

# PROJET PROMED DOCUMENTATION TECHNIQUE

---

BTS SIO SLAM 2022/2023



Melwin CHENU, Marie-Caroline GRIES, Anastacia NAGULA, Ségolène GANZIN

20/01/2023

# TABLE DES MATIÈRES

Contexte .....	3
Cahier des charges .....	3
Outils utilisés .....	4
Langages utilisés .....	4
Mise en place .....	5
Diagramme cas d'utilisation .....	5
Modèle relationnel projet piscines vannetaises .....	5
Diagramme de classe UML.....	6
Diagramme de la base de données avec typage .....	6
Connection à github .....	8
Modele Vue Controleur .....	13
Index.....	14
Modèle .....	15
BD.....	15
Métier.....	19
Vue .....	22
L'espace patient .....	22
L'espace praticien .....	23
Création d'un nouveau patient .....	23
Chercher un patient .....	24
Liste des rendez-vous.....	26
Inscription du praticien .....	27
Contrôleur .....	28
Création d'un nouveau patient .....	28
Chercher un patient .....	28
Liste des rendez-vous.....	29
Authentification .....	30
Inscription .....	31
Déconnexion .....	31
Validation de formulaire avec Ajax .....	32

## CONTEXTE

La société MediConcept souhaite la mise en place d'une interface de gestion individualisée en ligne, pour les praticiens.

Dans le but de développer sa présence sur internet, la société MediConcept désire développer une interface de gestion en ligne à l'usage exclusif des professions libérales de santé, telles que les kinésithérapeutes, psychothérapeutes, ergothérapeutes, orthophonistes, diététiciens et autres professions médicales.

Dans ce but, l'application nommée ProMed aura pour objectif de permettre aux praticiens de gérer à distance leur patientèle, et ce en toute sécurité.

Les praticiens pourront organiser les heures de rendez-vous.

Il y aura également une autre partie réservée exclusivement aux patients, pour leur permettre de visualiser leurs rendez-vous à venir et pouvoir éventuellement les annuler.

## CAHIER DES CHARGES

Depuis une application WEB sécurisée, il y aura 3 types d'accès :

- « **Accès praticien** » → Protégé par un email et un mot de passe.
  - « **Accès patient** » → Accès avec un email et un mot de passe.
  - « **Inscription** » → À destination d'un nouveau praticien, composé d'un formulaire.
- L'accès praticien sera structuré par plusieurs fonctionnalités :
- **Un agenda récapitulatif** des rendez-vous journalier et éventuelle(s) annulation(s).
  - **Visualisation/modification** des paramètres praticien, données et prises en charge.
  - **Ajout d'une fiche patient.**
  - **Visualisation d'une fiche patient** à partir d'un champ de recherche associé + **modification des données associées.**
  - **Suppression éventuelle** d'une fiche patient.
  - **Ajout de dates de rendez-vous** et **visualisation de la liste des rendez-vous** associées aux prises en charge des patients.
- L'accès patient sera essentiellement composé :
- D'un **récapitulatif de ses rendez-vous.**
  - La **possibilité d'annuler** un rendez-vous.

## OUTILS UTILISÉS



Visual Studio Code



**WampServer**



dbdiagram.io



**Figma**



**GitHub**



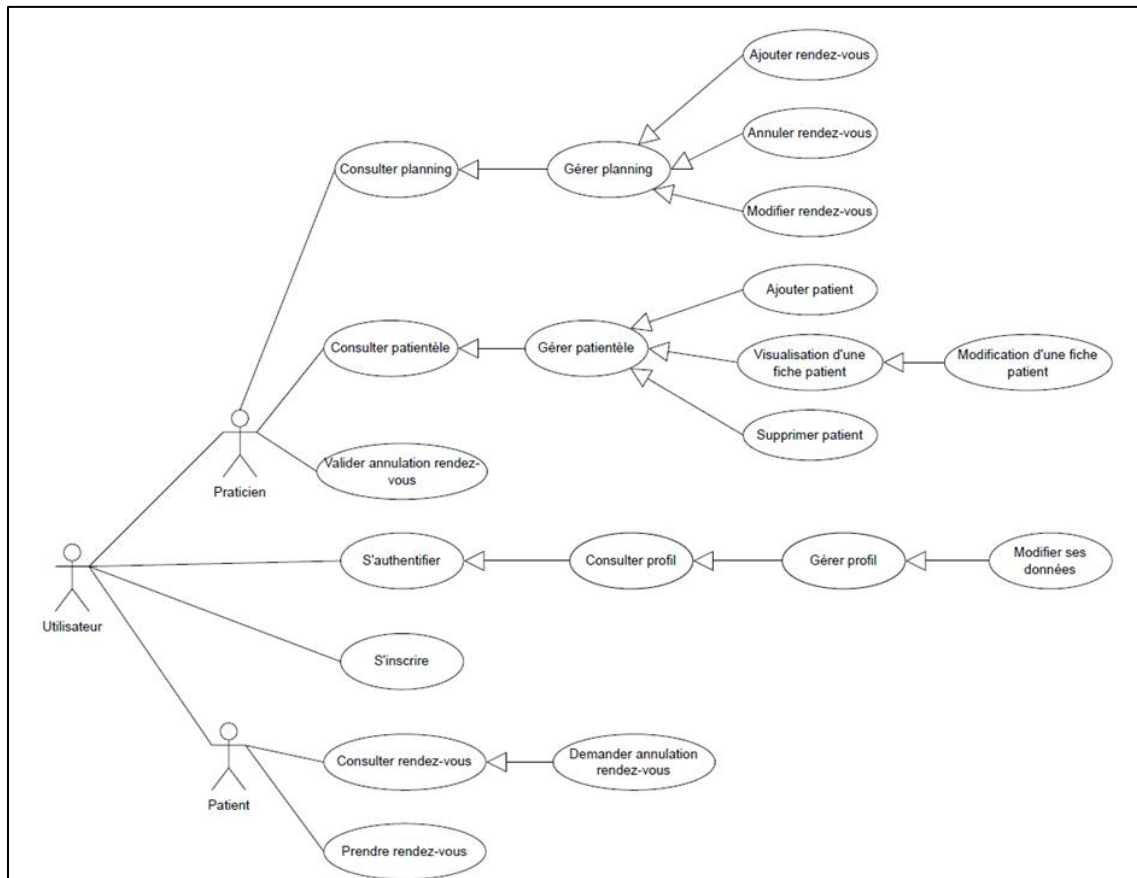
**Notion**

## LANGAGES UTILISÉS



## MISE EN PLACE

### DIAGRAMME CAS D'UTILISATION



Créé sur draw.io

## MODÈLE RELATIONNEL PROJET PISCINES VANNETAISES

adresse (id\_adresse, num, rue, ville, cp)

patient (id\_patient, date\_de\_naissance, #id\_identite)

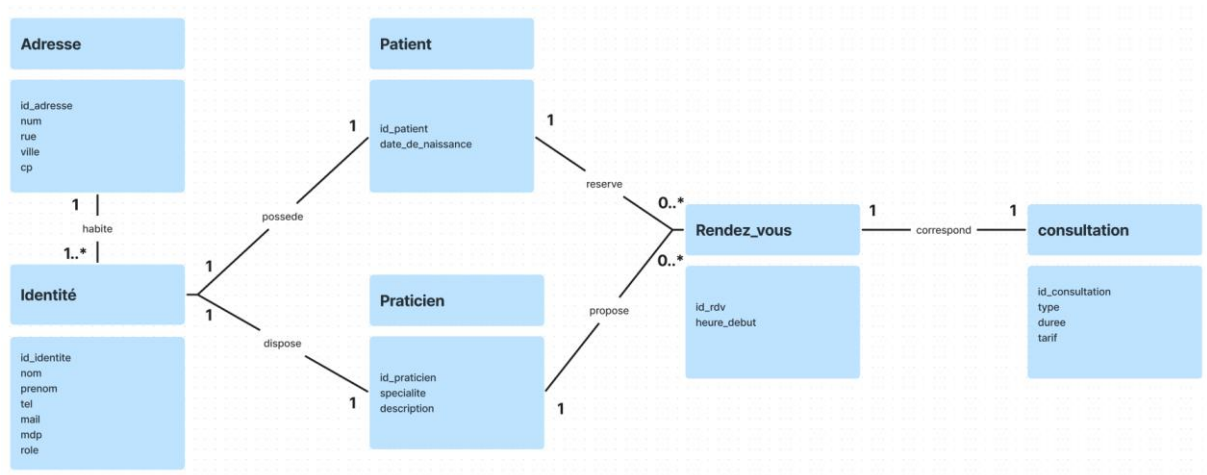
identité (id\_identite, nom, prenom, tel, mail, mdp, role, #id\_adresse)

praticien (id\_praticien, specialite, description, #id\_identite)

rdv (id\_rdv, heure\_debut, statut, #id\_patient, #id\_praticien, #id\_consultation)

