

PROJET PISCINES VANNETAISES

BTS SIO SLAM 2022/2023



Rémi COUSIN, Lionel DELABY, Ségolène GANZIN, Killian POSSEME

TABLE DES MATIÈRES

Contexte	4
Cahier des charges	4
Outils utilisés	5
Mise en place	6
Diagramme cas d'utilisation	6
Diagramme BD	7
Modèle relationnel projet piscines vannetaises	8
Création de la base de données dans SQL serveur management studio	9
Typage des données.....	10
Modèle Vue Contrôleur sur eclipse.....	12
Connection github.....	13
Installer eGit sur Eclipse	13
Relier le repertoire Github	14
Configurer le plugin eGit	14
Importer le repertoire Github dans Eclipse.....	15
Premier commit	17
Générer une clé sur github.....	17
Lier la clé à Eclipse.....	18
Modèle	20
dao (Data Access Object)	20
Prepare statement	22
Try / Catch	22
Gestion des tables d'association	23
Création du code	24
Piscine (métier)	25
Calcul du solde	26
Calcul des places restantes à un cours.....	26
Affichage des dates	27
Vue	27
Java FX.....	27
installation.....	27
Utilisation	27

IHM.....	29
Fichier css	29
Fenêtres « popup »	30
Messages personnalisés	31
Contrôleur	32
Application	32
GeneralController	32
Observables.....	33

CONTEXTE

Les piscines municipales de Vannes vont changer de logiciel afin de générer directement des codes permettant d'accéder aux bassins. Ces codes pourront être achetés sur des bornes situées à la piscine, sans aucune inscription. Nous ne traitons pas les problématiques de paiement en ligne, mais ce processus est simulé.

CAHIER DES CHARGES

La borne permettra de générer, au choix, des codes pour obtenir :

- Une leçon individuelle de nage sur un des créneaux libres
- Un abonnement solo 10 entrées valable 10 mois
- Un abonnement duo valable pour dix fois deux entrées simultanées, pendant un an.

Les administrateurs authentifiés pourront gérer :

- Le tarif
- Le nombre de places
- La durée de validité des titres en vente

La configuration devra être la plus générique/modulable possible.

Il faudra également définir un formulaire permettant à n'importe qui de tester la validité d'un code et de voir son contenu actuel.

