INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

CELINE CHAUGNY
DAMIEN POINTIN

Plateforme d'analyse technique pour la finance

Projet Fin d'Etude

Table des matières

Chapitre 1
Introduction

1.1 Introduction

Chapitre 2
Techniques utilisées

Une partie important de notre projet de fin d'étude était de savoir quelles techniques utiliser afin de répondre à nos besoins et exigences. Nous avons choisi de créer un site Internet en utilisant le langage Java et pour cela, l'environnement Eclipse avec un Web Dynamic Project s'y prete bien. Dans cette partie nous allons donc développer les différents points informatiques techniques que nous avons abordé et nous détaillerons les choix que nous avons fait pour notre application.

2.1 Le modèle MVC et l'environnement Java EE

2.2 Architecture Modèle-Vue-Contrôleur

Etant donné le fait que nous avons choisi de faire une application interactive, nous avons utilisé l'architecture logicielle **MVC**. Ainsi, les problématiques liées aux composantes de notre application sont bien séparées :

- **Le modèle** : notre code Java qui modélise les données et qui sera reliée à une base de données pour stocker diverses informations. C'est le coeur du programme.
- **La vue** : l'interface homme-machine qui sera sous forme d'un site Web.
- **Le contrôleur** : la partie de notre code qui fait le lien entre le modèle et la vue. Il agit en fonction de ce que l'utilisateur demande à la vue. Il est chargé de de chaque côté et les transmets à l'un et l'autre comme le montre le schéma ci-dessous.

Le schéma suivant présente les intéractions entre les trois entités de notre architecture logicielle.

1. L'utilisateur intéragi avec la vue (le site Web) et lance une action du contrôleur qui reçoit les informations.
2. Le contrôleur agit alors en conséquence en envoyant des informations au modèle.
3. Le modèle traite les informations puis renvoie le résultat de son action au contrôleur et des données dans certains cas.
4. Le contrôleur fait un compte rendu à l'utilisateur en mettant à jour la vue.

2.3 Environnement Java EE

L'environnement de développement Java EE es

2.3.1 Serveur Apache Tomcat 8.0

Apache Tomcat est un conteneur web libre de servlets et JSP Java EE. Il permet notammement l'utilisation des servlets et des JSP (JavaServer Pages). Tomcat est un serveur HTTP avant tout. Il est écrit en Java ce qui permet de l'utiliser sur n'importe quelle système d'exploitation via la machine virtuelle Java. Il est paramétrable par des fichiers XML. Dans notre projet, nous utilisons la dernière version d'Apache Tomcat 8.0 qui est suport de Java 7 et a d'autres avantages dont nous avions besoin.

2.3.2 Les servlets

Les servlets sont des classes implémentées en Java qui permettent de rendre dynamique une page HTML. Elles utilisent l'API Java Servlet qui correspond au package javax.servlet.

Si on repart du schéma suivant, on reprend le mécanisme existant entre le client et le serveur HTTP : le client envoie une requête HTTP au serveur et ce dernier lui retourne une réponse.

Tout l'intérêt de l'utilisation des servlets réside dans le fait qu'il devient alors possible de maîtriser le traitement des requêtes mais surtout de personnaliser les réponses HTTP qui sont faites au client.

Ainsi, dans la plus part des cas, le fonctionnement d'une servlet peut se résumer au fait que c'est une classe Java qui reçoit une requête HTTP envoyée depuis un navigateur par un client et lui renvoie une réponse HTTP.

L'ensemble des servlets est appelé conteneur de servlets ou conteneur web.
Lors de l'implémentation d'une servlet en Java, il faut créer une classe qui hérite de la classe abstraite HttpServlet et pour cette raison elle doit implémenter à minima l'un des trois méthodes suivantes :

- doGet() : gère la méthode GET
- doPost() :
- doHead() :

parler du XML (un pour chaque servlet)
En effet, une page HTML est statique et si on souhaite la rendre dynamique A titre d'exemple, Ainsi, les servlets seront assimilées au contrôleur de notre archicture MVC (et les pages JSP à la vue).

2.3.3 Les JSP (Java Server Pages)

Une Java Server Page (on utilisera le terme de JSP par la suite) est un document texte qui peut contenir des balises HTML mais également des balises permettant d'inclure du code Java. Le langage utilisé dans les JSP permet aussi l'utilisation d'autres technologies comme XML, les servlets, le CSS... et ce dans un seul et unique fichier ce qui rend les rend d'autant plus intéressante.

