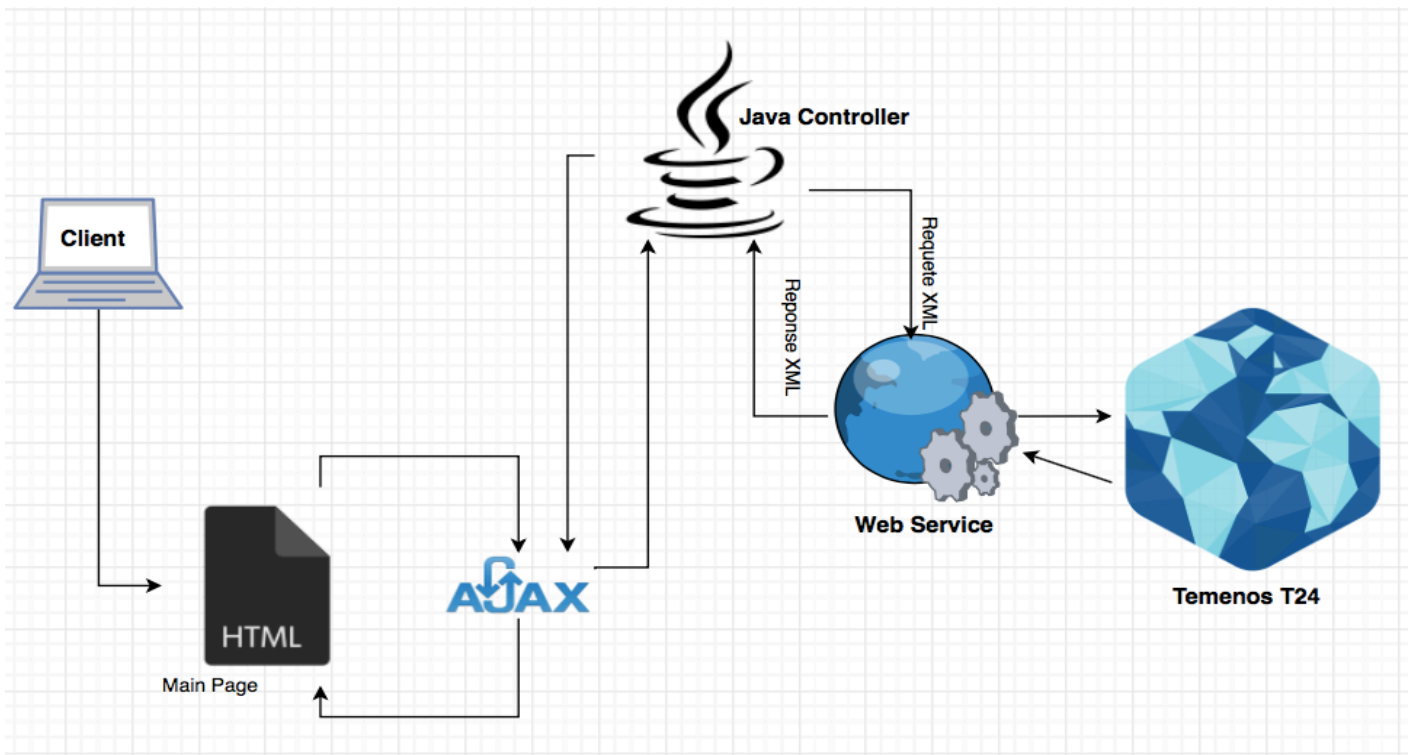


Made By:

ABDELILAH OUCHANI

Démonstration pour `github/Asesa/WebServiceApp`



L'utilisateur visite la page d'accueil, saisit les données d'authentification d'entrée, (Abdel,123):