OUTILS UTILISÉS



MISE EN PLACE

DIAGRAMME CAS D'UTILISATION

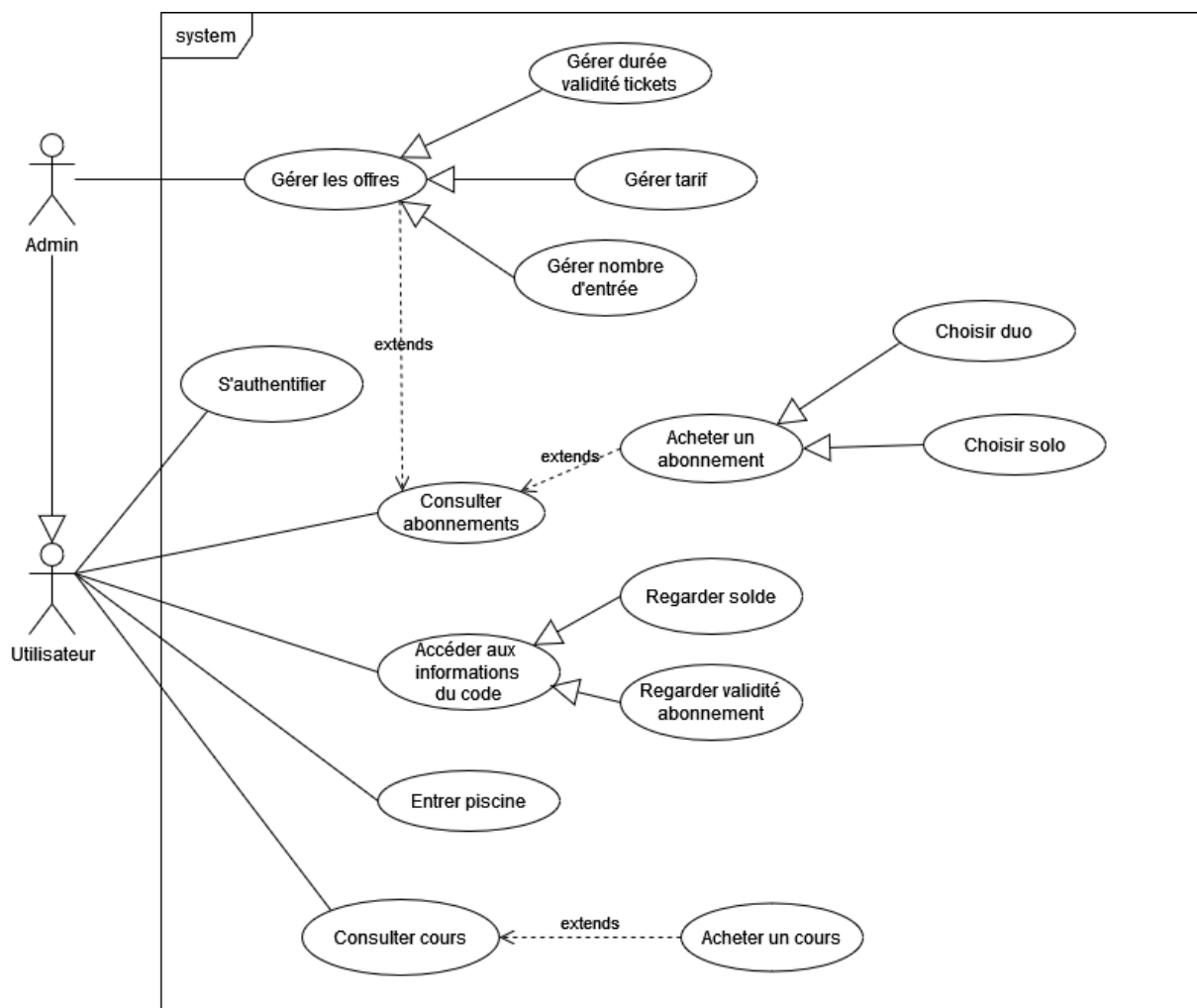
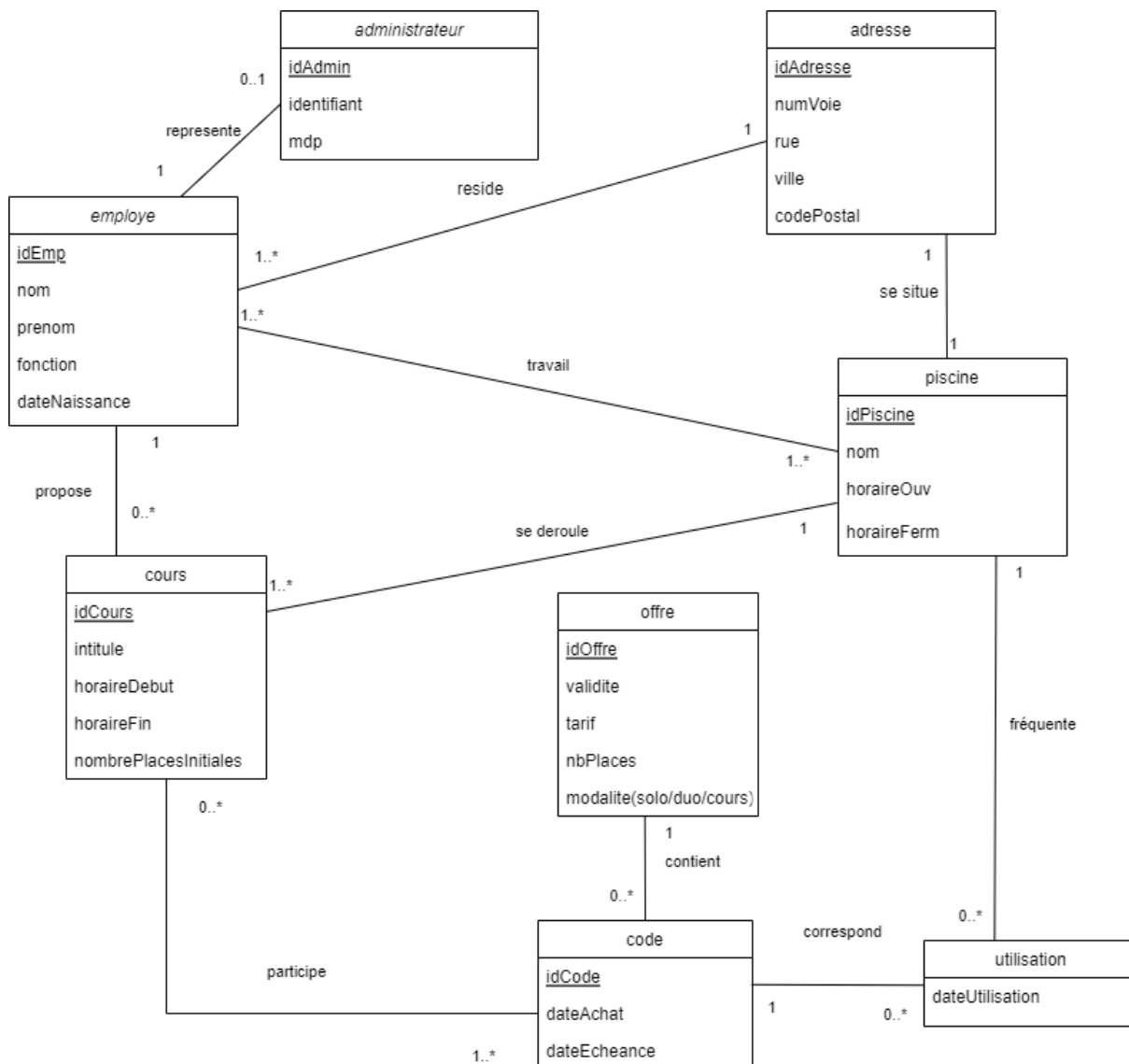


DIAGRAMME BD



MODÈLE RELATIONNEL PROJET PISCINES VANNETAISES

Administrateur (#idAdmin, identifiant, mdp) //idAdmin correspond à idEmp

Employe (idEmp, nom, prenom, fonction, dateNaissance, #idAdresse)

Adresse (idAdresse, numVoie, rue, ville, codePostal)

Piscine (idPiscine, nom, horaireOuv, horaireFerm, #idAdresse)

Travail (#idEmp, #idPiscine)

Offre(idOffre, valide, tarif, nbPlaces, modalite)