On utilise des JSP car cela simplifie l'utilisation des technologies servlets. En effet, écrire une page web en Java peut vite devenir pénible. Les JSP permettent donc d'utiliser la technologie des servlet d'une manière simplifiée puisqu'on peut écrire une page HTML de manière classique.

D'autre part, on les utilise afin de respecter la structure MVC puisque cela permet de retirer des servlets la partie 'vue' et de n'y laisser que la partie 'contrôleur'. De même on sépare la vue du modèle puisque il peut se trouver au milieu des servlets du code métier. Ainsi, la servlet est le contrôleur qui relie la vue (JSP) au modèle. Tout ceci respectant bien l'architecture MVC.

L'avantage de l'utilisation des JSP est que les pages sont exécutées par un serveur et ainsi on peut avoir une page dynamique contrairement aux pages HTML basiques.
La page étant alors dynamique, on peut faire varier l'affichage de la page et avoir une interaction avec l'utilisateur.
Les JSP
FAIRE EXEMMLE MVC avec Joueur

2.4 La base de données

2.4.1 MySQL

Présentation

MySQL est un système de gestion de bases de données (SGDB) relationnelles. Ce logiciel étant libre et open source, il est accessible à tout le monde et gratuitement dans la plupart des cas. Il utilise le langage SQL (Stuctured Query Language) pour les requêtes.

Mise en place

La mise en place de MySQL est assez simple. Il suffit de télécharger MySQL pour la version du système d'exploitation utilisé. On aura alors accès à un environnement en local dans lequel il est possible de créer des bases de données.

On fait maintenant l'hypothèse que l'on utilise Linux. On pourra alors créer notre base de données et ses tables soit en ligne de commande soit à l'aide d'une interface graphique. L'interface utilisée pourra être phpMyAdmin qui est simple à utiliser.

Le langage SQL permet de rechercher, ajouter, modifier ou supprimer des données dans une base de données relationnelle. Les instructions sont des requêtes plus ou moins complexes constituées de mots clés tels que 'SELECT', 'UPDATE', 'INSERT' ou encore 'DELETE'. Il est également possible d'utiliser des opérations d'algèbre relationnelle comme 'FROM', 'JOIN', ou 'GROUP'. On ne souhaite pas détailler l'ensemble des possibilités offertes par le langage SQL ici, mais voici une exemple dans lequel on créée une base de données pour stocker des joueurs avec leur *nom* et leur *prenom* dans une table. On ajoute ensuite un joueur à la table et on affiche l'ensemble des joueurs.

```
/* Creation de la base de donnees et on se place dedans */
CREATE DATABASE baseJoueurs;
USE baseJoueurs;

/* Creation de la table Joueur */
CREATE TABLE Joueur (
  nom VARCHAR(20) NOT NULL,
  prenom VARCHAR(20) NOT NULL
);

/* Ajout d'un joueur */
INSERT INTO Joueur (nom,prenom) VALUES ('Dupont', 'Jean');

/* Requete pour recuperer tous les joueurs de la table */
SELECT * FROM Joueur;
```

Avantage et Inconvénient

Le premier avantage pour nous est le fait que nous sommes familier avec ce langage alors que nous ne n'avons pas eu l'occasion d'utiliser d'autres SGDB. De plus, il est pratique de pouvoir utiliser phpMyAdmin afin de visualiser nos tables. Enfin, la capacité de stockage est amplement suffisante (plusieurs centaines de Go de données). Un inconvénient de ce SGDB dans le cadre de notre projet est qu'il ne permet pas d'effectuer des transactions finanicères intensives ce qui pourrait poser problème si l'on voulait ajouter cette fonctionnalité à notre système.

2.4.2 Connecteur JDBC

Présentation

JDBC signifie Java Data Base Connectivity. C'est une interface Java qui permet d'accéder à une base de données SQL ou MySQL. L'avantage de ce driver est qu'il permet de se connecter à une base, d'exécuter des requêtes et d'en récupérer les résultats de manière simple. Tout se fait en Java, ce qui permet de l'intégrer à notre environnement Java EE sans problème. Il suffit alors de connaître les commandes SQL et les quelques caractéristiques du JDBC.