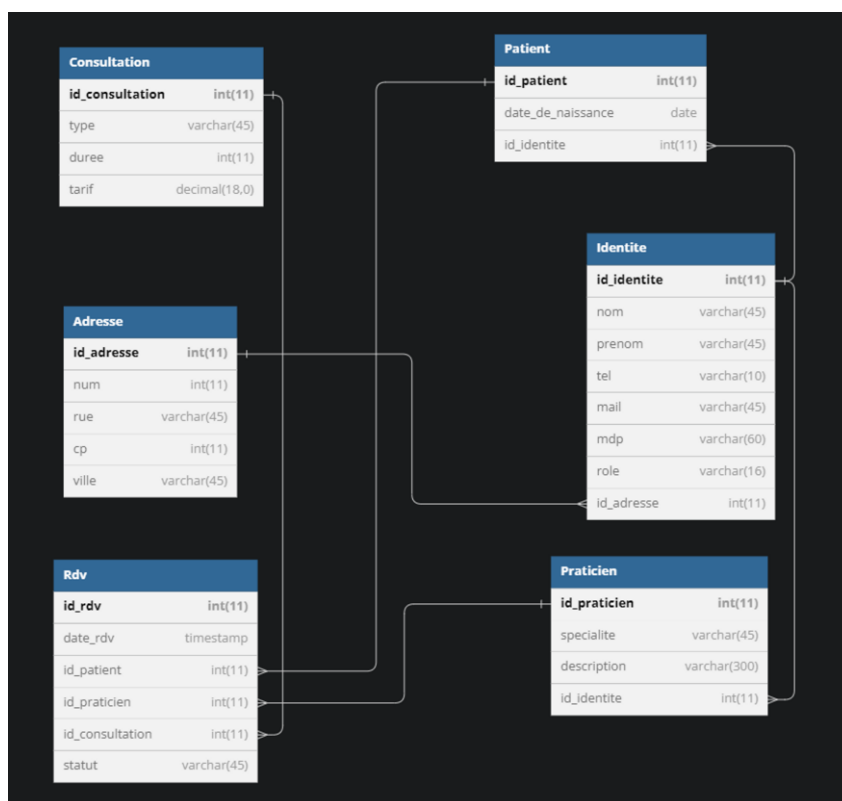
consultation (id\_consultation, type, duree, tarif)

## DIAGRAMME DE CLASSE UML



Réalisé sur [Figma](#)

## DIAGRAMME DE LA BASE DE DONNÉES AVEC TYPAGE



Créé avec [dbdiagram.io](#)

## CRÉATION DE LA BASE DE DONNÉES DANS MYSQL

On a lancé Wamp et ouvert PhpMyAdmin



On a créé une nouvelle base de données nommée promed

### Bases de données

Création d'une base de données

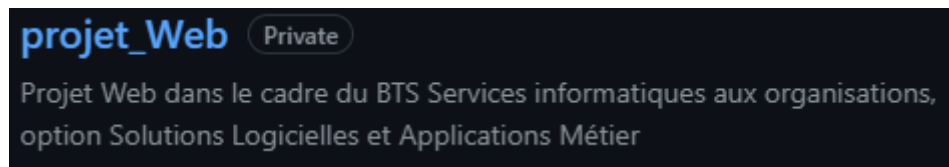
Ensuite on a importé les scripts générés précédemment



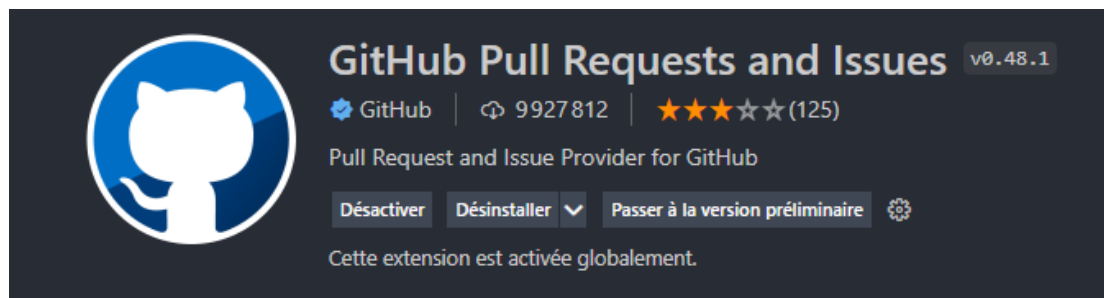
<input type="checkbox"/>	adresse	★	Parcourir	Structure	Rechercher	Insérer	Vider	Supprimer	0	InnoDB	utf8_general_ci	16,0 kio	-
<input type="checkbox"/>	consultation	★	Parcourir	Structure	Rechercher	Insérer	Vider	Supprimer	0	InnoDB	utf8_general_ci	32,0 kio	-
<input type="checkbox"/>	identite	★	Parcourir	Structure	Rechercher	Insérer	Vider	Supprimer	0	InnoDB	utf8_general_ci	32,0 kio	-
<input type="checkbox"/>	patient	★	Parcourir	Structure	Rechercher	Insérer	Vider	Supprimer	0	InnoDB	utf8_general_ci	32,0 kio	-
<input type="checkbox"/>	praticien	★	Parcourir	Structure	Rechercher	Insérer	Vider	Supprimer	0	InnoDB	utf8_general_ci	32,0 kio	-
<input type="checkbox"/>	rdv	★	Parcourir	Structure	Rechercher	Insérer	Vider	Supprimer	0	InnoDB	utf8_general_ci	48,0 kio	-

## CONNECTION À GITHUB

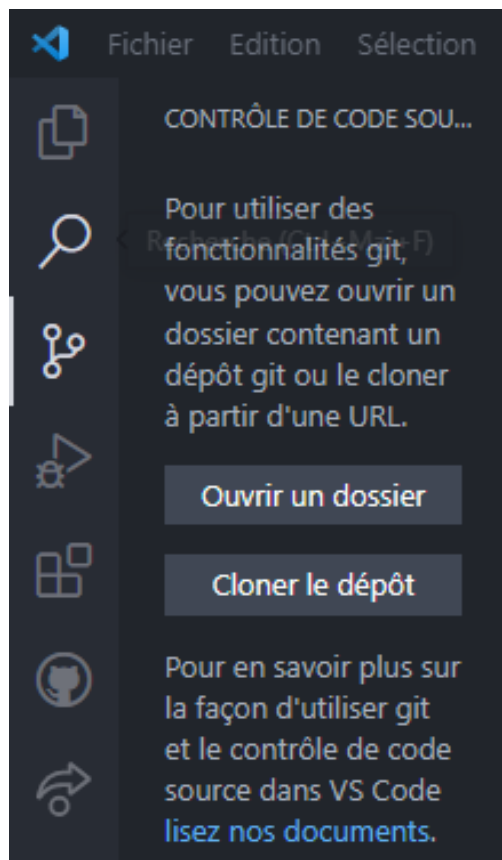
Un membre du groupe a créé un repository pour le projet : « projet\_web »



Dans l'onglet extension de Visual Studio Code (ctrl+Maj+X), on recherche :



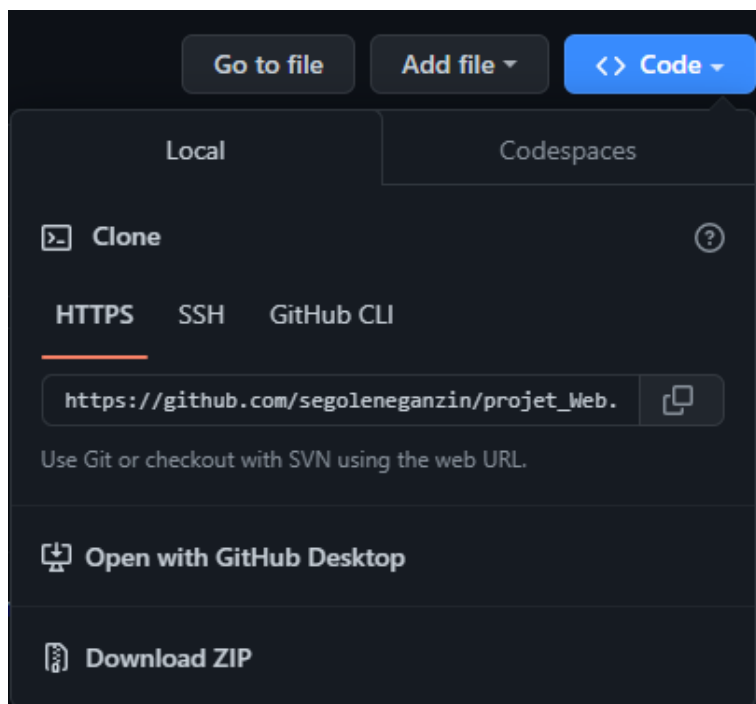
Une fois installé on peut accéder à l'onglet « **Contrôle de code source** » (Ctrl+Maj+G).



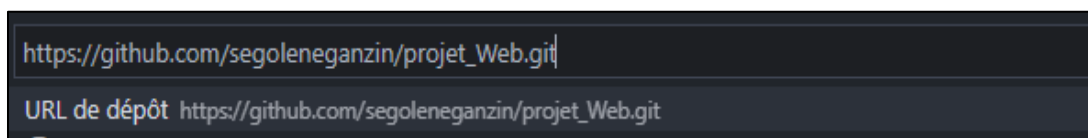
Pour relier Git à un dépôt distant existant il faut **cloner le dépôt**.