T24

LOG IN

LOGIN

Connectez-Vous

Username

Abdel

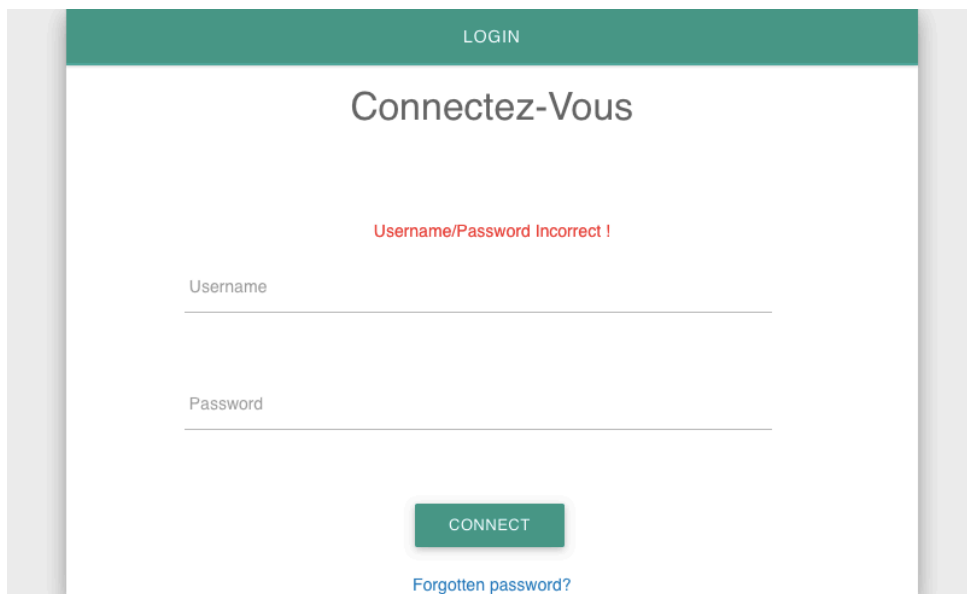
Password

...| 👁

CONNECT

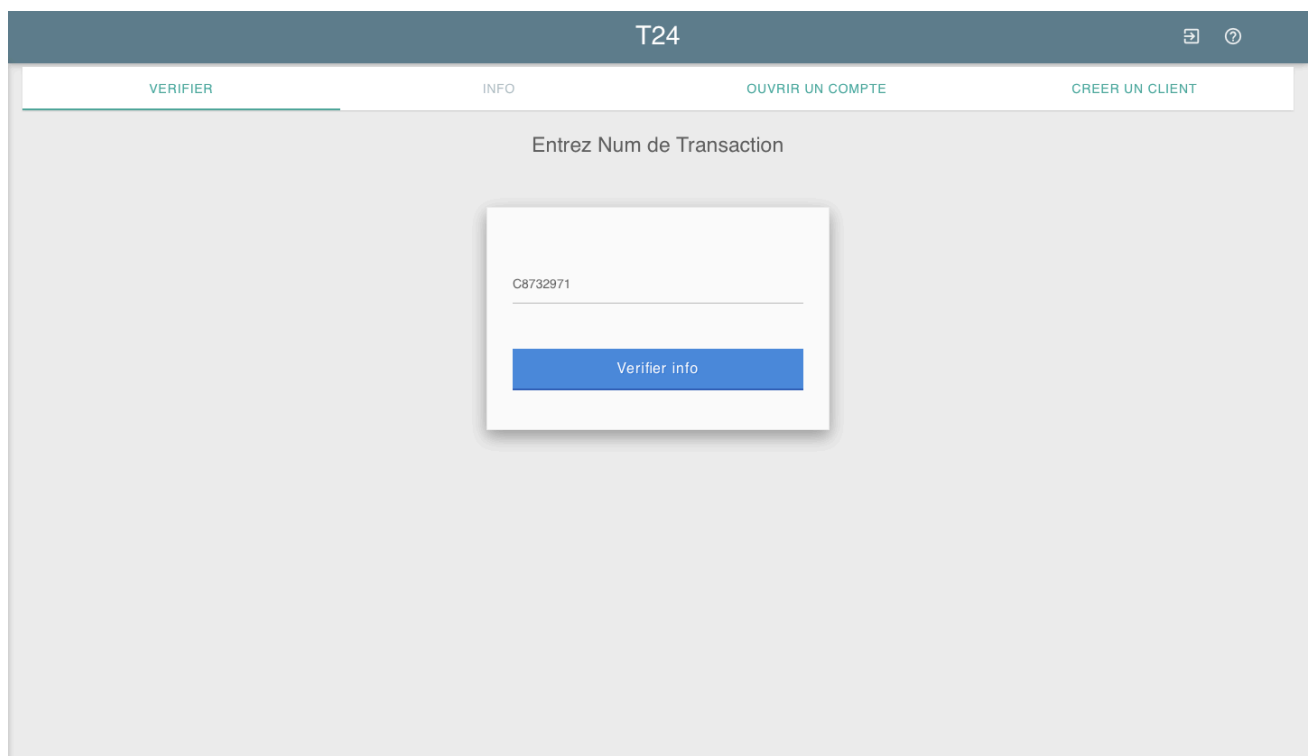
[Forgotten password?](#)

Après la saisie des données, si les informations sont incorrectes, on est présenté avec ceci :



A screenshot of a login page. At the top, there is a green header bar with the word "LOGIN" in white. Below this, the main heading "Connectez-Vous" is centered. A red error message "Username/Password Incorrect !" is displayed. Underneath, there are two input fields labeled "Username" and "Password". A green "CONNECT" button is positioned below the fields. At the bottom, there is a blue link that says "Forgotten password?".