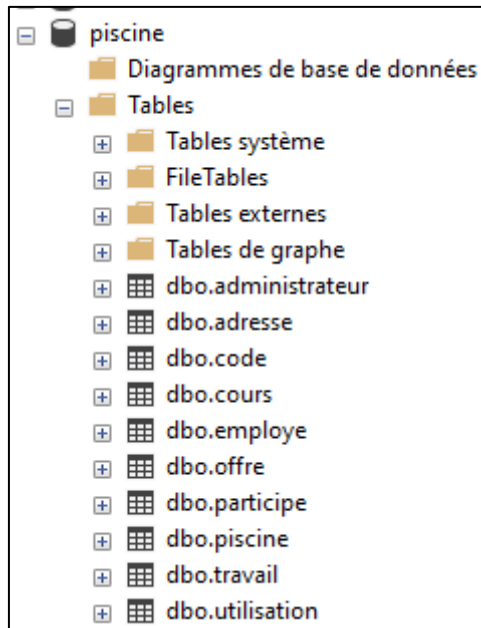
Code(idCode, dateAchat, dateEcheance, #idCatalogue, #idPiscine)

Participe(#idCode, #idCours)

Cours(idCours, intitule, horaireDebut, horaireFin, nombrePlacesInitiales, #idEmp, #idPiscine)


Utilisation(dateUtilisation, #idCode, #idPiscine)

CRÉATION DE LA BASE DE DONNÉES DANS SQL SERVEUR MANAGEMENT STUDIO




TYPAGE DES DONNÉES

Employe :

	Nom de la colonne	Type de données	Autoriser les valeurs Null
	idEmp	int	<input type="checkbox"/>
	nom	nvarchar(50)	<input type="checkbox"/>
	prenom	nvarchar(50)	<input type="checkbox"/>
	fonction	nvarchar(50)	<input type="checkbox"/>
	dateNaissance	date	<input type="checkbox"/>
	idAdresse	int	<input type="checkbox"/>



Adresse :

	Nom de la colonne	Type de données	Autoriser les valeurs Null
	idAdresse	int	<input type="checkbox"/>
	numVoie	nvarchar(50)	<input type="checkbox"/>
	rue	nvarchar(50)	<input type="checkbox"/>
	ville	nvarchar(50)	<input type="checkbox"/>
	codePostal	int	<input type="checkbox"/>


Piscine :

	Nom de la colonne	Type de données	Autoriser les valeurs Null
	idPiscine	int	<input type="checkbox"/>
	nom	nvarchar(50)	<input type="checkbox"/>
	horaireOuv	nvarchar(50)	<input type="checkbox"/>
	horaireFerm	nvarchar(50)	<input type="checkbox"/>
	idAdresse	int	<input type="checkbox"/>

Travail :

	Nom de la colonne	Type de données	Autoriser les valeurs Null
	idEmp	int	<input type="checkbox"/>
	idPiscine	int	<input type="checkbox"/>

Offre :

	Nom de la colonne	Type de données	Autoriser les valeurs Null
	idOffre	int	<input type="checkbox"/>
	validite	int	<input type="checkbox"/>
	tarif	float	<input type="checkbox"/>
	nbPlaces	int	<input type="checkbox"/>
	modalite	nvarchar(50)	<input type="checkbox"/>

Code :

	Nom de la colonne	Type de données	Autoriser les valeurs Null
🔑	idCode	nvarchar(50)	<input type="checkbox"/>
	dateAchat	datetime	<input type="checkbox"/>
	dateEcheance	datetime	<input type="checkbox"/>
	idOffre	int	<input type="checkbox"/>

Participe :

	Nom de la colonne	Type de données	Autoriser les valeurs Null
🔑	idCode	int	<input type="checkbox"/>
🔑	idCours	int	<input type="checkbox"/>

Cours :

	Nom de la colonne	Type de données	Autoriser les valeurs Null
🔑	idCours	int	<input type="checkbox"/>
	intitule	nvarchar(250)	<input type="checkbox"/>
	horaireDebut	datetime	<input type="checkbox"/>
	horaireFin	datetime	<input type="checkbox"/>
	nombrePlacesInitiales	int	<input type="checkbox"/>
	idEmp	int	<input type="checkbox"/>
	idPiscine	int	<input type="checkbox"/>

Administrateur :

	Nom de la colonne	Type de données	Autoriser les valeurs Null
🔑	idAdmin	int	<input type="checkbox"/>
	identifiant	nvarchar(50)	<input type="checkbox"/>
	mdp	nvarchar(50)	<input type="checkbox"/>