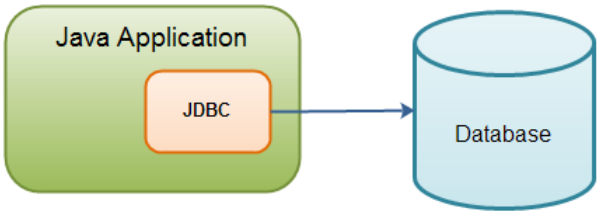


FIGURE 2.1 – Schéma simple du JDBC - http ://tutorials.jenkov.com/jdbc/index.html

L'API JDBC est composée des interfaces et classes suivantes :

Driver : cette interface gère les communications avec le serveur de base de données. On ne l'utilise pas directement, on utilisera plutôt le DriverManager.

DriverManager : cette classe gère la liste des pilotes de base de données. Il choisira à la connexion le pilote correspondant à la base utilisée.

Connection : cette interface contient toutes les méthodes permettant d'échanger avec la base de données.

Statement : cette interface permet de soumettre des requêtes SQL à la base de données.

ResultSet : cette classe contient les données récupérées suite à une requête à la base de données après son exécution.

SQLException : cette classe gère les différentes erreurs qui peuvent se produire lors de l'accès à la base de données.

Mise en place

Une fois que le pilote est chargé dans le code Java, on se connecte à la base de données voulue avec l'URL, l'utilisateur et le mot-de-passe. Le package java.sql est ici nécessaire. Lorsque la connexion a été établie, l'exécution d'une requête se fait de la manière suivante :

```
1 // Chargement du pilote :
  Class.forName("com.mysql.jdbc.Driver").newInstance();
3
  // Connexion a la base de donnees :
5 java.sql.Connection connexion = DriverManager.getConnection("jdbc:mysql:URL", "utilisateur", "
    motDePasse");

7 // Generation de l'objet statement associe  la connexion :
  java.sql.Statement statement = connexion.createStatement();
```

Maintenant que la connexion est établie, il est possible d'interagir avec la base de données. Dans les exemples ci-dessous, nous supposerons que la base de données est composée d'une table Joueur dont les attributs sont des chaînes de caractères représentant le nom et le prénom des joueurs. Il existe deux fonctions principales associées à différents types de requêtes et qui s'appliquent à l'objet Statement :

- *executeQuery(String sql)* : permet d'exécuter une requête du type 'SELECT' par exemple et qui donne retourne certaines lignes d'une table.

```
// Ecriture et execution de la requete SQL :
String requete = "SELECT * FROM Joueur";
ResultSet resultat = statement.executeQuery(requete);
```

- *executeUpdate(String sql)* : permet de mettre à jour une table de la base de données dans le cadre de requêtes du type 'UPDATE', 'INSERT', 'DELETE'.

```
// Mise a jour d'une table (ajout d'un joueur) :
String udpate = "INSERT INTO Joueur (nom, prenom) VALUES ('Dupont', 'Jean')";
int nbLignesMAJ = statement.executeUpdate(update);
```

Une fois que la requête a été exécutée, il ne reste plus qu'à lire les données reçues. Dans le cadre de l'utilisation de la fonction executeUpdate, on récupère simplement un entier qui donne le nombre de lignes mise à jour dans la table. En revanche, la fonction executeQuery retourne un objet du type ResultSet qui est moins trivial à étudier. Cet objet contient une certain nombre de lignes répondant aux conditions de la requête et peut être comparé à une table d'une base de données. Il est composé de colonnes symbolisant les attributs des enregistrements de la table : ils possèdent un nom et un domaine dans lequel ils prennent leur valeur (int, float, char, ...). De plus, pour parcourir cet objet on a une sorte de curseurs qui point sur une ligne. On utilise la commande next pour passer à la ligne suivante si elle existe.

```
// Parcours du ResultSet s'il est non nul et tant qu'il y a une ligne :
Vector<String> noms;
Vector<String> prenoms;
if (resultat != null) {
  while (resultat.next()) {
    noms.add(resultat.getString("nom"));
    prenoms.add(resultat.getString("prenom"));
  }
}
```

Après l'exécution de requêtes et la récupération des données, il ne reste plus qu'à 'refermer' la connexion de la manière suivante :

```
// Fermeture des ojets :
statement.close();
connexion.close();
```

Les exemples de codes précédents ne prennent pas en compte les erreurs pouvant être générées lors de la compilation de ce code. En réalité, il faut protéger les instructions et lever les exceptions générées. Voici les erreurs qui peuvent être générées :

- Lors de la connexion : échec de la connexion (ex : mauvais mot de passe, URL invalide,...).
- Lors de l'exécution d'une requête : mauvaise syntaxe ou appel de table/attributs inexistants.
- Lors de la déconnexion.