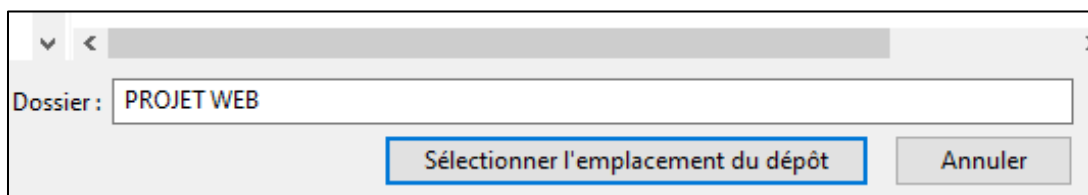
Sur la page GitHub du projet, on récupère l'URL :



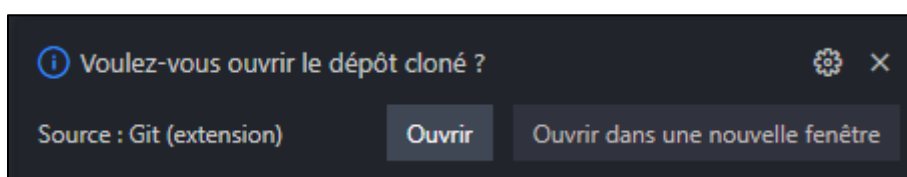
On rentre cette URL dans VSCode :



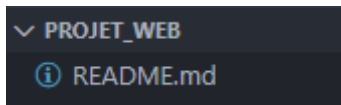
On choisit l'emplacement du dépôt sur le PC :



On ouvre le dépôt cloné :

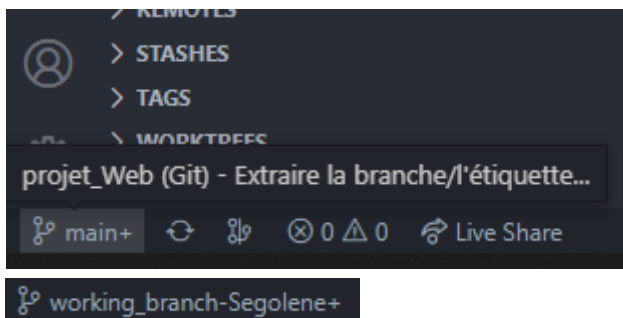


On constate qu'il a bien récupéré le contenu présent en ligne :

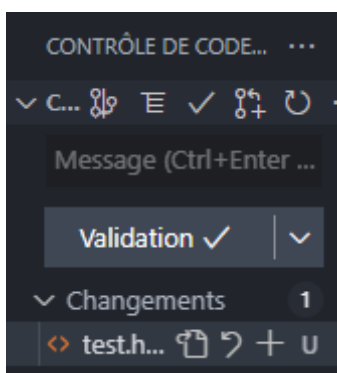
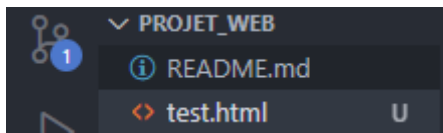


Ce répertoire sera relié à GitHub dès qu'il sera ouvert dans VSCode.

On sélectionne la branche dans laquelle on souhaite travailler en cliquant sur l'onglet tout en bas à gauche :

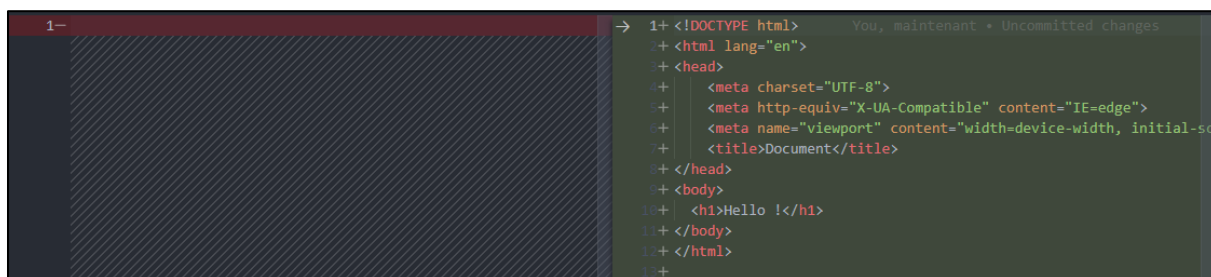


On teste l'ajout d'un nouveau document « test.html », et on constate qu'on a immédiatement une notification au niveau de l'onglet « Contrôle de code source ».



Les documents qui ont été modifiés mais pas commit sont visibles dans l'onglet « changements ».

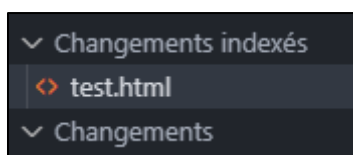
Si l'on clique sur le fichier on peut visualiser les changements effectués



```

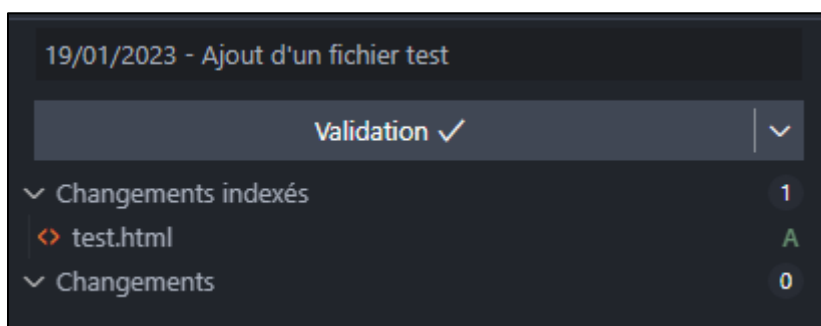
1+ <!DOCTYPE html>
2+ <html lang="en">
3+ <head>
4+   <meta charset="UTF-8">
5+   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6+   <meta name="viewport" content="width=device-width, initial-scale=1">
7+   <title>Document</title>
8+ </head>
9+ <body>
10+   <h1>Hello !</h1>
11+ </body>
12+ </html>
13+
  
```

On ajoute ce fichier (+), il passe dans l'onglet « Changements indexés »

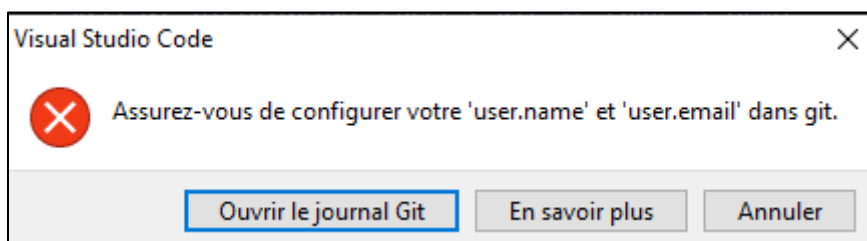


On entre un message de commit (obligatoire), le mieux est de respecter un « format » que l'on se fixe au début du projet pour être logique.

Exemple : « Date, actions effectuées ».



La première fois un message d'erreur s'affiche :



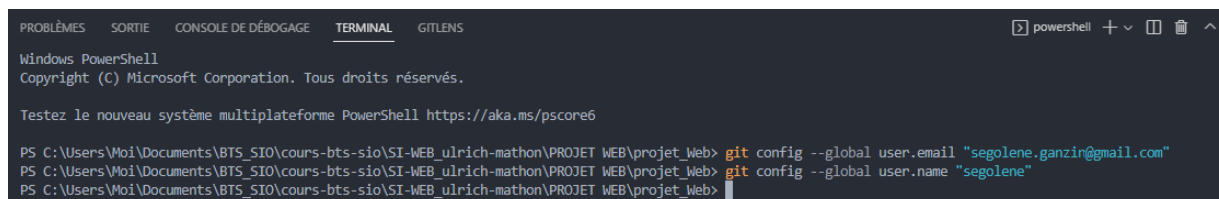
Il faut ouvrir le journal git, et on nous indique la commande à effectuer :

```
Run

git config --global user.email "you@example.com"
git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.
```

Ouvrir ensuite le terminal de VSCode :



```
PROBLÈMES  SORTIE  CONSOLE DE DÉBOGAGE  TERMINAL  GITLENS
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS C:\Users\Moi\Documents\BTS_SIO\cours-bts-sio\SI-WEB_ulrich-mathon\PROJET_WEB\projet_web> git config --global user.email "segolene.ganzin@gmail.com"
PS C:\Users\Moi\Documents\BTS_SIO\cours-bts-sio\SI-WEB_ulrich-mathon\PROJET_WEB\projet_web> git config --global user.name "segolene"
PS C:\Users\Moi\Documents\BTS_SIO\cours-bts-sio\SI-WEB_ulrich-mathon\PROJET_WEB\projet_web>
```

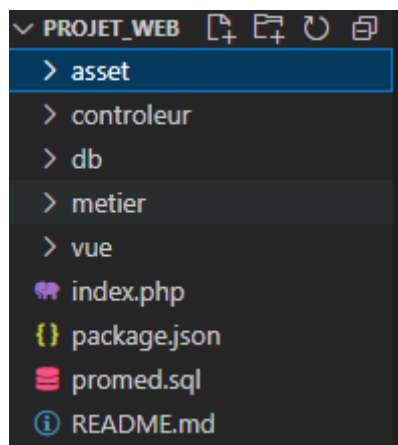
Ensuite on peut cliquer sur « valider » pour commit.