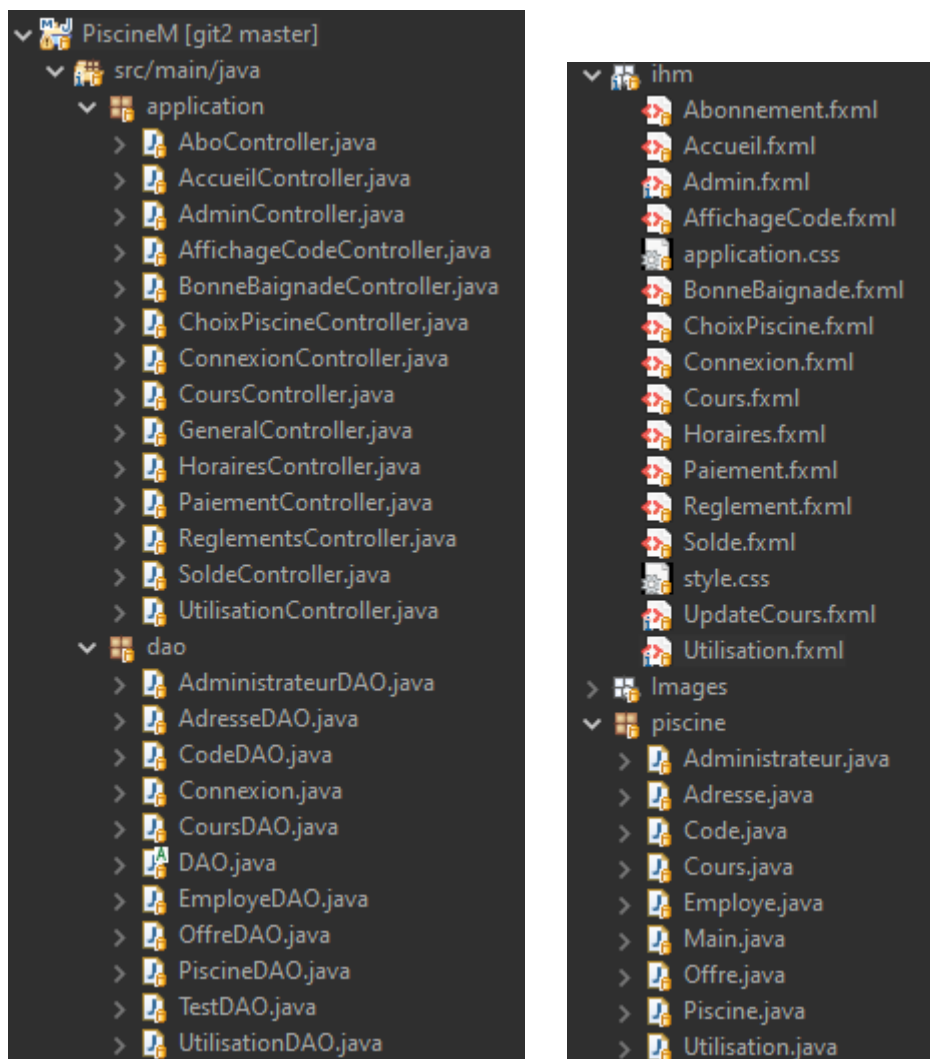
Utilisation :

	Nom de la colonne	Type de données	Autoriser les valeurs Null
🔑	dateUtilisation	datetime	<input type="checkbox"/>
🔑	idCode	nvarchar(50)	<input type="checkbox"/>
	idPiscine	int	<input type="checkbox"/>

MODÈLE VUE CONTRÔLEUR SUR ECLIPSE

Afin d'automatiser l'intégration continue lors du développement du logiciel, nous avons utilisé Apache Maven, qui est un outil de gestion et d'automatisation de production de projet logiciel Java.

Le modèle correspond aux packages « piscine » et « dao », la vue au package « ihm » et le contrôleur au package « application ». La connexion avec la base de données est gérée dans le package « dao ».

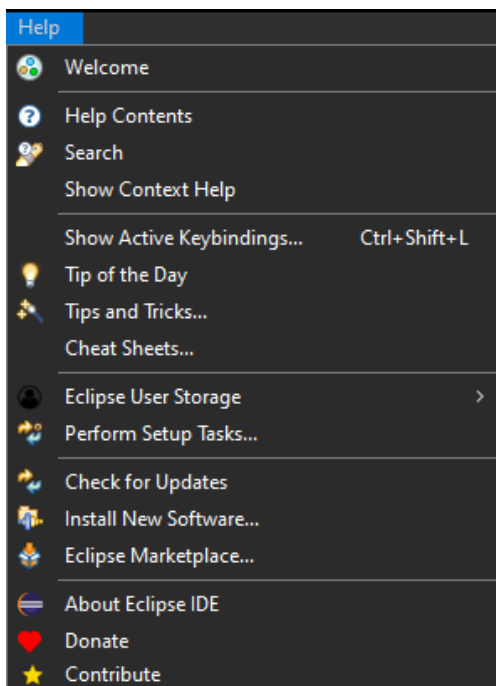


CONNECTION GITHUB

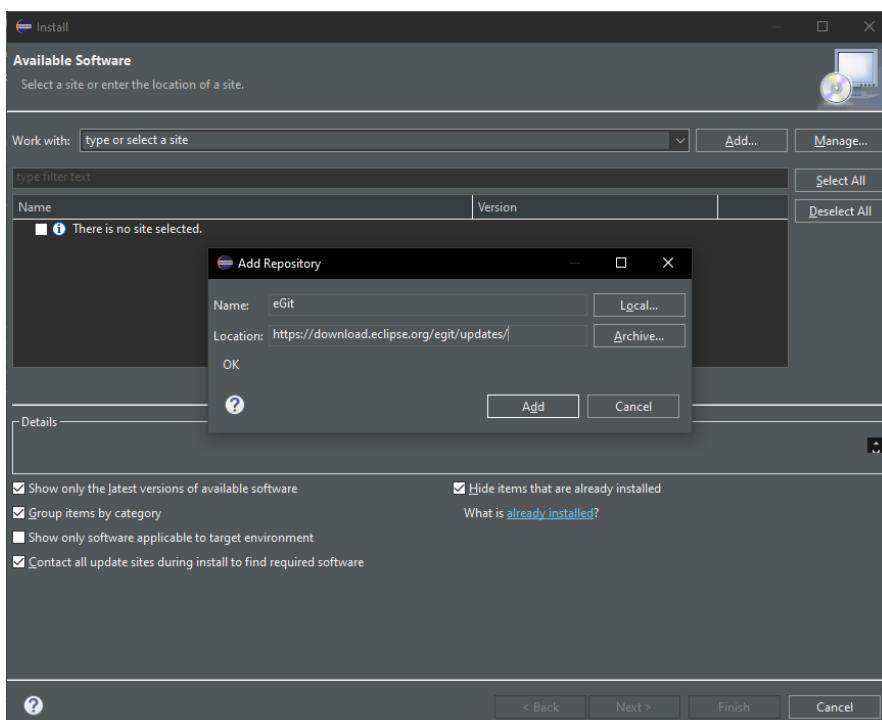
https://github.com/segoleneganzin/Project_JAVA