Avantage et Inconvénient

Il existe d'autre moyens d'accéder à une base de données MySQL en Java tel que Hibernate mais il s'utilise dans le cadre d'une architecture ORM (Objet/Relational Mapping) alors que nous allons utliser une architecture DAO (Data Access Object) pour l'accès à notre base de données. L'avantage du JDBC est qu'il est simple à utiliser (seulement deux méthodes pour exécuter des requêtes SQL) et qu'il contient des pilotes permettant d'accéder à n'importe quelle base de données de type relationnel.

2.4.3 DAO

Présentation

Un objet d'accès aux données (Data Access Object ou DAO en anglais) est un modèle qui permet d'isoler les méthodes concernant le stockage des données et de ne pas les écrire directement dans nos classes métier. Ainsi le modèle DAO permet de regrouper l'accès aux données dans des classes à part plutôt que de les disperser.

Le but du modèle DAO est d'arriver à encapsuler les méthodes concernant la communication avec la base de données dans une couche pour arriver au schéma suivant :

FIGURE 2.2 – Schéma simple du modèle DAO

Mise en place

La couche DAO va gérer les opérations classiques de stockage : ajout, lecture, modification et suppression. Ces quatre opérations sont souvent raccourcis par l'acronyme CRUD (create, read, update and delete).

Pour mettre en application le modèle DAO nous avons du créer nos propres exceptions. En effet, il faut que les exceptions générées par SQL ou JDBC soit référencées comme étant des exceptions dues à la boite noire DAO.
En reprenant la même idée, DAO se propose d'offrir une interface pour chacun des objets décrivant l'ensemble des méthodes qui seront accessibles dans l'objet. Ainsi, peu importe l'implémentation effectuée on pourra connaître les diverses méthodes de nos classes. Les interfaces permettent de décrire les méthodes des objets la couche donnée et ce n'est que l'implémentation qui sera dépendante du mode de stockage. Par exemple, pour un stockage SQL nous pouvons utiliser le driver JDBC présenté ci-dessus.

La couche DAO comportera également une fabrique. Cette fabrique sera unique et ne sera instanciée que si les informations de configuration sont correctes. Le but de cette fabrique sera de fournir une instance des différentes implémentations de la DAO.

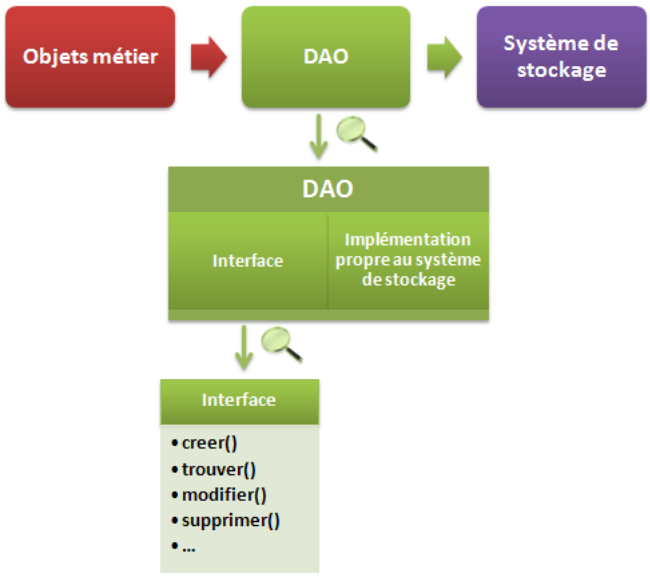


FIGURE 2.3 – Schéma détaillé du modèle DAO

Avantage et Inconvénient

L'avantage du modèle DAO est donc clair, le changement du mode de stockage de données est simple. En effet, nous aurons seulement à changer nos classes DAO pour changer ce mode de stockage. L'inconvénient est du à la mise en œuvre qui demande une couche supplémentaire.

