RAPPORT DE STAGE

Ines Zaidou

Table des matières

INTRODUCTION

Dans le cadre de ma deuxième année de BTS Services Informatiques aux Organisations (SIO), option Solutions Logicielles et Applications Métier (SLAM), j'ai effectué un stage au sein de **Mauriskills Software Ltd**, une entreprise spécialisée dans le développement d'applications full-stack. Ce stage, d'une durée de 6 semaines, m'a permis d'acquérir des compétences pratiques en développement backend et en intégration de bases de données, en mettant en application les connaissances théoriques acquises au cours de ma formation.

Mauriskills Software Ltd est une société de développement logiciel qui se concentre sur la conception d'applications complètes, intégrant à la fois le frontend et le backend. Durant mon stage, j'ai principalement travaillé sur la partie backend, en développant des fonctionnalités en **PHP** et en intégrant des bases de données **MySQL**.

L'une des principales missions qui m'a été confiée concernait le projet de **digitalisation des processus métier de Proxy Brokers Ltd**, un client de Mauriskills Software Ltd spécialisé dans le courtage en assurance. Ce projet visait à informatiser l'enregistrement des clients, des compagnies d'assurance, ainsi que la gestion des classes d'assurance et des renouvellements.

Ce rapport retrace mon expérience au sein de l'entreprise **Mauriskills Software Ltd**. Il commence par une présentation de l'entreprise d'accueil et de l'entreprise cliente, suivie d'une description détaillée du travail réalisé et des outils utilisés. Ensuite, il expose le processus de développement et d'intégration des API avant de conclure sur les enseignements tirés de cette expérience.

PRESENTATION DE MAURISKILLS SOFTWARE LTD

1.Présentation générale de l'entreprise

Mauriskills Software Ltd est une entreprise spécialisée dans le développement d'applications full-stack. Située à **Quatre-Bornes, Maurice**, elle propose des solutions logicielles complètes en combinant le développement frontend (interface utilisateur) et backend (gestion des données et logique métier). L'entreprise met en avant son expertise dans la conception et l'intégration d'applications performantes et adaptées aux besoins de ses clients.

2.Activités et domaines d'expertise

L'entreprise se concentre sur le **développement d'applications web et logicielles** en utilisant différentes technologies modernes. Son expertise couvre :

- **Le développement full-stack** : création d'applications intégrant aussi bien le frontend que le backend.
- **L'intégration de bases de données** : utilisation de **MySQL** pour stocker et gérer efficacement les données des applications.

3. Objectifs et mission

L'objectif principal de Mauriskills Software Ltd est de fournir à ses clients des solutions logicielles optimisées et adaptées à leurs besoins spécifiques. L'entreprise travaille avec divers clients, dont **Proxy Brokers Ltd**, afin de moderniser leurs processus en développant des outils digitaux sur mesure.

PRESENTATION DE PROXY BROKERS LTD

1. Présentation générale de l'entreprise

Proxy Brokers Ltd est une entreprise spécialisée dans le **courtage en assurance et le conseil**. Elle opère principalement à **Maurice**, offrant des services personnalisés à ses clients. Son objectif est de proposer des **solutions d'assurance sur mesure**, en s'appuyant sur l'expertise et le professionnalisme de son équipe. Ils se spécialisent dans divers domaines de l'assurance, en se concentrant sur la gestion des risques et les services de responsabilités. De plus, ils s'efforcent d'être des conseillers en assurance engagés et compétents qui font la différence grâce à leur approche unique et à leur personnel expérimenté.

2.Les besoins de Proxy

Dans le cadre de sa transformation numérique, Proxy Brokers Ltd a fait appel à **Mauriskills Software Ltd** pour développer un système permettant de **digitaliser et centraliser l'ensemble de ses processus métier**. L'objectif est de moderniser la gestion des clients et des contrats d'assurance en automatisant certaines tâches essentielles :

- **L'enregistrement et la gestion des clients** pour un suivi optimal des dossiers d'assurance.
- **La gestion des relations avec les compagnies d'assurance**, incluant l'enregistrement des assureurs et de leurs contacts.
- **La définition des classes d'assurance** et des couvertures associées, adaptées aux besoins spécifiques des clients.
- **Le suivi des renouvellements et des périodes de bordereau** afin d'assurer une continuité des contrats d'assurance.
- **La gestion des placements d'assurance des clients**, garantissant une administration efficace et sécurisée des contrats.

OBJECTIF DU STAGE

L'objectif principal de mon stage chez **Mauriskills Software Ltd** était de **développer mes compétences en développement backend et en conception d'API**, tout en contribuant à un projet concret de digitalisation pour **Proxy Brokers Ltd**. Ce stage m'a permis de mettre en pratique les notions acquises en cours et d'approfondir mes connaissances en **PHP, MySQL et intégration d'API**.

Plus précisément, mes objectifs étaient les suivants :

1. **Approfondir mes compétences en développement backend**
    - Maîtriser la programmation en **PHP** pour le développement de fonctionnalités serveur.
    - Structurer et organiser du code backend pour une application professionnelle.
2. **Développer des API pour l'application**
    - Concevoir et implémenter des **API** permettant la communication entre le frontend et le backend.
    - Travailler avec les méthodes **GET, POST, PUT, DELETE** pour la gestion des ressources.
3. **Gérer l'intégration des bases de données**
    - Concevoir et manipuler des bases de données **MySQL** pour stocker et gérer les informations des clients, assureurs et contrats d'assurance.
    - Optimiser les requêtes SQL pour améliorer la performance du système.
4. **Participer au développement d'une application métier**
    - Contribuer à la digitalisation des processus de **Proxy Brokers Ltd**, en travaillant sur la gestion des clients, des assureurs et des contrats d'assurance.
    - Assurer la bonne intégration des données et des fonctionnalités backend dans l'application.

Grâce à ce stage, j'ai pu **renforcer mes compétences en développement backend et en conception d'API**, tout en découvrant les enjeux de la digitalisation d'un système de gestion d'assurance.

TRAVAIL REALISE

1. Planification du stage

Pour mener à bien mon stage chez **Mauriskills Software Ltd**, une planification a été mise en place dès mon intégration dans l'entreprise. Les premières semaines ont été consacrées à la prise en main des outils de développement et à la compréhension des besoins du projet de digitalisation pour **Proxy Brokers Ltd**. Par la suite, les tâches ont été organisées en fonction des priorités du projet et des délais impartis.

2. Rôle et responsabilités

Pendant mon stage, j'ai occupé le rôle de développeur backend, avec des responsabilités liées principalement à la création et à la gestion des **API REST** et à l'intégration des bases de données. Mes principales responsabilités incluaient :

- Conception et développement d'**API REST** pour assurer la communication entre le frontend et le backend.
- Intégration et gestion des bases de données **MySQL**.
- Implémentation des fonctionnalités nécessaires à la digitalisation des processus de Proxy Brokers Ltd.

3. Missions effectuées

J'ai réalisé plusieurs missions techniques et organisationnelles, notamment :

Développement des API et intégration Backend

- Création des **endpoints REST** en PHP pour la gestion des clients, des assureurs et des contrats d'assurance.
- Implémentation des requêtes **GET, POST, ADD, UPDATE, DELETE** pour manipuler les données de l'application.

Gestion et optimisation de la base de données

- Conception des schémas relationnels dans **MySQL** pour stocker efficacement les données.
- Rédaction de requêtes SQL optimisées pour garantir la performance du système.
- Liaison entre les API et la base de données pour assurer la cohérence des informations.

Digitalisation des processus métier de Proxy Brokers Ltd

- Développement des fonctionnalités de **gestion des renouvellements et des placements d'assurance**.
- Automatisation des mises à jour des contrats et des notifications de renouvellement.
- Optimisation des workflows métiers en intégrant des outils numériques adaptés.

4. Mon rôle au sein de l'entreprise

Lors de mon stage chez Mauriskills Software Ltd, j'ai été intégré à l'équipe de développement backend. Mon travail consistait à :

- Développer des fonctionnalités en **PHP** pour la gestion des données et la communication entre les différents composants de l'application.
- Travailler avec **MySQL** pour stocker et organiser les informations des clients.

5. Mon implication dans le projet

En tant que stagiaire chez **Mauriskills Software Ltd**, j'ai contribué au développement de ce projet en travaillant sur la **partie backend** de l'application. Mon travail a consisté à :

- Concevoir et implémenter des **fonctions en PHP** pour gérer les enregistrements de clients, assureurs et contrats.
- Développer des **intégrations avec MySQL** pour assurer un stockage et un accès optimisés aux données.
- Travailler sur l'amélioration de la **gestion des périodes de renouvellement et des placements d'assurance**.

OUTILS UTILISÉS

J'ai utilisé plusieurs outils et technologies pour mener à bien le développement des API et la gestion des bases de données. Ces outils m'ont permis d'optimiser mon travail et d'assurer une meilleure organisation du projet.

1. Environnement de Développement

- **Visual Studio Code** : Principal éditeur de code utilisé pour écrire les scripts PHP et configurer les API.

2. Technologies Backend

- **PHP (avec PDO pour MySQL)** : Langage principal pour le développement des API et la gestion des interactions avec la base de données.
- **MySQL** : Système de gestion de bases de données relationnelles utilisé pour stocker et organiser les informations des clients, assureurs et transactions.
- **WAMPP** : Serveur local utilisé pour exécuter les scripts PHP et tester les API en local.

3. Outils de Gestion de Base de Données

- **HeidiSQL** : Outil utilisé pour interagir avec la base de données MySQL, permettant d'exécuter des requêtes SQL, d'explorer les tables et de manipuler les données facilement.

4. Technologies Frontend et API

- **ReactJS** : Framework utilisé pour l'interface utilisateur, qui communique avec les API développées.
- **JSON** : Format de données utilisé pour l'échange d'informations entre le frontend et le backend.

DEVELOPPEMENT ET INTEGRATION DES API

Dans le cadre de mon stage, j'ai développé plusieurs **API REST** permettant d'interagir avec trois tables principales de la base de données :

1. **Placements** : Gère les placements d'assurance effectués par les clients.
2. **Transactions** : Suit les différentes transactions associées à un placement.
3. **InterestInsured** : Stocke les biens assurés pour chaque transaction.

Table Placements :

L'API repose sur **trois éléments essentiels** :

1. **Les données JSON**

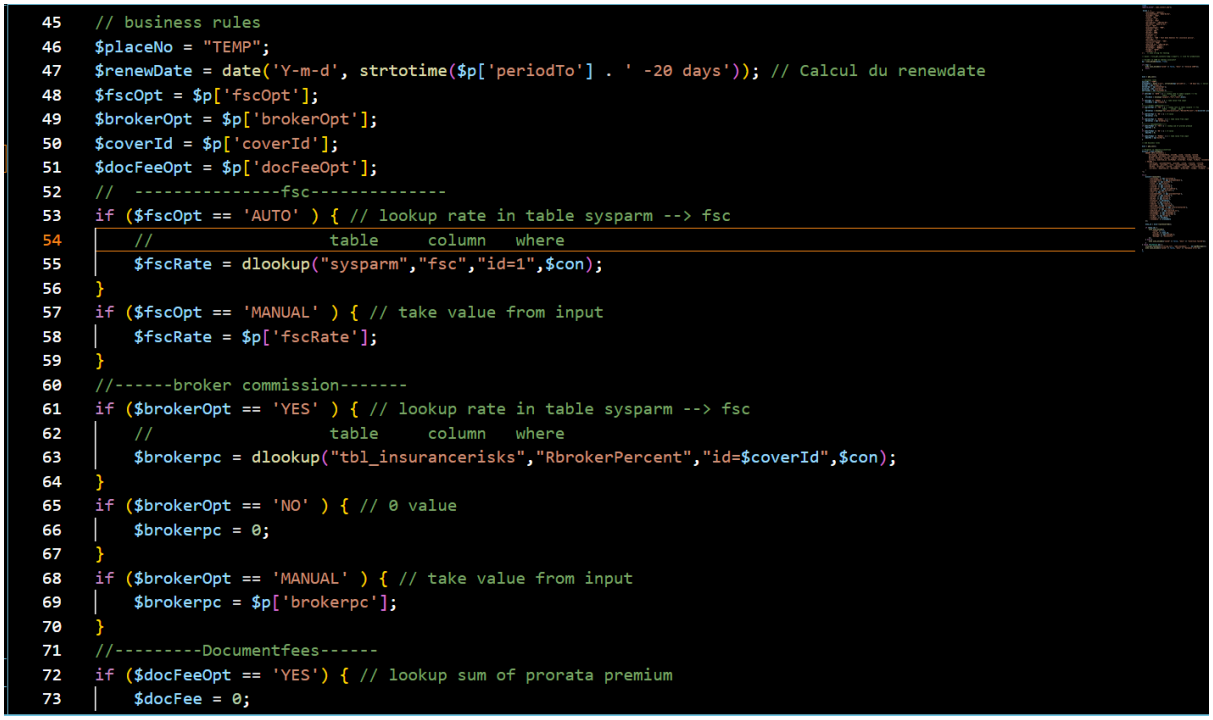
Le front-end **envoie une requête POST** avec les informations du placement sous format JSON.

```
$json = '{
   "entryType": "general",
   "placementDate": "2025-01-01",
   "placeNo": "XXX",
   "insId": "60",
   "classId": "10",
   "coverId": "574",
   "periodFrom": "2025-01-01",
   "periodTo": "2025-12-31",
   "stat": "WIP",
   "placementType": "NEW",
   "oneTime": true,
   "polFee": 500,
   "docFee": 3000,
   "brokerpc": 7,
   "userId": "1",
   "remarks": "ADD - Test data General for insurance policy",
   "policyNo": "",
   "insurerContactId": "404",
   "currency": "USD",
   "modified_on": "2024-10-25",
   "docFeeOpt": "MANUAL",
   "brokerOpt": "MANUAL",
   "fscOpt": "AUTO",
   "fscRate": 0
}';   // json string for testing

// $json = file_get_contents("php://input"); // live for productions
```

Pour les tests, nous utilisons une chaîne JSON statique définie dans le code. En production, cette ligne est remplacée par file_get_contents("php://input") pour récupérer les données envoyées par le front-end.

2. **Les Business Logic**



```php
// business rules
$placeNo = "TEMP";
$renewDate = date('Y-m-d', strtotime($p['periodTo'] . ' -20 days')); // Calcul du renewdate
$fscOpt = $p['fscOpt'];
$brokerOpt = $p['brokerOpt'];
$coverId = $p['coverId'];
$docFeeOpt = $p['docFeeOpt'];
//  ---------------fsc--------------
if ($fscOpt == 'AUTO' ) { // lookup rate in table sysparm --> fsc
    //                  table     column    where
    $fscRate = dlookup("sysparm","fsc","id=1",$con);
}
if ($fscOpt == 'MANUAL' ) { // take value from input
    $fscRate = $p['fscRate'];
}
//------broker commission-------
if ($brokerOpt == 'YES' ) { // lookup rate in table sysparm --> fsc
    //                  table     column    where
    $brokerpc = dlookup("tbl_insurancerisks","RbrokerPercent","id=$coverId",$con);
}
if ($brokerOpt == 'NO' ) { // 0 value
    $brokerpc = 0;
}
if ($brokerOpt == 'MANUAL' ) { // take value from input
    $brokerpc = $p['brokerpc'];
}
//---------Documentfees------
if ($docFeeOpt == 'YES') { // lookup sum of prorata premium
    $docFee = 0;
```

Les business logic définissent la logique de gestion appliquée aux données avant leur enregistrement ou leur traitement dans le système. Elles garantissent la cohérence et la validité des informations en appliquant des contrôles spécifiques

3. Les requêtes SQL

Elles utilisent les données validées pour interagir avec la base de données, que ce soit pour : insérer (INSERT), mettre à jour (UPDATE), supprimer (DELETE).

Ainsi, chaque requête SQL est construite à partir du JSON reçu et passe par la Business Logic avant d'être exécutée, garantissant cohérence et intégrité des données.

API développées :

- **Endpoint pour créer un placement** :

Le front-end envoie les données via une requête **POST** à l'API PHP (add_placement.php).

L'API **valide les données** (Business Logic) :

Si les validations sont OK, les données sont insérées dans la table placements.

Une réponse JSON est envoyée au front-end pour indiquer si l'opération a réussi.

- **Endpoint pour mettre à jour un placement**

L'utilisateur peut modifier un placement existant.

Une requête **PUT** est envoyée avec l'identifiant du placement et les nouvelles valeurs.

L'API effectue des **vérifications métiers** avant modification :

- Vérifier si le placement existe
- Vérifier si le statut autorise une modification

L’API met à jour les données dans la table placements et retourne une confirmation.

- **Endpoint pour supprimer un placement** :

Un placement peut être annulé si certaines conditions métiers sont respectées :

Un placement **déjà confirmé** ne peut pas être annulé.

Vérifier que **aucune transaction n’est en cours** sur ce placement.

Une requête **PUT** est envoyée avec l’identifiant du placement.

L’API **vérifie les contraintes métiers** et change le statut en **"CANCELLED"** si l'annulation est autorisée.

Une confirmation est envoyée au front-end.

Table Transactions :

Les transactions sont directement **liées aux placements** et permettent de suivre l'évolution financière d'un placement d'assurance. Chaque transaction est rattachée à un placement existant et contient des informations essentielles comme :

- **L’identifiant du placement** (placeId) pour assurer la liaison.
- **Le type de transaction** (exemple : NEW, PAYMENT, ADJUSTMENT).
- **Le montant total de la prime d'assurance**.
- **La date de la transaction et son statut** (WIP, CONFIRMED, CANCELLED).

L’API gérant les transactions fonctionne en interaction directe avec l’API des placements, garantissant que chaque transaction est bien associée à un placement valide.

Comme pour la table placements, l’API repose sur trois composants fondamentaux :

- **Les données JSON** : Représentent les informations envoyées par le front-end.
- **La Business Logic** : Vérifie la cohérence des données et applique des règles métiers spécifiques.
- **Les requêtes SQL** : Permettent d'exécuter les actions en base de données en fonction de l’API utilisée.

Chaque requête SQL est construite **en fonction du type d’API** utilisé :

- L’API **add_transaction** permet **d’ajouter une nouvelle transaction** en vérifiant d’abord **si le placement lié existe bien**, la business logic s'assure que le type et le statut de la transaction sont conformes aux règles établies.
- L’API **mod_transaction** requête permet **de modifier une transaction existante** tout en respectant certaines conditions métiers elle vérifie que la transaction existe avant de la modifier, s’assure que la modification est autorisée selon le statut (CONFIRMED, WIP).
- L’API **del_transaction** permet **de supprimer une transaction**, en vérifiant que la transaction existe avant de la supprimer et en s’assurant que la transaction n’a pas encore été confirmée avant de l’autoriser à être supprimée.

Table InterestInsured :

La table **InterestInsured** permet d'enregistrer les biens ou intérêts assurés liés à une **transaction spécifique**. Cette étape intervient **après la création d'une transaction**, et elle joue un rôle crucial dans le calcul des **primes d'assurance** ainsi que dans la **génération du devis proforma**.

Chaque bien assuré est rattaché à une **transaction existante** et contient des informations comme :

- **L'identifiant du placement (placeId)** auquel l'intérêt assuré est lié.
- **L'identifiant de la transaction (txId)** qui définit le contexte financier.
- **L'assureur (insId)** et le client concerné (custId).
- **Le capital assuré (SumInsured)** et le **taux d'assurance (rate)**.
- **Les montants calculés** : prime annuelle, prime prorata, frais de service (fsc), courtage (brokerFee).

L'API gérant **InterestInsured** fonctionne **en interaction directe avec l'API des transactions**, garantissant que chaque bien assuré est bien **lié à une transaction valide et modifiable**.

Comme pour les autres tables, l'API repose sur **trois composants fondamentaux** :

- **Les données JSON** : Elles contiennent les détails du bien assuré.
- **La Business Logic** : Elle applique les règles métiers et effectue les calculs automatiques avant insertion.
- **Les requêtes SQL** : Elles insèrent, mettent à jour ou suppriment les données dans la base de données.

Chaque requête SQL est **exécutée en fonction de l'API utilisée** :

- L'API **add interestinsured**, permet **d'ajouter un bien assuré** en s'assurant que la **transaction associée n'est pas encore confirmée**. La Business Logic effectue les **calculs automatiques** avant insertion :

add_interestinsured.php api\interestinsured\add_interestinsured.php\...

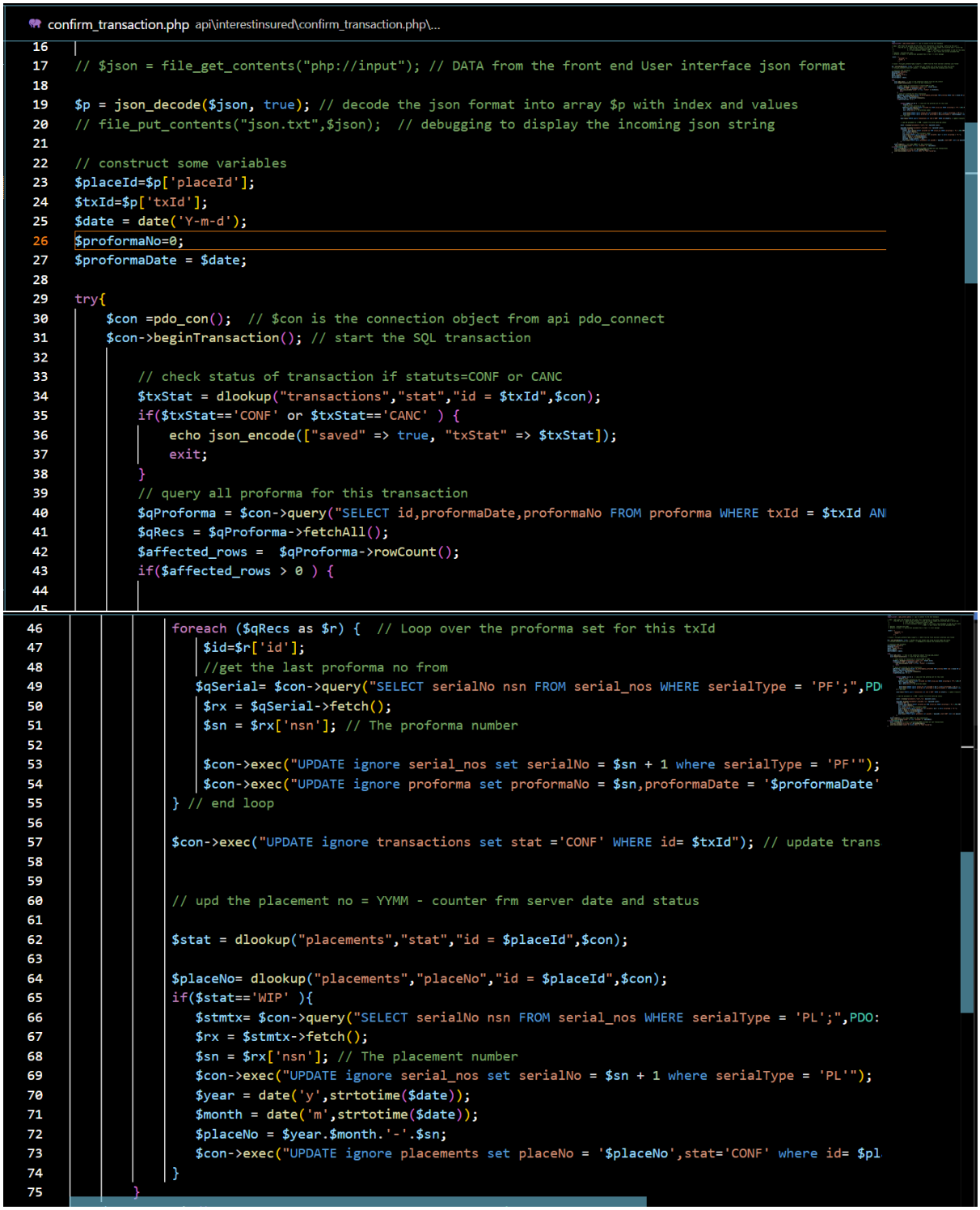
```php
}
$placeId = $p ['placeId'];
$txId = $p ['txId'];
$rate = $p['rate'] ;
$SumInsured = $p['SumInsured'];
$custId = $p['custId'];
try{
    $con = pdo_con();
    // check status of transaction if status=CONF or CANC
    $txStat = dlookup("transactions","stat","id = $txId",$con);
    if  ($txStat =='CONF' OR $txStat =='CANC' ){
    |   echo "txstat:"; echo $txStat; echo "<br>";
    exit;
    }
    // Get some data from the placement
    $stmt = $con ->query ("SELECT fscRate,brokerOpt,coverId,brokerpc,polFee,docFee,docFeeOpt,fscOpt,ins
    |   |   |   |   |   |   FROM placements WHERE id =$placeId;",PDO::FETCH_ASSOC);
    $rs=$stmt->fetch();
    $fscRate = $rs['fscRate'];
    $brokerOpt = $rs['brokerOpt'];
    $coverId = $rs['coverId'];
    $brokerpc = $rs['brokerpc'];
    $polFee = $rs['polFee'];
    $docFee = $rs['docFee'];
    $docFeeOpt = $rs['docFeeOpt'];
    $fscOpt = $rs['fscOpt'];
    $insId = $rs['insId'];
```

```php
    // Business Logic
    $annualPremium = ($rate / 100) * $SumInsured;
    //  Get the number of days for this transaction
    $stmt = $con ->query("SELECT DATEDIFF (toDate,fromDate ) AS noDays FROM transactions WHERE id = $tx
    $rs=$stmt->fetch();
    $noDays = $rs ['noDays'];
    $prorataPremium = ($noDays/365) *  $annualPremium;

    // ---------------fsc--------------
    if ($fscOpt == 'AUTO' ) {
    $fscRate = dlookup("sysparm","fsc","id=1",$con);
    }
    if ($fscOpt == 'MANUAL' ) {
    $fscRate = $fscRate;
    }
    $fsc = $fscRate * $prorataPremium ;
    //------broker commission-------
    if ($brokerOpt == 'YES' ) {
    $brokerPercent = dlookup("tbl_insurancerisks","RbrokerPercent","id={$p['coverId']}",$con);
    }
    if ($brokerOpt == 'NO' ) {
    $brokerPercent = 0;
    }
    if ($brokerOpt == 'MANUAL' ) {
    $brokerPercent = $brokerpc;
    }

    $brokerFee = ($brokerPercent / 100) * $prorataPremium;
    $grossPremium = $prorataPremium + $fsc;
    $netPremium = ($prorataPremium + $fsc) - $brokerFee;
```

- L'API **confirm_transaction**, une fois les biens assurés ajoutés permet à l'utilisateur de **confirmer la transaction**, ce qui verrouille ses données et génère un **devis proforma** ,l'API met à jour la transaction et génère un **numéro de proforma unique ET** l'API met à jour la table proforma en regroupant les informations des biens assurés.

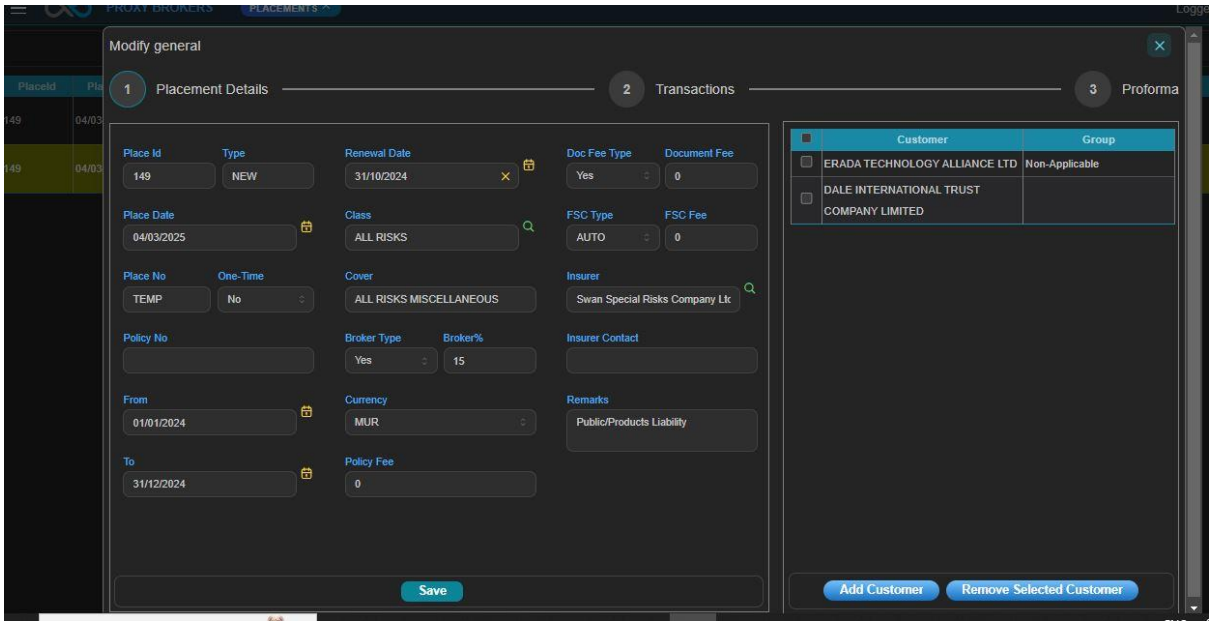


```
confirm_transaction.php api\interestinsured\confirm_transaction.php\...
16
17  // $json = file_get_contents("php://input"); // DATA from the front end User interface json format
18
19  $p = json_decode($json, true); // decode the json format into array $p with index and values
20  // file_put_contents("json.txt",$json);  // debugging to display the incoming json string
21
22  // construct some variables
23  $placeId=$p['placeId'];
24  $txId=$p['txId'];
25  $date = date('Y-m-d');
26  $proformaNo=0;
27  $proformaDate = $date;
28
29  try{
30      $con =pdo_con();  // $con is the connection object from api pdo_connect
31      $con->beginTransaction(); // start the SQL transaction
32
33          // check status of transaction if statuts=CONF or CANC
34          $txStat = dlookup("transactions","stat","id = $txId",$con);
35          if($txStat=='CONF' or $txStat=='CANC' ) {
36              echo json_encode(["saved" => true, "txStat" => $txStat]);
37              exit;
38          }
39          // query all proforma for this transaction
40          $qProforma = $con->query("SELECT id,proformaDate,proformaNo FROM proforma WHERE txId = $txId AN
41          $qRecs = $qProforma->fetchAll();
42          $affected_rows =  $qProforma->rowCount();
43          if($affected_rows > 0 ) {
44
45
46              foreach ($qRecs as $r) {  // Loop over the proforma set for this txId
47                  $id=$r['id'];
48                  //get the last proforma no from
49                  $qSerial= $con->query("SELECT serialNo nsn FROM serial_nos WHERE serialType = 'PF';",PD
50                  $rx = $qSerial->fetch();
51                  $sn = $rx['nsn']; // The proforma number
52
53                  $con->exec("UPDATE ignore serial_nos set serialNo = $sn + 1 where serialType = 'PF'");
54                  $con->exec("UPDATE ignore proforma set proformaNo = $sn,proformaDate = '$proformaDate'
55              } // end loop
56
57              $con->exec("UPDATE ignore transactions set stat ='CONF' WHERE id= $txId"); // update trans
58
59
60              // upd the placement no = YYMM - counter frm server date and status
61
62              $stat = dlookup("placements","stat","id = $placeId",$con);
63
64              $placeNo= dlookup("placements","placeNo","id = $placeId",$con);
65              if($stat=='WIP' ){
66                 $stmtx= $con->query("SELECT serialNo nsn FROM serial_nos WHERE serialType = 'PL';",PDO:
67                 $rx = $stmtx->fetch();
68                 $sn = $rx['nsn']; // The placement number
69                 $con->exec("UPDATE ignore serial_nos set serialNo = $sn + 1 where serialType = 'PL'");
70                 $year = date('y',strtotime($date));
71                 $month = date('m',strtotime($date));
72                 $placeNo = $year.$month.'-'.$sn;
73                 $con->exec("UPDATE ignore placements set placeNo = '$placeNo',stat='CONF' where id= $pl
74              }
75          }
```

Fonctionnement du système : Du placement à la génération du proforma

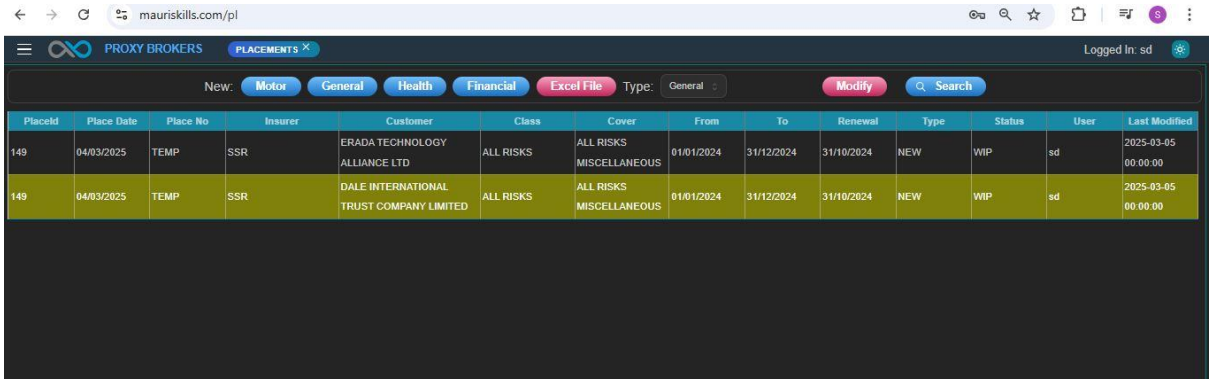
L'utilisateur suit plusieurs étapes pour enregistrer un placement, y ajouter des transactions, des biens assurés et générer un **proforma**.
Voici **le cheminement complet**, avec les interactions entre **l'interface utilisateur (ReactJS)** et le **back-end (API PHP)**, jusqu'à l'enregistrement des données en base.

L’utilisateur accède à l’interface **d’ajout d’un placement** et saisit les informations:



Le clic sur **"Save"** envoie une requête POST à l’API add_placement.php.

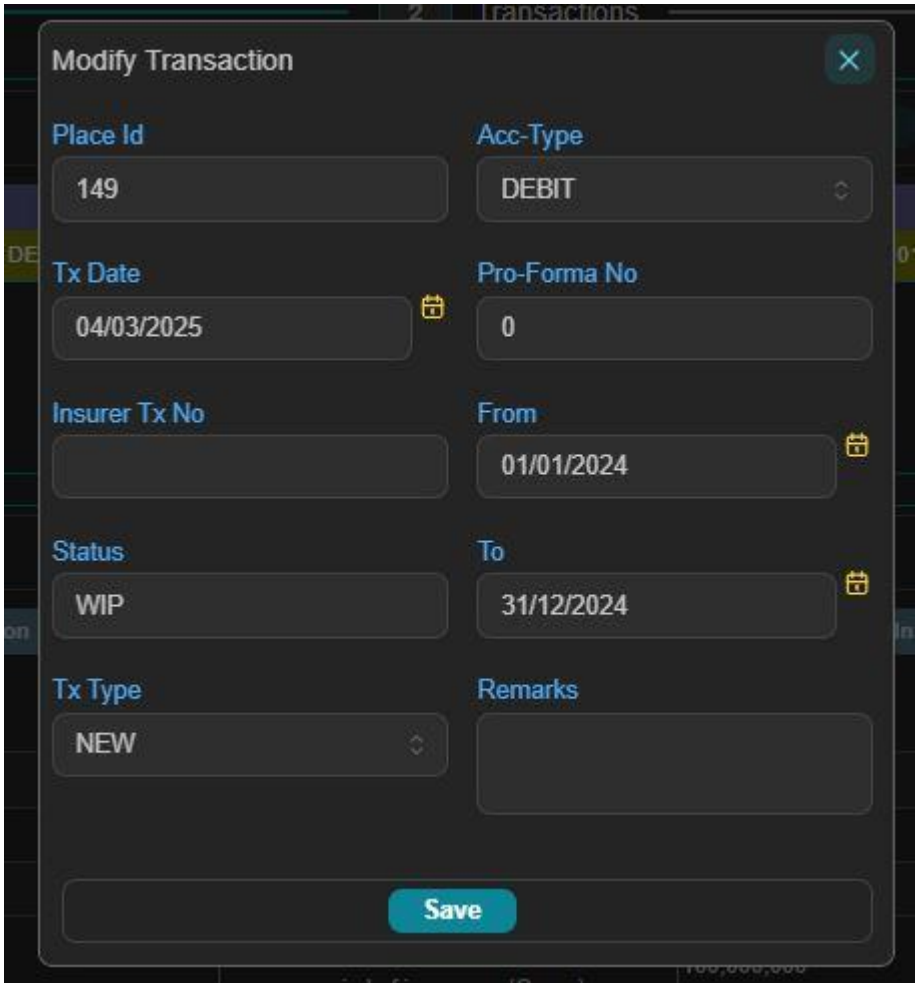
- L’API vérifie les **champs obligatoires** et applique les **business logic**.
- Si tout est valide, l’API insère les données



Une **réponse JSON** est envoyée confirmant l’ajout du placement.

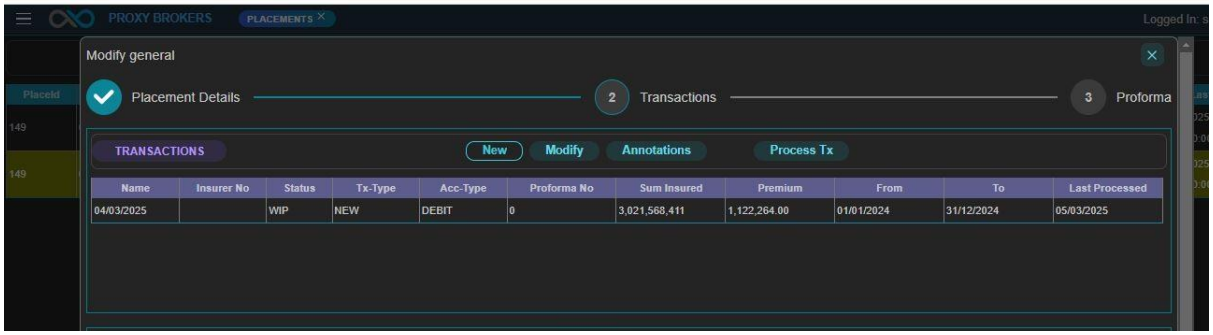
{"saved":true,"new_id":"149","placeNo":"XXX","message":"Successful"}

Dans la liste des placements, l’utilisateur sélectionne le placement créé et clique sur **"Transaction"**.
Il saisit les informations :



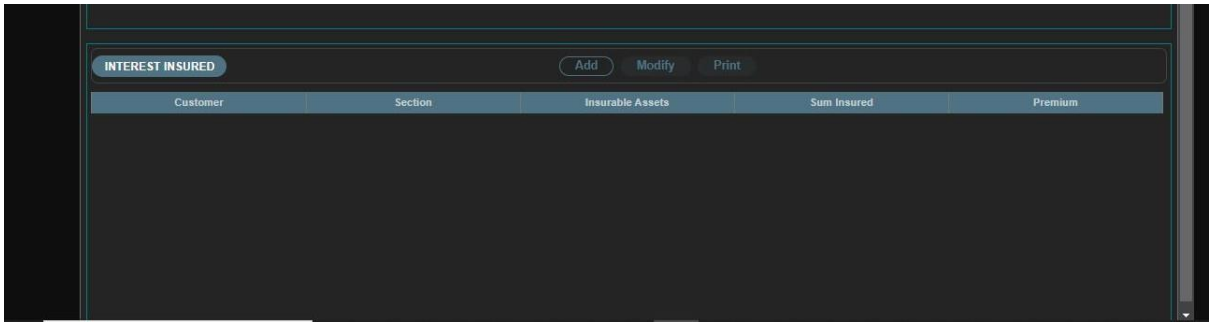
Le clic sur **"Save"** envoie une requête POST à add_transaction.php.

- L'API vérifie que le **placement existe** avant de traiter la transaction.
- Si tout est correct, l'API exécute :



Une **réponse JSON** est envoyée avec l'ID de la transaction.

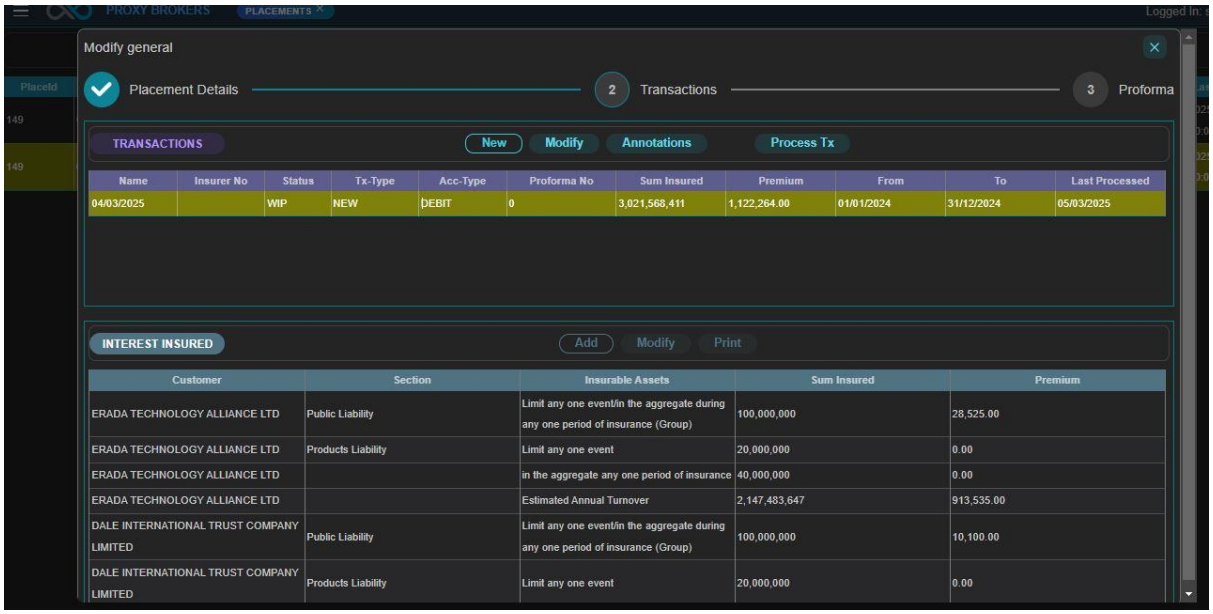
L'utilisateur sélectionne la transaction et clique sur **"Add Interest Insured"**.



Le clic sur **"Save"** envoie une requête POST à add_interestinsured.php.

- L'API vérifie que **la transaction est encore en cours (WIP)**.
- Elle applique les **calculs automatiques du business logic** :

Une **réponse JSON** est envoyée confirmant l'ajout du bien assuré.

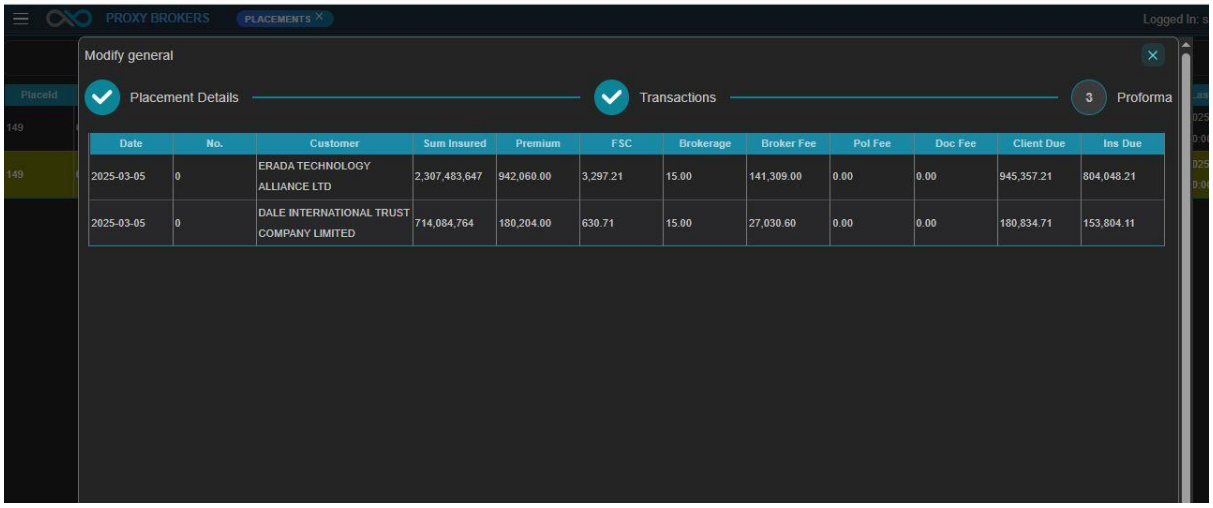


Après avoir ajouté tous les biens assurés, l'utilisateur confirme la transaction en cliquant sur **"Confirm Transaction"**.

Une requête POST est envoyée à confirm_transaction.php.

- L'API vérifie que la **transaction n'a pas déjà été confirmée**.
- Elle génère un **numéro de proforma unique** et met à jour la transaction :

Une **réponse JSON** est envoyée confirmant la validation et retournant le placeNo.



Grâce à ce processus, **tout le cycle d'un placement d'assurance est automatisé**

CONCLUSION

Ce stage chez **Mauriskills Software Ltd** a été une expérience très enrichissante tant sur le plan technique que professionnel. Il m'a permis d'approfondir mes connaissances en développement backend, en gestion des bases de données et en conception d'API, tout en me confrontant aux exigences et aux réalités du monde du travail.

D'un point de vue technique, j'ai pu consolider mes compétences en **PHP**, en **gestion des bases de données MySQL**, ainsi qu'en **API**. La mise en place de requêtes SQL optimisées, l'implémentation d'une architecture robuste pour le backend et l'intégration d'API avec le frontend ont constitué des défis stimulants qui m'ont permis de renforcer ma maîtrise du développement full-stack. J'ai également appris à écrire du code plus propre et plus optimisé, en respectant les bonnes pratiques et les standards de l'industrie.

Au-delà des aspects techniques, ce stage m'a également offert une expérience humaine et organisationnelle précieuse. Travailler au sein d'une équipe de développeurs m'a permis de mieux comprendre les méthodes de gestion de projet et de collaboration. Cette immersion dans un environnement professionnel dynamique m'a aidé à développer des compétences en communication, en gestion du temps et en autonomie.

Un autre aspect important de ce stage a été la capacité à résoudre des problèmes concrets. Confronter des bugs, optimiser des requêtes SQL ou s'assurer de la bonne communication entre les différents modules de l'application m'ont appris à réfléchir de manière logique et à trouver des solutions adaptées.

En outre, travailler sur un projet de digitalisation pour **Proxy Brokers Ltd** m'a offert une vision plus concrète des enjeux de transformation numérique dans les entreprises. Comprendre les besoins du client, adapter les solutions techniques en fonction des contraintes et proposer des améliorations pertinentes m'a permis de développer une approche plus concrète du développement logiciel.

Ce stage a confirmé mon intérêt pour le développement logiciel et m’a donné envie de continuer à apprendre et à progresser dans ce domaine. Il m’a permis d’acquérir une expérience précieuse sur le terrain, qui me sera très utile pour la suite de mon parcours. J’ai aussi pris conscience de l’importance de la veille technologique pour m’adapter aux besoins du marché.