INSTALLER EGIT SUR ECLIPSE

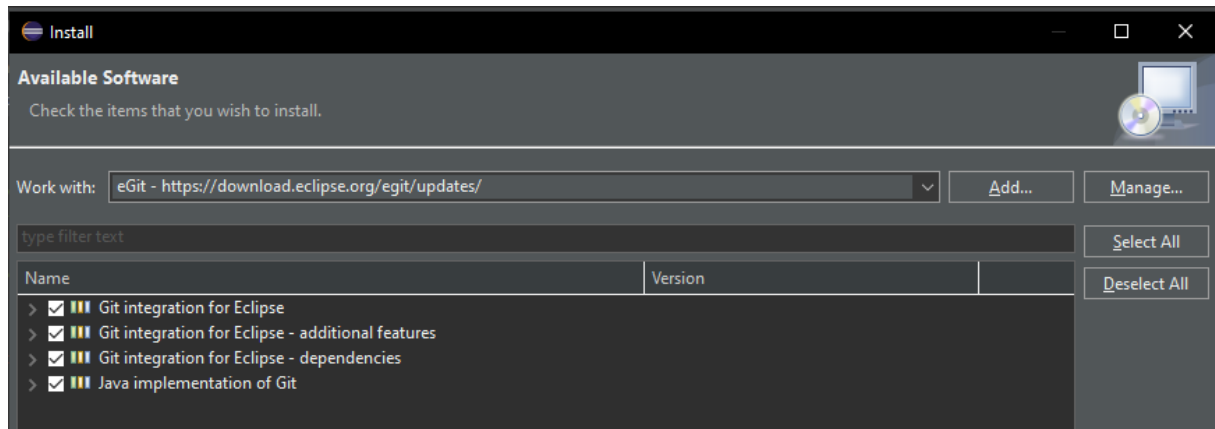
Help -> Install New Software



Cliquer sur add puis rentrer le nom et la location :



Ensuite cocher toutes les checkboxes :



Cliquer sur suivant et accepter les termes et conditions puis terminer.

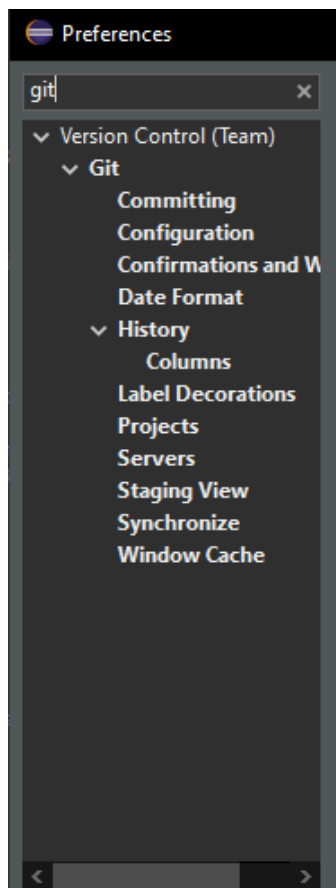
RELIER LE REPERTOIRE GITHUB

Au préalable, créer un repertoire sur son compte Github

CONFIGURER LE PLUGIN EGIT

Cliquer sur l'onglet window > preferences

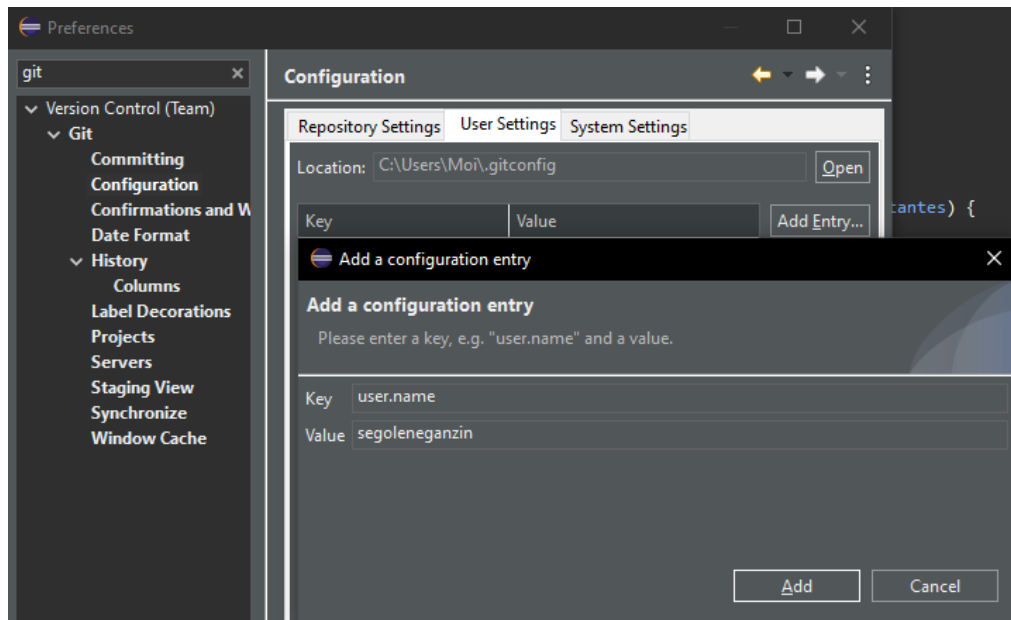
Dans la barre de recherche taper « git »