Et on peut constater que l'envoi s'est bien effectué en ligne, sur la branche sélectionnée :

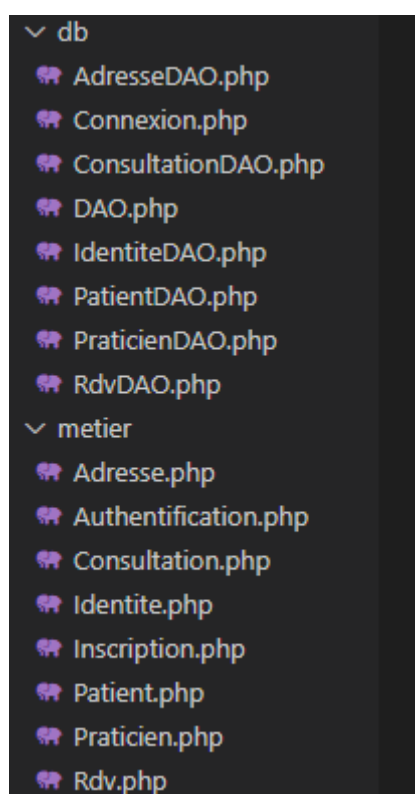
segoleneganzin 19/01/2023 - Ajout d'un fichier test			d85c470 now	🕒 4 commits
📄 README.md	Update README.md			41 minutes ago
📄 test.html	19/01/2023 - Ajout d'un fichier test			now

## MODELE VUE CONTROLEUR

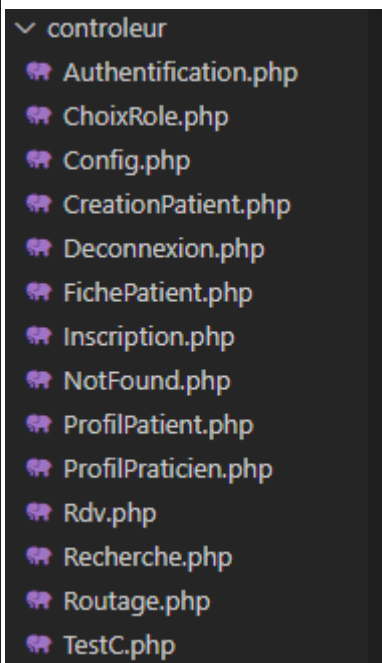
Nous avons respecté une architecture MVC. Le Modèle contient deux packages : db et métier.



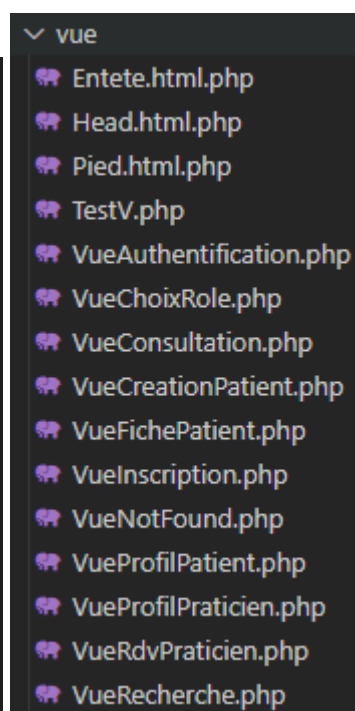
Arborescence du projet



Package db et metier du modèle



Package controleur



Package vue

## INDEX

L'index inclut les contrôleurs Config.php et Routage.php.

Config.php définit des constantes pour l'application. Ces constantes vont regrouper les fichiers nécessaires à l'exécution du programme de chaque Contrôleur.

Routage.php récupère les actions par la méthode GET et redirige vers la page associée.

L'index met en œuvre le système de routage pour diriger les requêtes vers les contrôleurs secondaires appropriés en fonction de l'action demandée par l'utilisateur.

```
index.php x
index.php > ...
1  <?php
2
3  /** Contrôleur principal */
4
5
6  require dirname(__FILE__) . "/contrôleur/Config.php";
7
8  require RACINE . "/contrôleur/Routage.php";
9
10 if (isset($_GET["action"])) {
11     $action = $_GET["action"];
12 } else {
13     $action = "default";
14 }
15
16 //Ajoute un contrôleur secondaire ($fichier) en fonction du métier ($action)
17 $fichier = redirigeVers($action);
18 require RACINE . "/contrôleur/" . $fichier;
19
```

Fichier Index.php (contrôleur principal)

```
index.php M  Routage.php x
contrôleur > Routage.php > redirigeVers
1  <?php
2
3  /**
4   * Module du routing (routage).
5   * Chaque action est récupérée par la méthode : $_GET
6   */
7
8  function redirigeVers($action = "default")
9  {
10
11     $lesActions = [];
12     $lesActions["default"] = "ChoixHole.php";
13     $lesActions["deconnexion"] = "Deconnexion.php";
14     $lesActions["connexion"] = "Authentification.php";
15     //*****Parcours praticien
16     $lesActions["inscription"] = "Inscription.php";
17     $lesActions["connexion-praticien"] = "Authentification.php";
18     $lesActions["rdv-praticien"] = "Rdv.php";
19     $lesActions["creation-patient"] = "CreationPatient.php";
20     $lesActions["recherche"] = "Recherche.php";
21     $lesActions["fiche-patient"] = "FichePatient.php";
22     //*****Parcours Patient
23     $lesActions["connexion-patient"] = "Authentification.php";
24     $lesActions["rdv-patient"] = "ProfilPatient.php";
25
26     //Pour les tests
27     $lesActions["scripts"] = "Testc.php";
28
29     $contrôleur_id = $lesActions[$action];
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
index.php M  Config.php x
contrôleur > Config.php > ...
1
2
3
4
5
6
7  //Chemin absolu de l'application :
8  define("RACINE", dirname(__DIR__));
9
10 /**Constantes chemin vers le metier
11 define("ADRESSE_METIER", RACINE . "/metier/Adresse.php");
12 define("AUTHENTIFICATION_METIER", RACINE . "/metier/Authentication.php");
13 define("CONSULTATION_METIER", RACINE . "/metier/Consultation.php");
14 define("IDENTITE_METIER", RACINE . "/metier/Identite.php");
15 define("INSCRIPTION_METIER", RACINE . "/metier/Inscription.php");
16 define("PATIENT_METIER", RACINE . "/metier/Patient.php");
17 define("PRATICIEN_METIER", RACINE . "/metier/Praticien.php");
18 define("RDV_METIER", RACINE . "/metier/Rdv.php");
19
20 /**Constantes chemin vers la dao
21 define("ADRESSE_DAO", RACINE . "/db/AdresseDAO.php");
22 define("CONNEXION", RACINE . "/db/Connexion.php");
23 define("CONSULTATION_DAO", RACINE . "/db/ConsultationDAO.php");
24 define("DAO", RACINE . "/db/DAO.php");
25 define("IDENTITE_DAO", RACINE . "/db/IdentiteDAO.php");
26 define("PATIENT_DAO", RACINE . "/db/PatientDAO.php");
27 define("PRATICIEN_DAO", RACINE . "/db/PraticienDAO.php");
28 define("RDV_DAO", RACINE . "/db/RdvDAO.php");
29
30 /**Constantes chemin vers la vue
31 define("PATH_HEAD", RACINE . "/vue/Head.html.php");
32 define("PATH_ENTETE", RACINE . "/vue/Entete.html.php");
33 define("PATH_TESTV", RACINE . "/vue/TestV.php");
34 define("PATH_AUTHENTIFICATION", RACINE . "/vue/VueAuthentication.php");
35
```

Fichier Routage.php (contrôleur secondaire)

Fichier Config.php (contrôleur secondaire)

# MODÈLE

Le modèle va contenir les fichiers traitant des données et des logiques en lien avec la base de données.

## BD

Le package bd contient les classes nécessaires à la connexion à la base de données, et les DAO qui définissent les méthodes de création, de suppression, de mise-à-jour et de lecture des tables (CRUD).

### CONNEXION À LA BASE DE DONNÉES

Le fichier Connexion.php s'occupe d'effectuer la connexion à la base de données MySQL :

```
db > Connexion.php X
1  <?php
2
3  namespace DB\Connexion {
4
5      use PDOException;
6
7      class Connexion
8      {
9          static function getInstance()
10         {
11             static $dbh = NULL;
12             if ($dbh == NULL) {
13
14                 $dsn = "mysql:host=localhost:3306;dbname=promed";
15                 $username = "root";
16                 $password = "";
17
18                 /** OPTIONS SQL **/
19                 //pour expliciter le namespace, on préfixe la classe avec \
20                 $options = array(
21                     \PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8",
22                     \PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION //gestion des erreurs SQL
23                 );
24
25                 try {
26                     $dbh = new \PDO($dsn, $username, $password, $options);
27                 } catch (PDOException $e) {
28                     echo "Problème de connexion!\n\r<br>", $e;
29                 }
30             }
31             return $dbh;
32         }
33     }
34 }
35
```

Fichier Connexion.php

## DATA ACCESS OBJECT

Pour chaque table dans la base de données, nous avons programmés une DAO. Chaque DAO de table sont des filles de la classe DAO.php qui va lister les méthodes attendues dans chacune des DAO filles. La classe mère DAO va aussi établir la connexion aux données en instanciant la classe Connexion :

```
DAO.php M X
db > DAO.php > {} DAO
1  <?php
2
3  namespace DAO {
4
5      use DB\Connexion\Connexion;
6
7      abstract class DAO
8      {
9
10         abstract function read($id);
11
12         abstract function update($objet);
13
14         abstract function delete($objet);
15
16         abstract function create($objet);
17
18         protected $key;
19
20         protected $table;
21
22         function __construct($key, $table)
23         {
24             $this->key = $key;
25             $this->table = $table;
26         }
27
28         function getLastKey()
29         {
30
31             return Connexion::getInstance()->lastInsertId();
32         }
33     }
34 }
```

*Fichier DAO.php (classe abstraite, mère des classes DAO)*

La classe DAO mère est abstraite, elle ne peut pas être instanciée, elle se contente de lister les fonctions nécessaires à ses classes filles.