2.4.4 La sérialisation

Principe

La sérialisation est le processus de conversion d'un objet pour l'enregistrer dans une base de données ou un fichier par exemple. Le processus inverse s'appelle la désérialisation. Nous pouvons donc représenter cette méthode selon le schéma suivant.

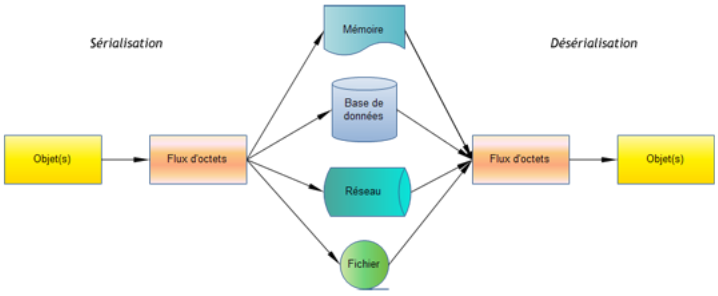


FIGURE 2.4 – Schéma de la sérialisation

Mise en place

On peut choisir par exemple d'effectuer sa sérialisation en XML.

La classe XMLEncoder permet de sérialiser un objet en XML. Cette sérialisation ne prend en compte que les champs ayant un getter et un setter public. XMLEncoder(output) crée une nouvelle instance qui

utilise le flux passé en paramètre comme résultat de la sérialisation.

La classe XMLDecoder permet quand à elle de désérialiser un objet à partir d'un document XML.

Pour notre application nous aurions besoin d'effectuer une sérialisation vers SQL. Prenons l'exemple du joueur, pour effectuer la sérialisation nous avons besoin d'une classe Joueur qui implémente Serializable, d'avoir un constructeur par défaut et d'avoir un getter et un setter public pour chaque attribut. Ensuite notre table SQL java doit contenir un id et un longblob (Binary Large Object). Ensuite nous n'avons plus qu'a effectue une sauvegarde dans la base de données. Pour la desérialisation, il nous faut récupérer le longblob pour le convertir en ObjectInputStream pour utiliser la méthode readObject().

Avantage et Inconvénient

La sérialisation présente des avantages comme une bonne portabilité ou le fait que le processus de sérialisation ne prenne pas en compte les champs qui ont leur valeur par défaut. Elle présente les inconvénients suivants :

- ne peut s'utiliser que sur des objets respectant la convention JavaBeans (constructeur par défaut, getter et setter pour tous les attributs ..)
- la taille des données sérialisés est plus importante que leur équivalent binaire

2.4.5 Notre choix : DAO vs Serialisation

Pour la communication avec notre base de données nous avons choisi d'utiliser le modèle DAO.

Le modèle DAO demande plus de travail pour être mis en place que la sérialisation mais possède l'avantage de nous fournir une base de données plus facilement utilisable dans un autre contexte. En effet, chacune de nos tables ne seront pas simplement un blob mais une valeur pour chacun des champs de notre table.

2.5 Introduction

2.6 Cryptage du mot de passe

Lors du déroulement du jeu nous allons avoir besoin de stocker le mot de passe du joueur. Pour cela, nous avons pensé qu'il était préférable d'encrypter ce mot de passe lors de son stockage.
Nous avons ainsi chercher différentes méthodes pour effectuer l'encryptage du mot de passe.

2.6.1 Fonctions SQL

Dans le langage SQL, la fonction MDA5() permet de chiffrer une chaîne de caractère en un entier hexadécimal de 32 caractères.
L'algorithme MDA() est une fonction de hachage cryptographique qui calcule à partir d'une chaîne de caractère son empreinte avec une probabilité très forte que deux empreintes soient différentes. Depuis 2004, une équipe chinoise a découvert des collisions complètes et MD5 n'est donc plus considéré comme sur au sens cryptographique.

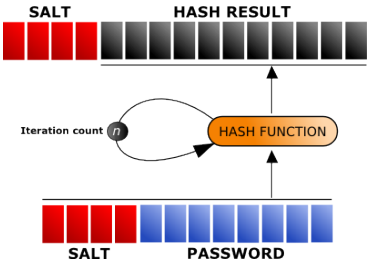
La fonction SQL SHA1() permet de chiffrer une chaîne de caractère sous la forme d'un chaîne de caractères de 40 caractères. SHA1 est également une fonction de hachage cryptographique. Elle a l'avantage d'être considéré comme sur contrairement à MDA().