Choisir le path « Team > Git > Configuration »

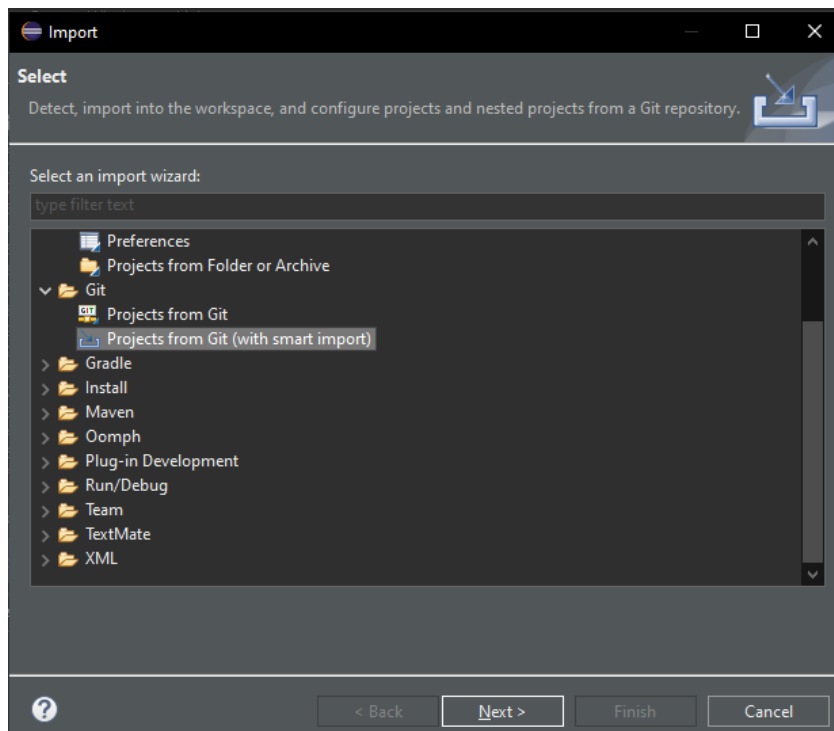
Key → user.name

Value → username Github



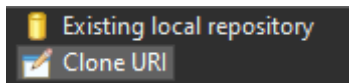
IMPORTER LE RÉPERTOIRE GITHUB DANS ECLIPSE

File > Import



Git > Projects from Git (with smart import)

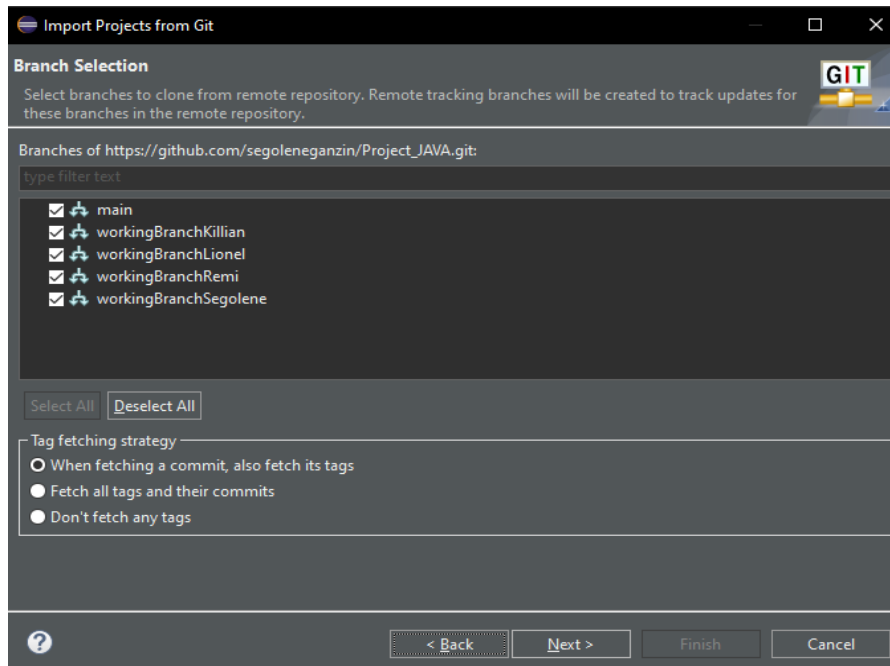
Cliquer sur Clone URI et suivant



Remplir les champs :

Le user et password sont ceux du compte github

Le lien se fait, on constate qu'il détecte bien les branches déjà créées :



Choisir « Import as general project »

Tous les fichiers modifiés dans ce dossier seront reliés à github

Donc il faut importer le zip du dossier piscine à l'intérieur du dossier Project_JAVA

PREMIER COMMIT

GÉNÉRER UNE CLÉ SUR GITHUB

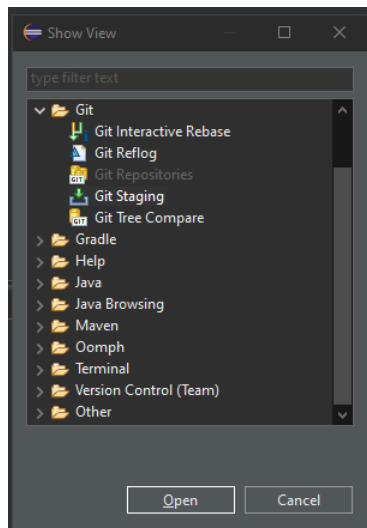
- Sur votre profil Github, cliquer sur l'onglet profil en haut à droite, puis **Settings > Developer settings > Personal access tokens**
- Cliquer sur **Generate new token** :
 - Note** : GitHub repo token
 - Expiration** → No expiration

Cocher le checkbox **repo**

- Cliquer sur **Generate token**
- Copier cette clé

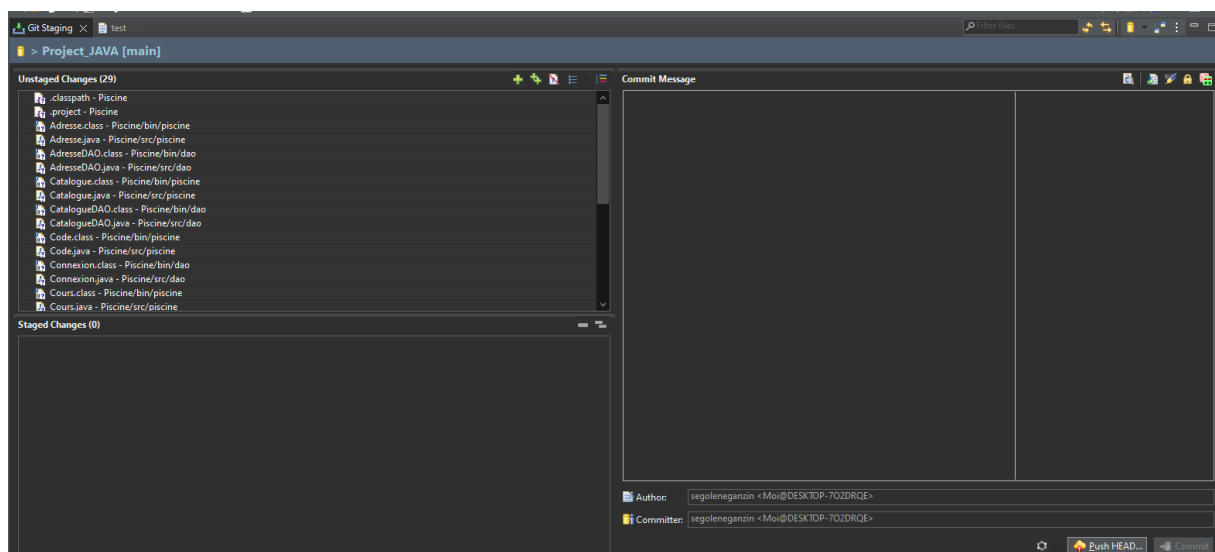
LIER LA CLÉ À ECLIPSE

Window > Show View > Other > Git Repositories



- Déplier l'onglet **Remotes**
- **Clique droit** sur **origin** (ou main selon son nom)
- Choisir **Configure Push...**
- Cliquer sur **Change...**
- Remplacer le **password** par la **clé GitHub**
- Cliquer sur **Finish** et **Save** pour appliquer les changements

Voici ensuite la vue GitStaging lorsque des fichiers sont ajoutés/modifiés/supprimés :

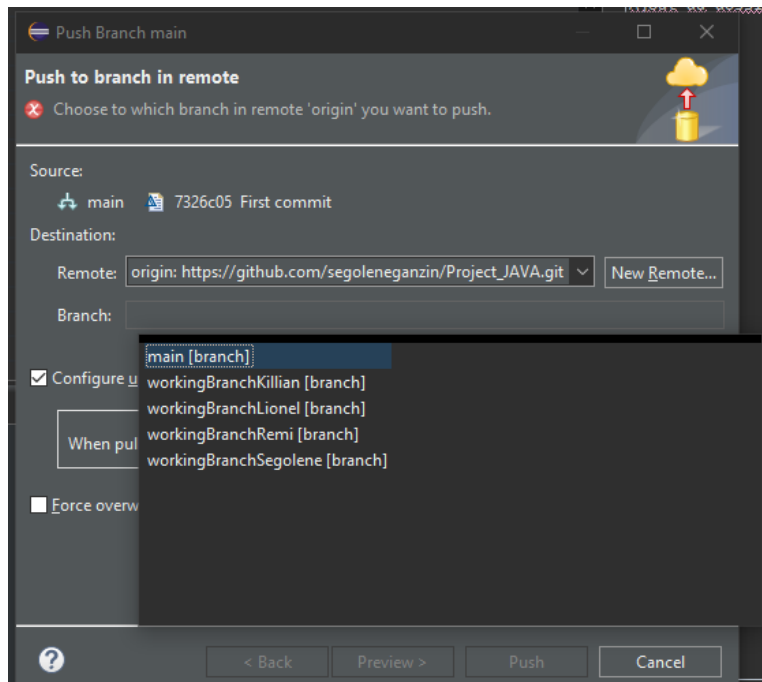


Cliquer sur les 2 plus de l'onglet « Unstaged Changes »

Ajouter un message de commit (Il faut toujours ajouter un message de commit !)

Cliquer sur Push HEAD

On choisit la branche où l'on souhaite faire le commit (ctrl+espace pour afficher les possibilités)



Ensuite on push

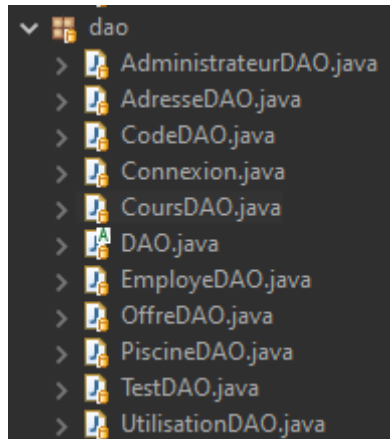
On vérifie que cela a fonctionné sur la plateforme GitHub :

segoleneganzin and segoleneganzin Ajout du dossier piscine		1f7b701 1 minute ago	🕒 5 commits
📁 Piscine	Ajout du dossier piscine		1 minute ago
📄 .project	First commit		15 minutes ago
📄 test	Update test		2 weeks ago

MODÈLE

Le modèle contient les fichiers permettant de se connecter à la base de données et de manipuler les données.