Exemple d'une classe DAO fille :

```

db > IdentiteDAO.php > {} DAO\Identite
1  <?php
2
3  namespace DAO\Identite {
4
5      use DB\Connexion\Connexion;
6
7      class IdentiteDAO extends \DAO\DAO
8      {
9
10
11         function __construct()
12         {
13             parent::__construct("id_identite", "identite");
14         }
15
16         public function create($objet)
17         { ...
18         }
19
20         public function read($id)
21         { ...
22         }
23
24         public function update($objet)
25         { ...
26             mdp = :mdp, role = :role, id_adresse = :id_adresse WHERE $this->key=:id; ...
27         }
28
29         public function delete($objet)
30         { ...
31             SET FOREIGN_KEY_CHECKS=1; ...
32         }
33
34         public function readAllPatients()
35         { ...
36         }
37
38         static function getUtilisateurByMailU($mail)
39         { ...
40         }
41     }
42 }
  
```

Fichier IdentiteDAO.php (classe fille de DAO.php)

Certaines DAO de table ajoutent une fonction readAll() pour permettre la récupération de toute la table.

Par exemple la RdvDAO :

```

public function readAllRdv()
{
    $sql = "SELECT * FROM $this->table";
    $stmt = Connexion::getInstance()->prepare($sql);
    $stmt->execute();

    $resultat = array();
    while ($row = $stmt->fetch()) {
        $id_rdv = $row["id_rdv"];
        $date_rdv = $row["date_rdv"];
        $id_praticien = $row["id_praticien"];
        $id_patient = $row["id_patient"];
        $id_consultation = $row["id_consultation"];
        $statut = $row["statut"];
        $daoPrat = new \DAO\Praticien\PraticienDAO();
        $praticien = $daoPrat->read($id_praticien);
        $daoPat = new \DAO\Patient\PatientDAO();
        $patient = $daoPat->read($id_patient);
        $daoConsult = new \DAO\Consultation\ConsultationDAO();
        $consultation = $daoConsult->read($id_consultation);
        $rep = new \Promed\Rdv\Rdv($date_rdv, $praticien, $patient, $consultation, $statut);
        $rep->setId($id_rdv);
        $resultat[] = $rep;
    }
    return $resultat;
}
  
```

Méthode readAllRdv() (Classe RdvDAO.php)

## PREPARE STATEMENT

Afin de sécuriser les requêtes SQL, on ne les envoie pas directement en clair dans la requête, on va les préparer. Dans la requête, on pointe les valeurs comme ceci «:nom, :prenom, :tel, ... », puis on va ensuite, hors requête, lier les valeurs pointées aux valeurs récupérées de l'objet créé.

```
public function create($objet)
{
    try {
        $sql = "INSERT INTO $this->table (num, rue, cp, ville)
VALUES (:num, :rue, :cp, :ville)";
        $stmt = \DB\Connexion\Connexion::getInstance()->prepare($sql);
        $num = $objet->getNum();
        $rue = $objet->getRue();
        $cp = $objet->getCp();
        $ville = $objet->getVille();
        $stmt->bindParam(':num', $num);
        $stmt->bindParam(':rue', $rue);
        $stmt->bindParam(':cp', $cp);
        $stmt->bindParam(':ville', $ville);
        $stmt->execute();
        $objet->setId(parent::getLastKey());
    }
```

Méthode create() d'une Identite (Classe IdentiteDAO.php)

On adoptera cette préparation pour chacune des méthodes du CRUD.

## TRY / CATCH

Afin de mieux gérer les erreurs dans le processus d'exécution, nous avons mis en place des Try / Catch permettant d'attraper l'erreur si elle se produit et de la gérer. Nous utilisons cette gestion d'erreurs dans toutes les méthodes du CRUD.

```
public function read($id)
{
    try {
        // On utilise le prepared statement qui simplifie les typages
        $sql = "SELECT * FROM $this->table WHERE $this->key=:id";
        $stmt = \DB\Connexion\Connexion::getInstance()->prepare($sql);
        $stmt->bindParam(':id', $id);
        $stmt->execute();

        $row = $stmt->fetch();
        $id_adresse = $row["id_adresse"];
        $num = $row["num"];
        $rue = $row["rue"];
        $cp = $row["cp"];
        $ville = $row["ville"];
        $rep = new \Promed\Adresse\Adresse($num, $rue, $cp, $ville);
        $rep->setId($id_adresse);
    } catch (\PDOException $e) {
        die("Erreur !: " . $e->getMessage());
    }
    return $rep;
}
```

Try/Catch dans la méthode read() d'une Adresse (Classe AdresseDAO.php)

## MÉTIER

Le packages métier regroupent les classes métiers de chaque table. Chaque classe définit un objet avec ses attributs, son constructeur, et les méthodes pour accéder et modifier ses attributs :

```
Identite.php M X
metier > Identite.php > {} Promed\Identite > Identite > getPrenom
1  <?php
2
3  namespace Promed\Identite {
4
5      class Identite
6      {
7
8          private $idIdentite = 0;
9          private $nom = "";
10         private $prenom = "";
11         private $tel = "";
12         private $mail = "";
13         private $mdp = "";
14         private $role = "";
15         private $adresse;
16
17         function __construct($nom, $prenom, $tel, $mail, $mdp, $role, $adresse)
18         {
19             $this->nom = $nom;
20             $this->prenom = $prenom;
21             $this->tel = $tel;
22             $this->mail = $mail;
23             $this->mdp = $mdp;
24             $this->role = $role;
25             $this->adresse = $adresse;
26         }
27     }
28 }
```

Attributs / constructeur (Classe Identite.php)

```
Identite.php M X
metier > Identite.php > {} Promed\Identite > Identite > getPrenom
28  public function getId()
29  { ...
31  }
32  public function getNom()
33  { ...
35  }
36  public function getPrenom()
37  { ...
39  }
40  public function getTel()
41  { ...
43  }
44  public function getMail()
45  { ...
47  }
48  public function getMdp()
49  { ...
51  }
52  public function getRole()
53  { ...
55  }
56  public function getAdresse()
57  { ...
59  }
60  public function setId($idIdentite)
61  { ...
64  }
65  public function setNom($nom)
66  { ...
69  }
70  public function setPrenom($prenom)
71  { ...
74  }
75  public function setTel($tel)
76  { ...
79  }
80  public function setMail($mail)
81  { ...
84  }
85  public function setMdp($mdp)
86  { ...
89  }
90  public function setRole($role)
91  { ...
94  }
95  public function setAdresse($adresse)
96  { ...
99  }
100 }
```

Méthodes getters/setters (Classe Identite.php)

On va aussi trouver deux fichiers : Authentication.php et Inscription.php. Ces deux classes vont permettre l'authentification et l'inscription des patients et praticiens.

L'inscription va notamment permettre le hashage du mot de passe avant l'envoi en BD des données, et hiérarchiser les actions à effectuer afin qu'il n'y ait pas de conflit avec les clefs étrangères (on crée d'abord une adresse qui ne possède pas de FK, puis une identité qui contient l'adresse, puis ici un praticien qui possède une identité) :

```
class Inscription
{
    static function inscriptionPraticien($nomId, $prenomId, $telId, $emailId, $numAdr, $nomAdr, $cpAdr, $villeAdr)
    {
        $daoAdresse = new \DAO\Adresse\AdresseDAO();
        $adresse = new Adresse($numAdr, $nomAdr, $cpAdr, $villeAdr);
        $daoAdresse->create($adresse);

        $daoIdentite = new \DAO\Identite\IdentiteDAO();
        //methode de hashage de mot de passe
        $mdp_brut = $mdp;
        $hash = password_hash($mdp_brut, PASSWORD_DEFAULT);
        $identite = new Identite($nomId, $prenomId, $telId, $emailId, $hash, "praticien", $adresse->getId());
        $daoIdentite->create($identite);

        $daoPraticien = new \DAO\Praticien\PraticienDAO();
        $praticien = new Praticien($specialitePrat, $descPrat, $identite->getId());
        $daoPraticien->create($praticien);

        exit;
    }
}
```

Méthode inscriptionPraticien (Classe Inscription.php)

La classe Authentication va quant à elle permettre la comparaison entre les données envoyées via le formulaire et celle présente dans la BD :

```
class Authentication
{
    static function login($mail, $mdp)
    {
        if (!isset($_SESSION)) {
            session_start();
        }
        $util = \DAO\Identite\IdentiteDAO::getUtilisateurByMailU($mail);
        $mdpBD = $util["mdp"];
        $role = $util["role"];
        echo "$mail, $mdpBD, $mdp";
        var_dump(password_verify($mdp, $mdpBD));

        $verify = password_verify($mdp, $mdpBD);

        if ($verify) {
            // le mot de passe est celui de l'utilisateur dans la base de données
            $_SESSION["mail"] = $mail;
            $_SESSION["mdp"] = $mdpBD;
            $_SESSION["role"] = $role;
        }
    }
}
```

Méthode login (Classe Authentication.php)

Elle va aussi permettre à l'utilisateur de se déconnecter et vérifier à chaque action si l'utilisateur est bien connecté (ce qui permettra de sécuriser les pages, inaccessibles si l'utilisateur n'est pas connecté).

```
static function logout()
{
    if (!isset($_SESSION)) {
        session_start();
    }
    unset($_SESSION["mail"]);
    unset($_SESSION["mdp"]);
    unset($_SESSION["role"]);
    header('Location: ./');
}
```

Méthode logout (Classe Authentification.php)

```
static function isLoggedIn()
{
    if (!isset($_SESSION)) {
        session_start();
    }
    $ret = false;

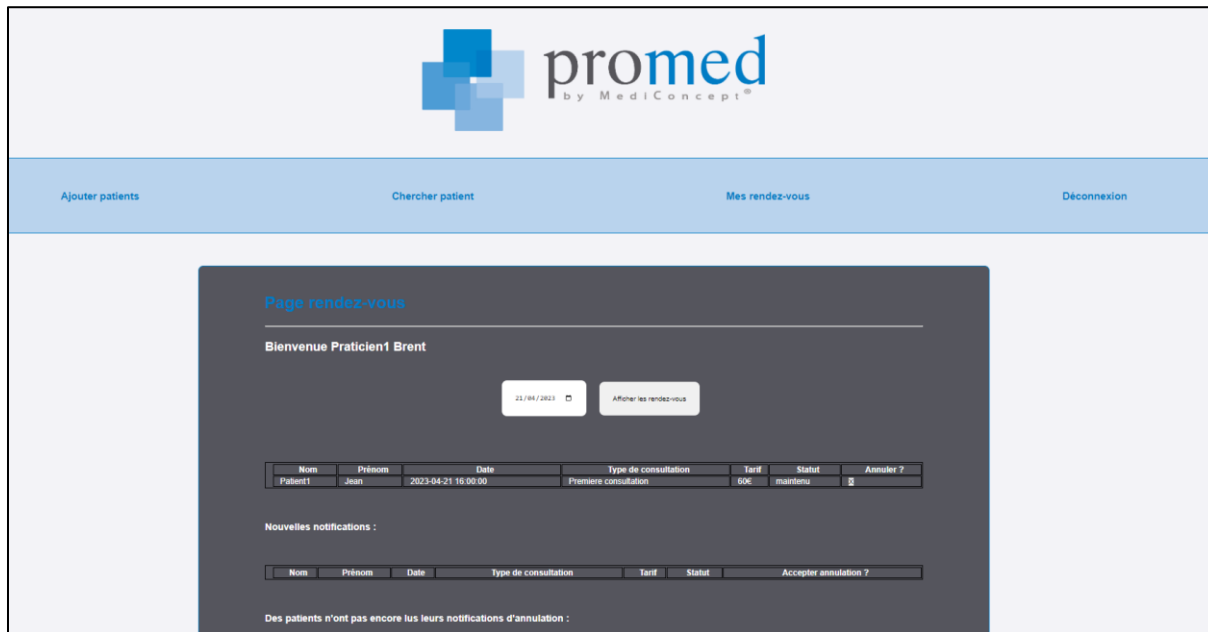
    if (isset($_SESSION["mail"])) {
        $util = IdentiteDAO::getUtilisateurByMailU($_SESSION["mail"]);
        if (
            $util["mail"] == $_SESSION["mail"] && $util["mdp"] == $_SESSION["mdp"]
        ) {
            $ret = true;
        }
    }
    return $ret;
}
```

Méthode isLoggedIn (Classe Authentification.php)



## L'ESPACE PRATICIEN

L'espace praticien se compose de plusieurs pages : une page qui affiche la liste des rendez-vous, une page de recherche patient, qui permet d'accéder aux fiches patient, et une page permettant la création d'un nouveau patient.

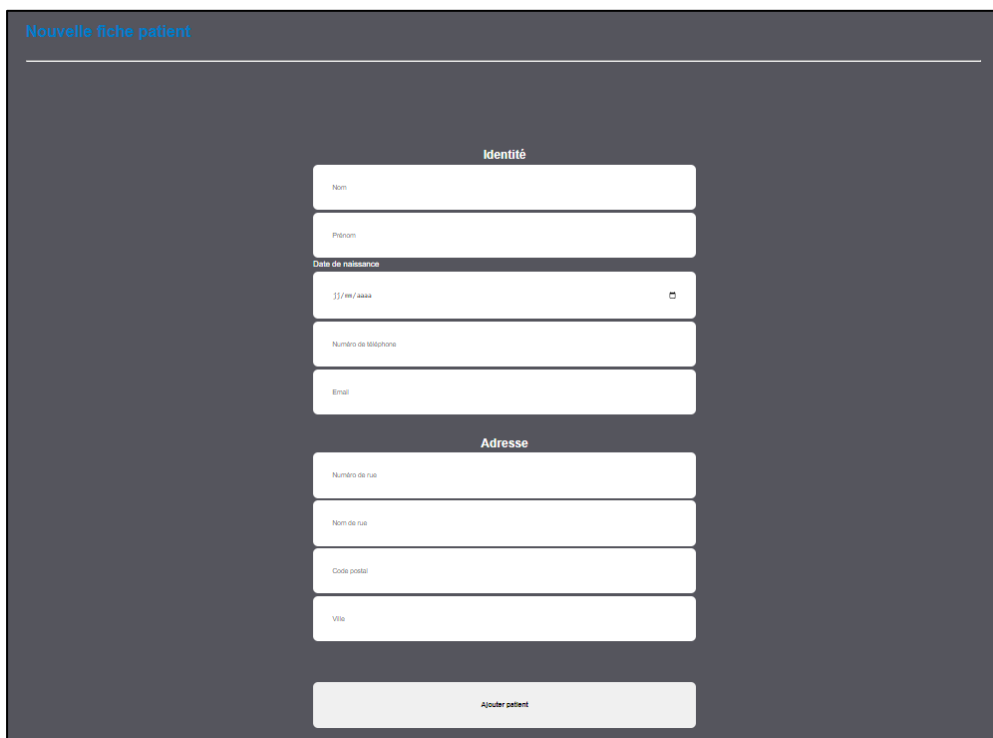


The screenshot shows the 'Page rendez-vous' (Appointment Page) of the Promed application. At the top, there is a navigation bar with four links: 'Ajouter patients', 'Chercher patient', 'Mes rendez-vous', and 'Déconnexion'. The main content area is titled 'Page rendez-vous' and 'Bienvenue Praticien1 Brent'. It features a date selector set to '22/04/2023' and a button 'Afficher les rendez-vous'. Below this is a table of appointments:

Nom	Prénom	Date	Type de consultation	Tarif	Statut	Annuler ?
Patient1	Jean	2023-04-21 16:00:00	Premiere consultation	60€	maintenu	<input type="checkbox"/>

Below the table, there is a section for 'Nouvelles notifications :'. It contains a table with columns: 'Nom', 'Prénom', 'Date', 'Type de consultation', 'Tarif', 'Statut', and 'Accepter annulation ?'. At the bottom, a message states: 'Des patients n'ont pas encore lus leurs notifications d'annulation :'. The interface is clean and professional, with a dark grey background and white text.

## CRÉATION D'UN NOUVEAU PATIENT



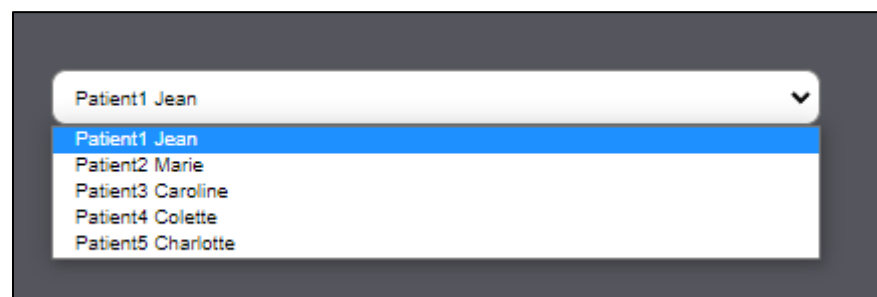
The screenshot shows the 'Nouvelle fiche patient' (New Patient Form) in the Promed application. The form is titled 'Nouvelle fiche patient' and is divided into two main sections: 'Identité' and 'Adresse'. The 'Identité' section includes fields for 'Nom', 'Prénom', 'Date de naissance' (with a date picker), 'Numéro de téléphone', and 'Email'. The 'Adresse' section includes fields for 'Numéro de rue', 'Nom de rue', 'Code postal', and 'Ville'. At the bottom of the form is a button labeled 'Ajouter patient'. The form is designed with a dark grey background and white text, providing a clear and structured layout for data entry.