2.6.2 Jasypt

Jasypt est une librairie java qui permet d'encrypter facilement les mots de passe avec une une grande sécurité.
Jasypt possede les avantages suivants :

- il permet de choisir la fonction de hachage que nous souhaitons (MDA ou SHA par exemple)
- il ajoute un salage au mot de passe qui permet d'avoir deux mots de passe cryptés différents pour le même mot de passe de départ
- il applique un nombre aléatoire de fois notre fonction de hachage (nombre > 1000 pour rendre plus difficile les attaques)

Nous pouvons résumer l'algorithme de chiffrement de Jasypt comme suit :



Le code pour encrypter le mot de passe est très simple, il nous suffit de créer un objet de type ConfigurablePasswordEncryptor, de définir l'algorithme de chiffrement. La méthode setPlainDigest nous permet avec l'argument false de choisir la méthode la plus sure avec un salage et un nombre d'itération aléatoire pour notre fonction de hachage. Enfin il ne nous reste plus qu'a appeler la méthode encryptPassword qui nous renvoie notre mot de passe encrypté à partir d'un mot de passe donné en entrée.

```
ConfigurablePasswordEncryptor passwordEncryptor = new ConfigurablePasswordEncryptor();
passwordEncryptor.setAlgorithm( ALGO_CHIFFREMENT );
passwordEncryptor.setPlainDigest( false );
String motDePasseChiffre = passwordEncryptor.encryptPassword( motDePasse );
```

De même que pour vérifier que notre mot de passe correspond au mot de passe chiffré il existe une méthode qui nous renvoie vraie en cas de correspondance :

```
passwordEncryptor.checkPassword(motDePasse, motDePasseChiffre )
```

2.6.3 Notre choix

L'inconvénient d'utiliser les fonctions de SQL est que l'on choisi une manière de crypter dépendante de notre base de données. Si nous décidons de changer notre manière de stocker notre base de données nous devrons ainsi trouver une nouvelle fonction.

Nous avons donc choisi d'utiliser Jasypt pour encrypter notre mot de passe. Cette librairie a l'avantage de nous permettre d'encrypter uen chaîne de caractère de manière relativement sure sans avoir de grandes compétences en cryptographie. En effet, nous ne connaissons pas en détail le fonctionnement de l'algorithme de cryptage mais avons simplement une idée globale de son fonctionnement.

Nous avons choisi comme fonction de hachage (ALGOCHIFFREMENT) SHA car nous avons que MDA n'est plus sur. Une fois l'encryptage du mot de passe effectué nous obtenons une chaîne de caractères de taille 56.

2.7 Introduction

2.8 Introduction

Chapitre 3
Théorie Finance

3.1 Introduction

3.2 Introduction

3.3 Introduction

3.4 Introduction

Chapitre 4
Modélisation

4.1 Introduction

4.2 Introduction

4.3 Introduction

4.4 Introduction

4.5 Introduction

4.6 Introduction

Chapitre 5
Utilisation de l'application

5.1 Introduction

Chapitre 6
Gestion de projet

6.1 Introduction

6.2 Introduction

Chapitre 7
Conclusion

7.1 Introduction

Chapitre 8
Bibliographie

http ://www.jasypt.org/howtoencryptuserpasswords.html
http ://sql.sh/fonctions/sha1
http ://sql.sh/fonctions/md5
https ://openclassrooms.com/courses/creez-votre-application-web-avec-java-ee/le-modele-dao
http ://www.easywayserver.com/blog/save-serializable-object-in-java/

JDBC :
http ://www.tutorialspoint.com/jdbc/jdbc-introduction.htm
http ://www.fobec.com/java/943/connecter-une-base-mysql-avec-driver-jdbc.html http ://tutorials.jenkov.com/jdbc/index.html

Table des figures

RENSEIGNEMENTS

Département GM
02.32.95.65.31
gm@insa-rouen.fr

INSA Rouen
Campus du Madrillet
685 avenue de l'Université – BP 08
76801 SAINT-ÉTIENNE-DU-ROUVRAY cedex
www.insa-rouen.fr

Membre de

GROUPE INSA
Normandie Université

Financeurs institutionnels

RÉGION HAUTE
MINISTÈRE DE L'ÉDUCATION NATIONALE, DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE