stories (feuille CSS, script d’accès à la base…), copiez une seule fois le code correspondant dans votre dossier.

Afin d’utiliser ces méthodes, on appelle dans notre Controller la méthode getfailed qui contrôle en premier lieu les données. Puis on applique la méthode update pour un mise à jour et add pour ajouter un nouveau client.

    public function addController($classname)

    {

        $formName = 'App\Forms\\' . $classname . 'Form';

        $form = new $formName($\_POST);

        $failed = $form->getFailed();

        if (empty($failed)) {

            if (empty($\_POST['id'])) {

                $form->add();

            } else {

                $form->update();

            }

        } else {

            echo json\_encode($failed);

        }

    }

Si l’on ajoute un client, on vide en premier lieu le modal lors du clic sur le bouton ajouter un client.

/\*\*

 \* @author Simon

 \* 26/10/20

 \* Permet de vidé le modal pour ajouter un client

 \*/

$("#addCustomer").click(function () {

  let inputs = $("#modalCustomer input:not([type = radio])");

  $(inputs).each(function (index, element) {

    $(element).val("");

  });

  $("#modalCustomer").modal("toggle");

});

Au clic sur le bouton valider, on contrôle les informations modifiées. Si toutes les informations sont correctes, le modal se referme et le client est mis à jour ou ajouter. Sinon, les informations erronées sont encadrées et la modification ou l’ajout ne peuvent pas être effectuée tant qu’une des informations est erroné. Tout cela est effectué en JavaScript.

/\*\*

 \* @author Simon

 \* @author Doryan

 \* @author Aymeric

 \* 26/10/20

 \* Permet lors du submit de modifier les clients

 \*/

$("#form\_customer").submit(function (e) {

  e.preventDefault();

  let inputs = $("#modalCustomer input:not([type = radio])");

  let customer = {};

  $(inputs).each(function (index, element) {

    let value = element.value;

    let key = element.id;

    customer[key] = value;

  });

  customer["sector\_activity"] = $("#modalCustomer input[name=sector\_activity]:checked").val();

  $.post("/addCustomer", customer, function (data) {

    //requete AJAX lors de /addCustomer

    let failed;

    let inputs = $("#modalCustomer input:not([type = radio])");

    inputs["sector\_activity"] = $("#modalCustomer input[name=sector\_activity]:checked").val();

    if (!$.isEmptyObject(data)) {

      failed = JSON.parse(data);

      $(inputs).each(function (key, input) {

        if ($.inArray(input.id, failed) !== -1) {

          $(input).removeClass("is-valid");

          $(input).addClass("is-invalid");

        } else {

          $(input).removeClass("is-invalid");

          $(input).addClass("is-valid");

        }

      });

    } else {

      $("#modalCustomer").modal("toggle");

    }

  });

});

Afin de supprimer un client, on appel la méthode delete dans notre Controller.

public function deleteCustomerController()

    {

        $id = $\_POST['id'];

        $customerRepository = new CustomerRepository();

        $customer = $customerRepository->find($id);

        $customer->delete();

    }

Lors de l’appuie sur le bouton supprimer, la méthode récupère l’id et supprime dans la base de donnée le client correspondant.

/\*\*

 \* @author Simon

 \* @author Doryan

 \* @author Aymeric

 \* 26/10/20

 \* Permet de Supprimer un client.

 \*/

$(".delete").click(function () {

  let id = $(this).siblings("#id").val();

  $.post("/deleteCustomer", { id: id }, function () {

    location.reload();

  });

  $("#modalCustomer").modal("toggle");

});

## Suite user story

Les fonctionnalités

Vous énoncez le libellé de la User Story*.*

Vous précisez l’état de la User Story au moment de la livraison (terminée, tâches en cours).

**Volet GS : Lister tous les clients (User story terminée)**

Maquettes

Vous insérez ici une copie de chaque écran développé pour cette User Story.

Les ressources utilisées et particularités de la Story

Le développement des fonctionnalités utilise des ressources diverses extérieures : tables de base de données, classes Métier ou d’accès à la BDD, bibliothèques de code... Le lien avec ces ressources doit pouvoir être retrouvé aisément par le développeur en situation de maintenance.

Si le travail de développement est bien fait, chaque écran a fait l’objet d’un maquettage validé puis d’une ***programmation modulaire dûment commentée, en particulier par le ‘cartouche’ indiquant en début de script l’auteur et les relations de ce script avec les autres ressources***.

Vous pouvez préciser ou rappeler ici certains **algorithmes** ou **règles de gestion** particulièrement importants ou spécifiques.

Code développé

Vous insérez ***ici tout le*** ***code HTML/CSS/JavaScript/PHP -commenté- développé pour chacun des composants***.

Pour des scripts communs à plusieurs User-Stories (feuille CSS, script d’accès à la base…), copiez une seule fois le code correspondant dans votre dossier.