La VueCreationPatient.php contient un formulaire en HTML avec les entrées attendues dans les tables identité, adresse et patient (un patient ayant une identité, une identité étant liée à une adresse) :

```
VueCreationPatient.php
1 <div class="container nouveau-patient">
2   <h1>Nouvelle fiche patient</h1>
3   <hr />
4   <div class="form">
5     <form class="form--input-container" action="./?action=inscription" method="POST">
6       <h2>Identité</h2>
7       <input type="text" name="nom" placeholder="Nom" />
8       <input type="text" name="prenom" placeholder="Prénom" />
9       <label for="date_naissance">Date de naissance</label>
10      <input type="date" name="date_naissance" placeholder="Date de naissance" />
11      <!-- varchar pour ne pas que le 0 disparaisse -->
12      <input type="text" name="tel" placeholder="Numéro de téléphone" />
13      <input type="email" name="email" placeholder="Email" />
14      <h2>Adresse</h2>
15      <!-- Le numero de rue est en varchar car il peut y avoir des bis/ter -->
16      <input type="text" name="num" placeholder="Numéro de rue" />
17      <input type="text" name="rue" placeholder="Nom de rue" />
18      <input type="number" name="cp" placeholder="Code postal" />
19      <input type="text" name="ville" placeholder="Ville" />
20
21      <input type="submit" value="Ajouter patient" class="button-submit" />
22    </form>
23  </div>
24 </div>
25 </div>
```

## CHERCHER UN PATIENT

Pour prendre rendez-vous avec un patient, le praticien va devoir le chercher via l'onglet « Chercher patient », en sélectionnant le patient recherché, il pourra voir sa fiche et potentiellement poser un rendez-vous.



Deux fichiers vue vont permettre l’affichage : VueRecherche.php et VueFichePatient.php

```

VueRecherche.php X
vue > VueRecherche.php > div.container.recherche
1 <div class="container recherche">
2   <h1>Rechercher un patient</h1>
3   <hr />
4   <div class="form">
5     <form class="form--input-container" action="" method="POST">
6       <select name="identite_id">
7         <?php foreach ($identites as $identite) : ?>
8           <option value="<?php echo $identite->getId(); ?>">
9             <?php echo $identite->getNom() . ' ' . $identite->getPrenom(); ?>
10          </option>
11        <?php endforeach; ?>
12      </select>
13      <input type="submit" name="submit" value="Rechercher" />
14    </form>
15  </div>
16</div>
17

```

Le fichier VueFichePatient.php contient du HTML pour la structure et du PHP pour l’affichage des données :

```


VueFichePatient.php X
vue > VueFichePatient.php > div#fiche_patient.container > form.form--input-container
1 <div class="container" id="fiche_patient">
2   <h1>Fiche patient</h1>
3   <hr />
4   Nom : <?php echo $fiche->getNom(); ?><br>
5   Prénom : <?php echo $fiche->getPrenom(); ?><br>
6   Adresse : <?php echo $adr; ?><br>
7   Téléphone : <?php echo $fiche->getTel(); ?><br>
8   Mail : <?php echo $fiche->getMail(); ?><br>
9
10  <form class="form--input-container" action="" method="POST">
11    <label for="prise_rdv">Prendre un rendez-vous</label>
12    <input type="date" name="prise_date" placeholder="Date de Rendez-vous" />
13    <input type="time" id="prise_heure" name="prise_heure" min="09:00" max="19:00" />
14    <select name="consultation">
15      <?php foreach ($consultations as $consultation) : ?>
16        <option value="<?php echo $consultation->getId(); ?>">
17          <?php echo $consultation->getType(); ?>
18        </option>
19      <?php endforeach; ?>
20    </select>
21    <input type="submit" value="Prendre Rendez-Vous" name="submit" />
22  </form>
23</div>


```

## LISTE DES RENDEZ-VOUS

**Page rendez-vous**

**Bienvenue Praticien2 Matthew**

21/04/2023  [Afficher les rendez-vous](#)

Nom	Prénom	Date	Type de consultation	Tarif	Statut	Annuler ?
Patient1	Jean	2023-04-24 14:15:00	Premiere consultation	60€	maintenu	

**Nouvelles notifications :**

Nom	Prénom	Date	Type de consultation	Tarif	Statut	Accepter annulation ?
Patient4	Colette	2023-04-19 14:53:25	Suivi	50€	demande d'annulation	<input type="checkbox"/> Oui <input checked="" type="checkbox"/> Non
Patient5	Charlotte	2023-04-19 14:53:25	Suivi	50€	demande d'annulation	<input type="checkbox"/> Oui <input checked="" type="checkbox"/> Non

**Des patients n'ont pas encore lus leurs notifications d'annulation :**

Nom	Prénom	Date	Type de consultation	Tarif	Statut
Patient4	Colette	2023-04-20 14:53:25	Bilan	30€	annulé
Patient5	Charlotte	2023-04-20 14:53:25	Bilan	30€	annulé

La VueRdvPraticien présente des tables dans lesquelles seront affichées les données concernant l'identité du patient, la date de rendez-vous, etc. Le code mêle donc du HTML pour la structure et du PHP pour l'affichage des données :

```
<table class="table">
  <tr>
    <th>Nom</th>
    <th>Prénom</th>
    <th>Date</th>
    <th>Type de consultation</th>
    <th>Tarif</th>
    <th>Statut</th>
    <th>Annuler ?</th>
  </tr>
  <?php
  foreach ($rdvs as $rdv) {
    // La fonction substr prend trois arguments : la chaîne de caractères d'origine,
    // la position de départ et le nb de caractère à extraire (on ne veut que la date, pas l'heure pour le filtre)
    if (($dateSelect == substr($rdv->getDateRdv(), 0, 10) && $rdv->getStatut() == "maintenu")) {
      echo "<tr>" .
        "<td>" . $rdv->getPat()->getIdentite()->getNom() . "</td>" .
        "<td>" . $rdv->getPat()->getIdentite()->getPrenom() . "</td>" .
        "<td>" . $rdv->getDateRdv() . "</td>" .
        "<td>" . $rdv->getConsultation()->getType() . "</td>" .
        "<td>" . $rdv->getConsultation()->getTarif() . "</td>" .
        "<td>" . $rdv->getStatut() . "</td>" .
        "<td>" .
          <form method="post">
            <input type="hidden" name="id" value=" " . $rdv->getId() . ">
            <button type="submit" name="annuler">X</button>
          </form>
        "</td>" .
        "</tr>";
    }
  }
  </tr>
</table>
```

La vue permet de filtrer les rendez-vous par date.

## INSCRIPTION DU PRATICIEN

### Nouveau praticien

#### Identité

Nom

Prénom

Numéro de téléphone

root

#### Adresse

Numéro de rue

Nom de rue

Code postal

Ville

#### Activité professionnelle

Spécialité

Description

\*\*\*\*\*

Le mot de passe doit contenir au moins :

- minimum 8 caractères
- une lettre majuscule
- une lettre minuscule
- un chiffre
- un caractère spécial
- maximum 15 caractères

S'enregistrer

Retour

# CONTRÔLEUR

## CRÉATION D'UN NOUVEAU PATIENT

CreationPatient.php récupère les données passées dans le formulaire de VueCreationPatient.php, appelle la fonction inscriptionPatient pour créer de nouvelles entrées dans la base grâce aux données récupérées.

```
CreationPatient.php
controleur > CreationPatient.php > ...
20 // recuperation des donnees POST
21 if (isset($_POST["nom"]) && isset($_POST["prenom"]) && isset($_POST["date_naissance"]) && isset($_POST["tel"]
22     $nom = $_POST["nom"];
23     $prenom = $_POST["prenom"];
24     $date_naissance = $_POST["date_naissance"];
25     $tel = $_POST["tel"];
26     $email = $_POST["email"];
27     $num = $_POST["num"];
28     $rue = $_POST["rue"];
29     $cp = $_POST["cp"];
30     $ville = $_POST["ville"];
31
32 // mot de passe pour les test
33 $mdp = "toto";
34
35 // inscription
36 \Promed\Inscription\Inscription::inscriptionPatient($nom, $prenom, $date_naissance, $tel, $email, $num,
37 ) else {
38     $nom = null;
39     $prenom = null;
40     $tel = null;
41     $date_naissance = null;
42     $email = null;
43     $num = null;
44     $rue = null;
45     $cp = null;
46     $ville = null;
47     $mdp = null;
48 }
```

## CHERCHER UN PATIENT

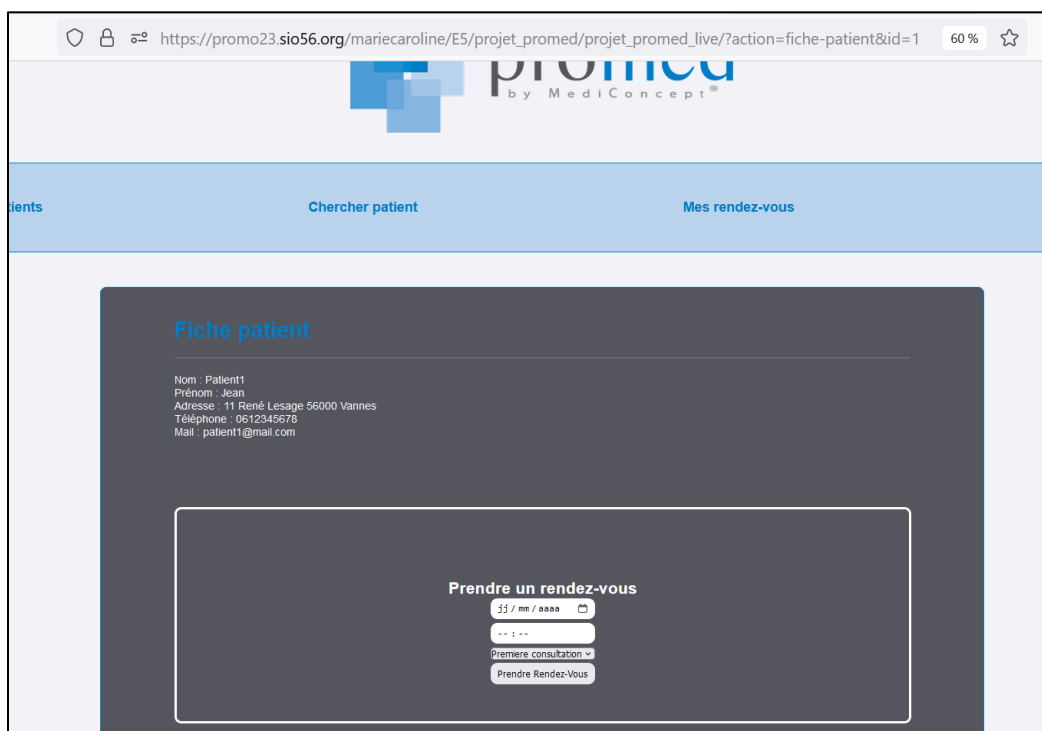
Recherche.php propose une liste de patients parmi laquelle le praticien peut choisir. Son action le redirigera vers une page FichePatient.php à laquelle on ajoutera l'id du patient sélectionné :

```
if (\Promed\Authentication\Authentication::isLoggedIn()) {
    if (isset($_SESSION["role"]) && $_SESSION["role"] == "praticien") {
        $identiteDao = new DAO\Identite\IdentiteDAO();

        $identites = $identiteDao->readAllPatients();

        if (isset($_POST['submit'])) {
            $selectedId = $_POST['id_identite'];
            header("Location: ?action=fiche-patient&id=$selectedId");
            // empêche l'exécution d'autres instructions pour éviter erreur
            exit();
        }

        // appel du script de vue avec le titre associé
        $titre = "Recherche Patient";
        vueRecherche($titre, $identites);
    } else { // l'utilisateur est un patient, il n'a pas accès à cette page, il est donc redirigé
        $titre = "Mes rendez-vous";
        header('Location: ?action=rdv-patient');
    }
} else { // l'utilisateur n'est pas connecté, on affiche le formulaire de connexion
    $titre = "Authentification";
    header('Location: ?action=connexion');
}
```



## LISTE DES RENDEZ-VOUS

Dans « Mes rendez-vous », plusieurs actions sont possibles : le praticien peut filtrer la liste des rendez-vous selon la date, peut annuler un rendez-vous ou confirmer la demande d'annulation d'un client.

Le contrôleur Rdv.php va s'occuper de permettre tout ça. La partie suivante permet de récupérer les rendez-vous associés au praticien connecté. Elle fait appel à une méthode définie dans le métier Praticien.php.

```
// ici on récupère l'ID du praticien pour pouvoir afficher les rendez-vous du praticien connecté
$id_identite = $infoIdentite["id_identite"];
$idPraticien = $praticienDAO->readByIdIdentite($id_identite)->getId();
$rdvs = \Promed\Praticien\Praticien::getRdvPraticien($idPraticien);
```

On peut filtrer l'affichage en fonction de la date grâce à cette méthode :

```
if (isset($_POST['submit'])) {
    $dateSelect = $_POST['date'];
} else {
    $dateSelect = date('Y-m-d');
}
```

Et on peut maintenir ou annuler un Rdv avec ces méthodes :

```
// permet d'annuler un rdv
if (isset($_POST['annuler'])) {
    $id = $_POST['id'];
    // on appelle la méthode update
    $rdv = $RdvDAO->read($id);
    $rdv->setStatut("annulé");
    $RdvDAO->update($rdv);
    echo "<script>alert('Le rendez-vous a bien été annulé.');
```

## AUTHENTIFICATION

Le contrôleur Authentication.php permet de récupérer les données envoyées depuis le formulaire et d'appeler les méthodes métiers (Authentication) correspondantes :

```
// recuperation des donnees POST
if (isset($_POST["mail"]) && isset($_POST["mdp"])) {
    $mail = $_POST["mail"];
    $mdp = $_POST["mdp"];
    // connexion
    \Promed\Authentication\Authentication::login($mail, $mdp);
} else {
    $mail = null;
    $mdp = null;
}

if (\Promed\Authentication\Authentication::isLoggedIn()) { // si l'utilisateur est connecté
    //if role = praticien :
    if (isset($_SESSION["role"])) {
        $role = $_SESSION["role"];
        if ($role == "praticien") {
            header("Location: ?action=rdv-praticien");
        } else {
            header("Location: ?action=rdv-patient");
        }
    }
} else { // l'utilisateur n'est pas connecté, on affiche le formulaire de connexion
    // appel du script de vue avec le titre associé
    $titre = "Authentication";
    vueAuthentication($titre);
}
```

## INSCRIPTION

Le contrôleur Inscription.php permet de récupérer les données envoyées depuis le formulaire et d'appeler les méthodes métiers (Inscription) correspondantes :

```
// recuperation des donnees POST
if (isset($_POST["nom"]) && isset($_POST["prenom"]) && isset($_POST["tel"]) && isset($_POST["email"]) && isset($_POST["num"]) && isset($_POST["rue"]) && isset($_POST["cp"]) && isset($_POST["ville"]) && isset($_POST["specialite"]) && isset($_POST["description"]) && isset($_POST["mdp"])) {
    $nom = $_POST["nom"];
    $prenom = $_POST["prenom"];
    $tel = $_POST["tel"];
    $email = $_POST["email"];
    $num = $_POST["num"];
    $rue = $_POST["rue"];
    $cp = $_POST["cp"];
    $ville = $_POST["ville"];
    $specialite = $_POST["specialite"];
    $description = $_POST["description"];
    $mdp = $_POST["mdp"];
    // inscription
    \Promed\Inscription\Inscription::inscriptionPraticien($nom, $prenom, $tel, $email, $num, $rue, $cp, $ville, $specialite, $description, $mdp);
} else {
    $nom = null;
    $prenom = null;
    $tel = null;
    $email = null;
    $num = null;
    $rue = null;
    $cp = null;
    $ville = null;
    $specialite = null;
    $description = null;
    $mdp = null;
}
```

Inscription.php va fonctionner de paire avec des fichiers Ajax qui vont permettre la validation des données proposées par l'utilisateur.

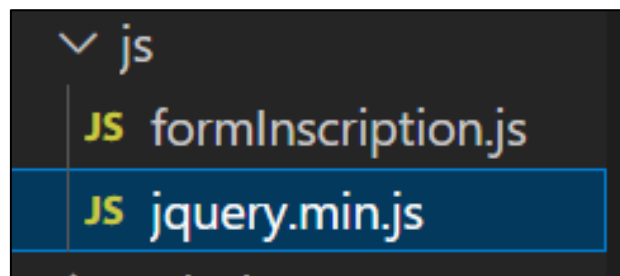
## DÉCONNEXION

Déconnexion va appeler la méthode logout du métier Authentification :

```
contrôleur > Deconnexion.php > ...
1  <?php
2
3  /**
4   * Contrôleur secondaire : deconnexion
5   */
6
7  if ($_SERVER["SCRIPT_FILENAME"] == __FILE__) {
8      // Un MVC utilise uniquement ses requêtes depuis le contrôleur principal : index.php
9      die('Erreur : ' . basename(__FILE__));
10 }
11
12 // On require_once les dependances nécessaires
13 array_map(function ($dependances) {
14     require_once $dependances;
15 }, DECONNEXION);
16
17 // traitement si necessaire des donnees recuperees
18 \Promed\Authentification\Authentification::logout();
19
```

## VALIDATION DE FORMULAIRE AVEC AJAX

Pour permettre la validation ou non des formulaires selon des règles de format mail ou mot de passe, on utilise un script js. Dans le dossier js on trouve deux fichiers : la bibliothèque JQuery qui va permettre l'utilisation d'une syntaxe simplifiée de JavaScript, et un fichier formInscription.js qui contient les méthodes pour valider ou non les formulaires.



Le fichier formInscription.js pose les conditions d'un format mail et d'un format mot-de-passe valide :

```
function validateEmail(email) {  
  let pattern =  
    /^[a-zA-Z0-9_.+-]+\@((([a-zA-Z0-9-])+\.)+([a-zA-Z0-9]{2,4})+)$/;  
  return pattern.test(email);  
}  
  
function validateMdp(mdp) {  
  var regex =  
    /^(?=.*[A-Z])(?=.*[a-z])(?=.*[0-9])(?=.*[!@#$%^&*])(?!.*[<>'"\\]).{8,}$/;  
  return regex.test(mdp);  
}
```

Si les formats sont valides, les données du formulaire pourront être récupérées.