Une seule Métrique (Transaction Id) doit être entrée pour avoir des informations sur un compte donné :




A screenshot of the T24 application interface. The top header is dark blue with "T24" in white. To the right of the header are two icons: a square and a circle. Below the header is a navigation bar with four tabs: "VERIFIER" (highlighted), "INFO", "OUVRIR UN COMPTE", and "CREER UN CLIENT". The main content area has a light gray background. At the top of this area, it says "Entrez Num de Transaction". In the center, there is a white box with a text input field containing "C8732971" and a blue button labeled "Verifier info".

Puis on clique sur "Vérifier Info" le traitement commence immédiatement, on est redirigé vers une autre tabulation qui va présenter tous les informations, mais ce qui passe en réalité c'est tout un appel vers une la contrôleur, puis lorsqu'on est redirigé, on récupère tous les informations de Webservice dans la nouvelle tabulation (Info), et on l'est gère au côté client pour bien afficher les informations en utilisant MaterializeJS et JSTL.

Dans un cas du Succès, On voit dans cette page :

Des informations sur le compte appelé, on a décidé d'afficher le Short Name, et le Gender(sex), on peut toujours afficher plus d'information.



INFO


OUVRIR UN COMPTE

Status is: SUCCESS

Id	Short Name	Country	Gender
100000244	XXXXX XXXXX	MA	Femme

Dans un cas d'Erreur, On voit dans cette page :

L'erreur concerné,



INFO

OUVRIR UN COMPTE

Error: TOO MANY MNEMONIC CHAR.

2) Dans la figure ci-dessous, pour créer un nouveau client, il faut remplir cette forme avec tous les informations nécessaires, puis on clique sur le bouton vert "Créer", une requête Ajax est performée vers le contrôleur. Ce dernier so i affirme les données comme correctes ou non, dans les deux cas un code JavaScript est exécuté pour lancer un "Alert" avec l'affirmation du contrôleur.

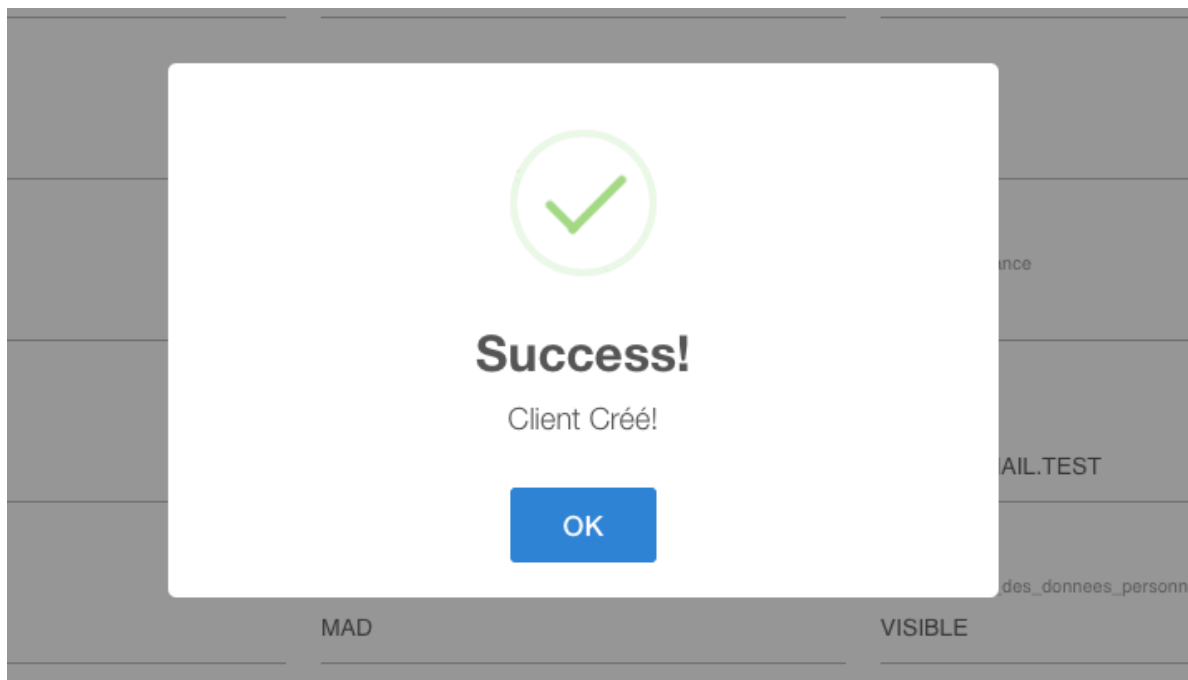
The form is titled "CREER UN CLIENT" and contains the following input fields:

Nom Usuel	Adresse 1	Complement Address
Ville	Code Postal	pays_de_residence
Pays	Agent Economique	Gestionnaire Principal
Pays Nationalite	Identifiant Marche Principal	

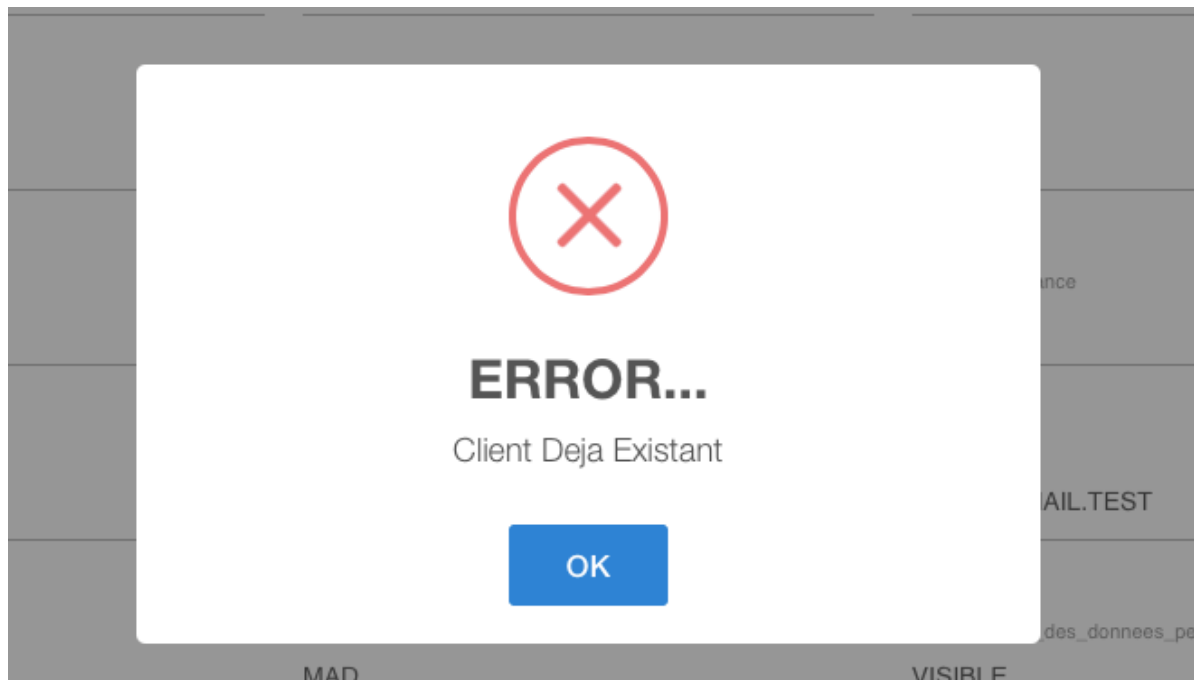
Puis Sur



Dans le cas du succès :



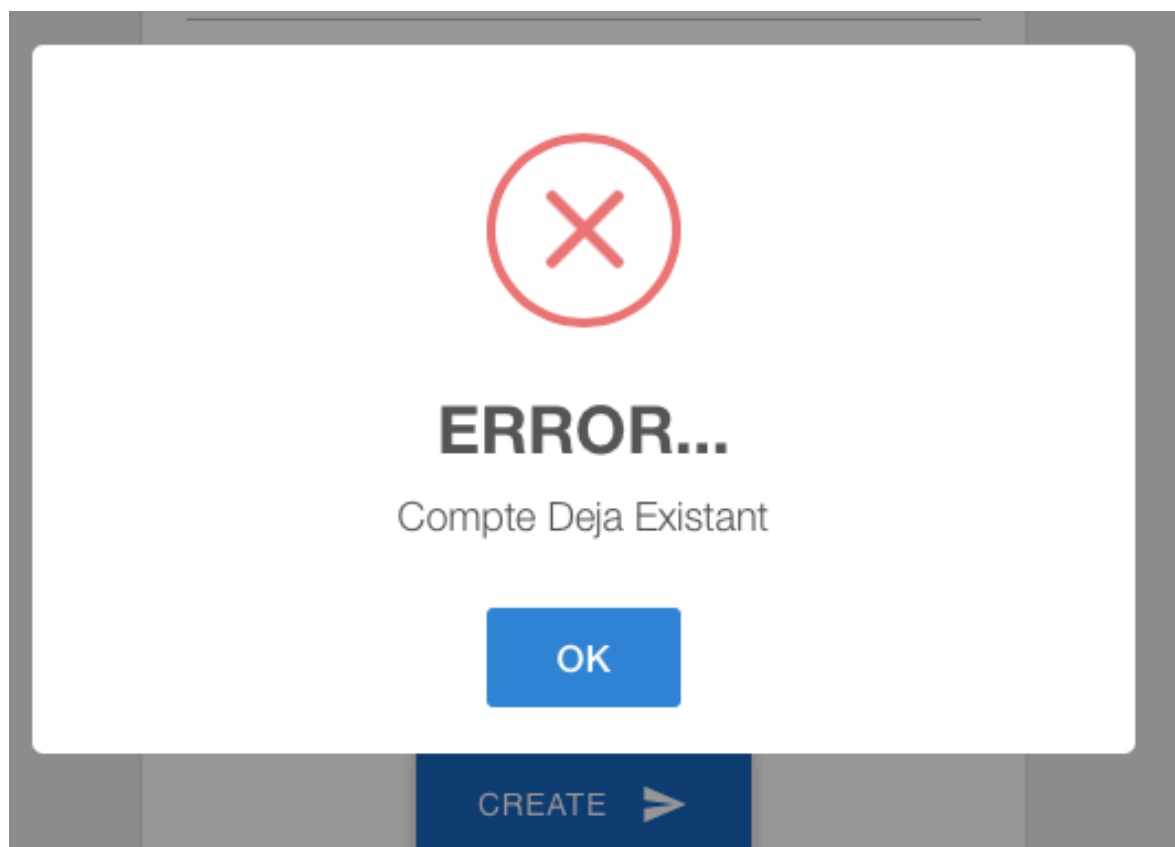
Dans un cas D'erreur : d



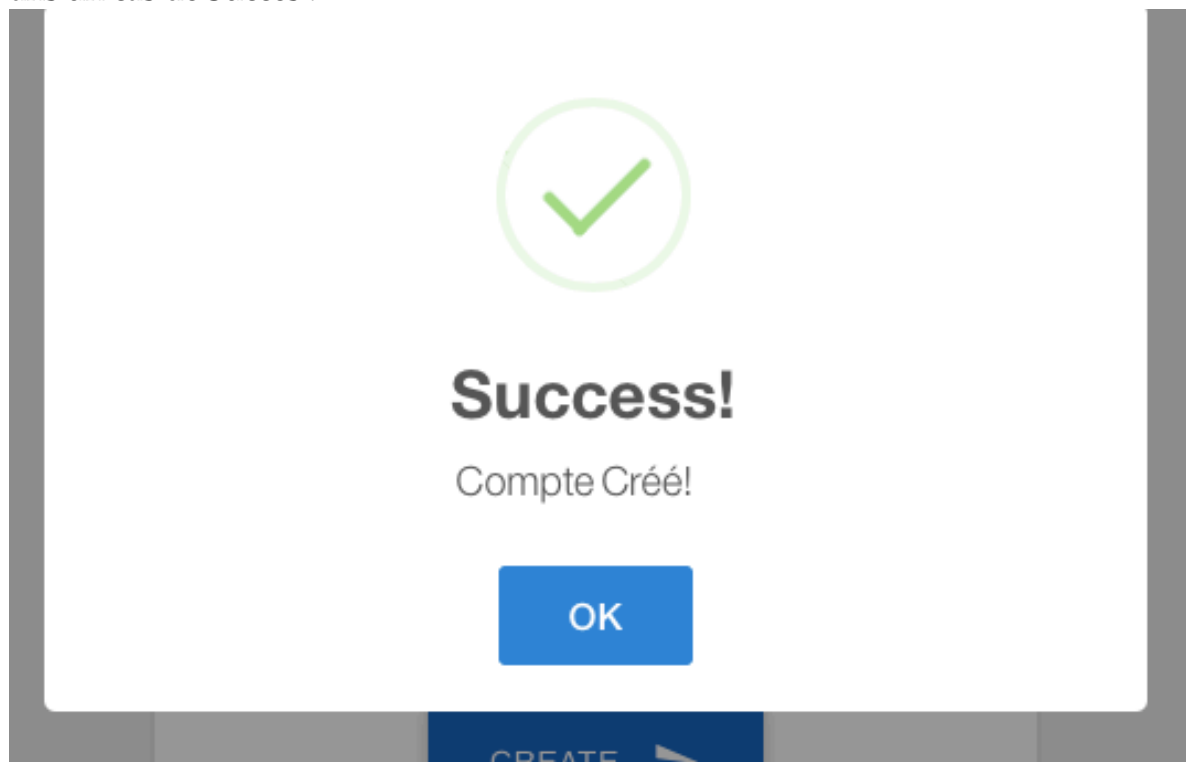
3) Dans la figure ci-dessous, pour ouvrir un nouveau compte, il faut remplir cette forme avec tous les informations nécessaires, puis on clique sur le bouton bleu "Create" , une requête Ajax est performée vers le contrôleur. Ce dernier soi affirme les données comme correctes ou non, dans les deux cas un code JavaScript est exécuté pour lancer un "Alert" avec l'affirmation du contrôleur.

A screenshot of a web application showing a form titled 'OUVRIR UN COMPTE'. The form has five input fields: 'Client', 'Categorie' (with a dropdown arrow), 'Intitule du Compte', 'Devise' (with a dropdown arrow showing 'MAD'), and 'LPLURALACC'. At the bottom of the form is a blue button labeled 'CREATE' with a right-pointing arrow.

Dans un cas d'Erreur :



Dans un cas de Succes :



Code-Source en détail (explication)

Notre code est détaillé comme suit :

1. Une page HTML d'accueil avec un nom et mot de passe pour authentifier. Cette page utilise la fonctionnalité Spring_security_login pour fermer notre Site Web, d'une façon assez sécurise.
2. Une page HTML "Main Page". Cette page contient 4, "Vérifier, Info, Créer un client, Ouvrir un compte", chaque tab est censé d'exécuter certaines commandes, Le code suivant présente la partie HTML, en utilisant La Librairie **Materialize.JS** on a pu concevoir un site web attirant, plus simple, le code suivant contient des inputs pour les métriques, le code est plus long, mais le reste c'est seulement des scripts et des tags nécessaires pour le design + le reste des inputs.

Cette page, conçu avec MaterializeJS, et à l'aide du librairie SweetAlert, JQuery , on peut visualiser les information initialement récupérées.

```
</div>
</div>
<div id="create" class="col s12">
  <div class="card">
    <div class="card-content">
      <span class="card-title center">Creer Un Client</span>
      <form action="#" class="col s12" id="create_form">
        <div class="row">
          <div class="input-field col s12 m4">
            <input name="name" required type="text" value="MEHDI01" class="validate">
            <label for="name">First Name</label>
          </div>
          <div class="input-field col s12 m4">
            <input name="company" required type="text" value="MA0011001" class="validate">
            <label for="company">Company</label>
          </div>
          <div class="input-field col s12 m4">
            <input name="password" id='password' required type="password" value="a123*123" class="validate">
            <label for="password">Password</label>
          </div>
        </div>
        <div class="row">
          <div class="input-field col s12 m4">
            <input name="nom_usuel" type="text" required value="MR TEST TEST" class="validate">
            <label for="nom_usuel">Nom Usuel</label>
          </div>
          <div class="input-field col s12 m4">
            <input name="adresse1" type="text" required value="AAAA" class="validate">
            <label for="adresse1">Adresse 1</label>
          </div>
        </div>
      </form>
    </div>
  </div>
</div>
```

Des fichiers JS sont accompagnées : SweetAlertJS, MaterializeJS, JQuery, JQuery Formatter...

Chaque fichier a un travail donné. Généralement de design ou manipulation du DOM.

Notre architecture de code contient 3 type de package :

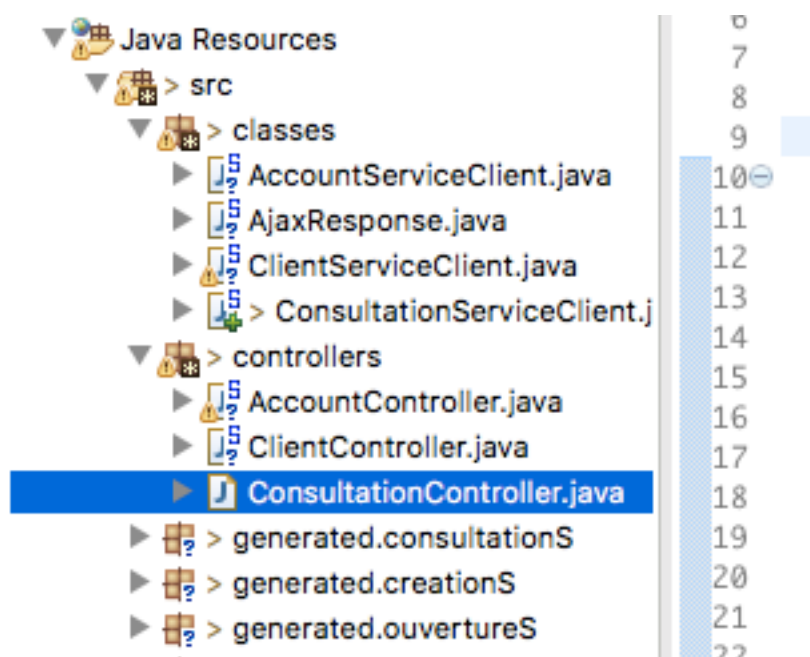
1) Le 1er package contient des modèles de communication avec les WebServices sous forme de 3 classe.

AccountServiceClient.java ==> Model Ouverture de Compte
ConsultationController.java ==> Model Consultation de Compte
ClientController.java ==> Model Création de Client

2) Le 2eme package contient 3 contrôleurs, chaque contrôleur pour une fonctionnalité donnée : vérification, création de client ou ouverture de compte.

AccountController.java ==> Contrôleur Ouverture de Compte
ConsultationController.java ==> Contrôleur Consultation de Compte
ClientController.java ==> Contrôleur Création de Client

3) 3 packages restants ont été générés par JAXB d'après les schémas XSD qui se trouvent dans les fichiers WSDL accompagnant (generated. **).



On prendra par exemple la fonctionnalité **d'ouvrir un compte** : Pour le Model de class AccountServiceClient (création de client) on va enregistrer les informations récupérées depuis T24 à travers le WebService dans un objet réponse de type BCPDIGOUVCPTTEPPHResponse, la collection JAXBElement c'est pour "wrapper" le résultat et faciliter la lecture de cet objet réponse.

```

public class AccountServiceClient extends WebServiceGatewaySupport {
    BCPDIGOUVCPTTEPPH request = null;
    JAXBElement<BCPDIGOUVCPTTEPPHResponse> response = null;
    @SuppressWarnings("unchecked")
    public JAXBElement<BCPDIGOUVCPTTEPPHResponse> getResponse() {
        try {
            response = (JAXBElement<BCPDIGOUVCPTTEPPHResponse>) getWebServiceTemplate().marshalSendAndReceive(request);
        } catch (Exception e) {
            e.printStackTrace();
            response = null;
        }

        return response;
    }

    public BCPDIGOUVCPTTEPPH getRequest() {
        return request;
    }

    public void setRequest(BCPDIGOUVCPTTEPPH request) {
        this.request = request;
    }
}

```

creation d'objet de requete BCPDIGOUVCPTTEPPH

Request Code

Cette **réponse** est récupérée dans le contrôleur AccountController, puis renvoyé vers le Navigateur à l'aide d'Ajax pour visualiser le résultat.

```

sc.setRequest(request):
response = sc.getResponse();
if(response!=null){
    message = response.getValue().getStatus().getSuccess
    successIndicator = response.getValue().getStatus().g
}
System.out.println(message+"--"+successIndicator);
return message+"--"+successIndicator;
}

```

Un bout de code dans la figure ci-dessous qui clarifie une requête AJAX, qui nous permet d'envoyer une requête asynchrone vers notre contrôleur avec un Mapping virtuel 'account/' qui envoie elle-même les requêtes vers une Webservice à travers la classe AccountServiceClient.java.

```
function sendOpen() {  
    var data = {}  
    $("#account_form input").each(function(){  
        data[$(this).attr("name")]=$(this).val();  
    });  
    $.ajax({  
        type : "post",  
        data: data,  
        url : "account/",  
        timeout : 100000,  
        success : function(data) {  
            $(".progress").hide();  
            if (data.indexOf("T_24_ERROR") >= 0){  
                if (data.indexOf("OVERRIDE") >= 0){  
                    swal('ERROR...', 'Compte Deja Existant', 'error')  
                }  
            }  
            else{  
                swal('ERROR...', data, 'error')  
            }  
        }  
        if (data.indexOf("SUCCESS") >= 0){  
            swal('Success!', 'Compte Créé!', 'success')  
        }  
    })  
}
```

Après recevoir une réponse, une librairie SweetAlert nous aide à afficher un 'alert' qui visualise le résultat de notre réponse, Succès ou Erreur, il faut noter que tout le travail est exécuté au côté serveur dans le contrôleur AccountController

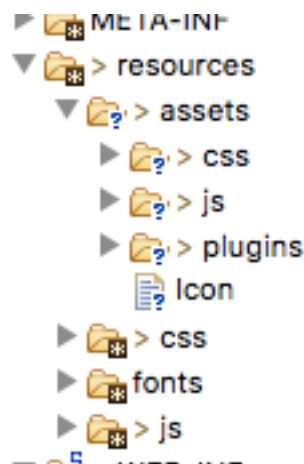
Dans la partie suivante, on va 'se plonger' plus profondément, on parlera d'abord du raw-servlet.xml, qui est la "ServletDispatcher" qui nous aide à définir tous les beans, comme InternalViewResolver le responsable du Mapping MVC des pages JSP, et Marshaller qui est le responsable de la sérialisation des objets Java depuis les éléments définis dans la XSD du fichier XML. :

```

<bean class="org.springframework.oxm.jaxb.Jaxb2Marshaller" id="marshallerCreation">
  <property name="contextPath" value="generated.creationS"></property>
</bean>
<bean class="classes.ClientServiceClient" id="creationServiceClientBean">
  <property name="defaultUri" value="https://29010srvwas7:9444/DIGCLISNG/services?wsdl" />
  <property name="marshaller" ref="marshallerCreation" />
  <property name="unmarshaller" ref="marshallerCreation" />
</bean>

```

1. On a créé plusieurs Bean, chacune pour une fonctionnalité donnée, nous observons 2 beans relatives au Marshalling et UnMarshalling, le travail est répété 2 fois pour les 2 autres WebService de consultation et création de compte.
2. Tous les ressources, image, Css, Js sont disponible dans le dossier Resources :



Spring Security:

Pour la Sécurisation de notre application on a utilisé Spring Security pour l'authentification et la décision des droits de visites des pages :

```

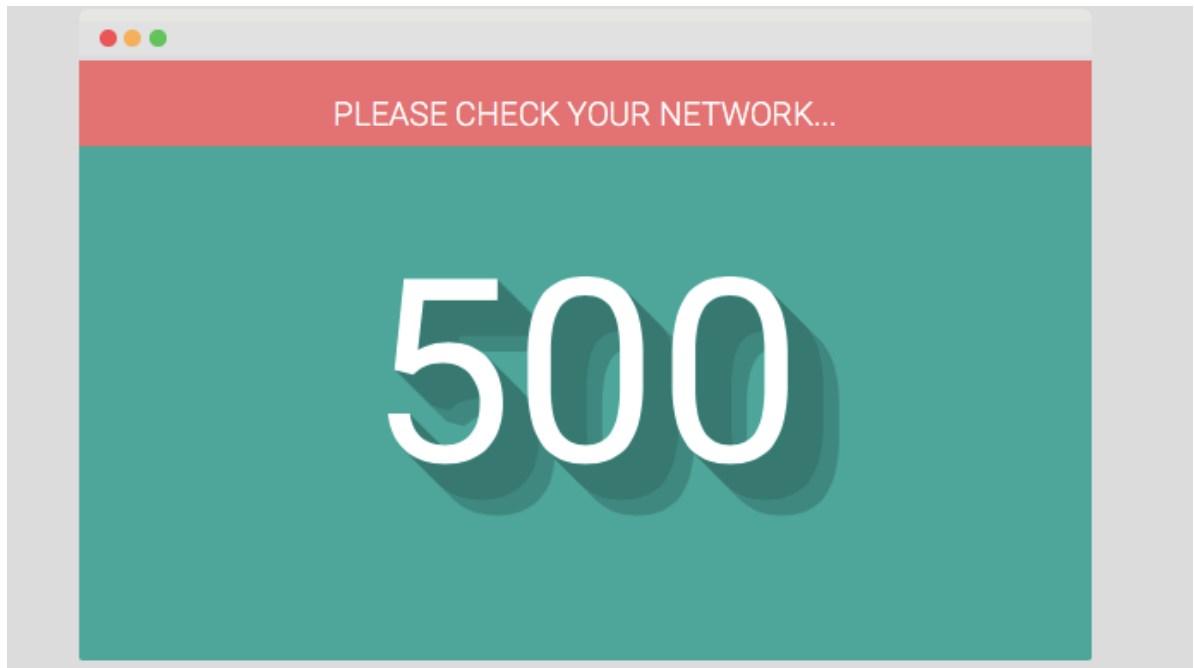
<security:form-login login-page="/login"
  default-target-url="/"
  username-parameter="username"
  password-parameter="password"
  authentication-failure-url="/login?error=failed"
  login-processing-url="/j_spring_security_check" />
<security:logout logout-url="/logout" logout-success-url="/" />
<security:csrf/>

</security:http>

<security:authentication-manager>
  <security:authentication-provider>
    <security:user-service>
      <security:user name="abdel" password="123" authorities="ROLE_USER" />
    </security:user-service>
  </security:authentication-provider>
</security:authentication-manager>

```

Et Si une erreur de connexion au réseau de l'environnement, on est présente avec une erreur de type Internal Server Error 500, Check Your Network !



Bibliographie

- [1] SOAP: ,
Available at : <https://www.w3.org/TR/soap/>

- [2] Materialize, Google's own design language's reference and all CSS, JS related Files
Available at : <http://materializecss.com>

- [3] Diagrams : Diagram drawn with draw.io tool,
Available at : <http://draw.io>

- [4] Spring documentation for :
Available at: <https://spring.io>

- [5] All of the above, are official ressources that were used in the process of making this report as well as the web application itself, and when a problem was encountered, a basic Google search would usually solve the issue.