DAO (DATA ACCESS OBJECT)



Le package dao contient la classe nécessaire à la connexion à la base de données (Connexion.java) :

```
public class Connexion {  
  
    private static Connection connect = null;  
  
    private static final String SQL_SERVER = "localhost\\SQLEXPRESS";  
    private static final String BASE_DE_DONNEES = "piscine";  
    private static final String ID = "Projet";  
    private static final String MDP = "Tortue_Ninja";  
  
    /**  
     * Patron de conception Singleton  
     * @return l'instance unique de connexion  
     */  
    public static Connection getInstance() {  
        if (connect==null) {  
            try {  
  
                SQLServerDataSource ds = new SQLServerDataSource();  
                ds.setUser(ID);  
                ds.setPassword(MDP);  
                ds.setServerName(SQL_SERVER);  
                ds.setDatabaseName(BASE_DE_DONNEES);  
                connect = ds.getConnection();  
                System.out.println("Connecté");  
            }  
            catch (SQLException e){  
                System.out.println("Echec de la tentative de connexion : " + e.getMessage() + e.getStackTrace());  
            }  
        }  
        return connect;  
    }  
  
    private Connexion() {  
        super();  
    }  
}
```

Dans chaque classe DAO, qui fait la liaison avec les tables de Microsoft SQL Server, on définit les méthodes de création, de suppression, de mise-à-jour et de lecture des tables (CRUD).

Certaines classes ont des fonctions supplémentaires pour répondre aux besoins de l'application, comme le `readAllCoursDispo(Piscine piscine)`, de la classe `CoursDAO.java`, qui permet d'afficher l'ensemble des cours d'une piscine qui ont des places disponibles et sont postérieurs à « aujourd'hui ».

```
//selectionner les cours posterieurs a la date du jour, avec des places restantes
public List<Cours> readAllCoursDispo(Piscine piscine) {
    List<Cours> lesCours = new ArrayList<Cours>();
    try {
        String requete = "SELECT * FROM " + TABLE + " WHERE " + HORAIREDEBUT + " > GETDATE() AND " + PISCINE + " = " + piscine.getIdPiscine();
        ResultSet rs = Connexion.executeQuery(requete);
        while (rs.next()) {
            int idCours = rs.getInt(CLE_PRIMAIRE);
            Cours cours = CoursDAO.getInstance().read(idCours);
            int placesRestantes = cours.getPlacesRestantes();
            if (placesRestantes > 1) {
                lesCours.add(cours);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return lesCours;
}
```

LA CLASSE ABSTRAITE 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.

```
package dao;

import java.util.HashMap;

/**
 * Patron de conception DAO
 *
 * @param <T> avec type generique T (comme pour ArrayList<T>)
 */
public abstract class DAO<T> {

    protected final HashMap<Integer, T> donnees = new HashMap<Integer, T>();

    /**
     * Methode de creation d'un objet de type "T",
     * peut etre amene a injecter l'id cree dans le programme
     * @param obj
     * @return boolean
     */
    public abstract boolean create(T obj);

    /**
     * Methode pour effacer selon l'id de l'objet
     * @param obj
     * @return boolean
     */
    public abstract boolean delete(T obj);

    /**
     * Methode de mise a jour selon l'id de l'objet
     * @param obj
     * @return boolean
     */
    public abstract boolean update(T obj);

    /**
     * Methode de recherche des informations qui retourne un objet T
     * @param id
     * @return T
     */
    public abstract T read(int id);
}
```

PREPARE STATEMENT

Afin de sécuriser les requêtes SQL, on ne les envoie pas directement en clair dans la requête, on va les préparer. La donnée est envoyée en deux temps :

```
public Code read(String id) {
    Code code = null;
    try {
        String requete = "SELECT * FROM " + TABLE + " WHERE " + CLE_PRIMAIRE + " = ? ";
        PreparedStatement pst = Connexion.getInstance().prepareStatement(requete);
        pst.setString(1, id);
        pst.execute();
        ResultSet rs = pst.getResultSet();
        rs.next();
        LocalDateTime dateAchat = rs.getTimestamp(ACHAT).toLocalDateTime();
        LocalDateTime dateEcheance = rs.getTimestamp(ECHEANCE).toLocalDateTime();
        Offre idOffre = OffreDAO.getInstance().read(rs.getInt(OFFRE));
        List<Cours> lesCours = new ArrayList<Cours>();
        requete = "SELECT * FROM " + PARTICIPE + " WHERE " + ID_CODE_PARTICIPE + " = ? ";
        PreparedStatement pst2 = Connexion.getInstance().prepareStatement(requete);
        pst2.setString(1, id);
        pst2.execute();
        ResultSet rs2 = pst2.getResultSet();
        while (rs2.next()) {
            int idCours = rs2.getInt(ID_COURS_PARTICIPE);
            Cours cours = CoursDAO.getInstance().read(idCours);
            lesCours.add(cours);
        }
        code = new Code(id, dateAchat, dateEcheance, idOffre, lesCours);

    } catch (SQLException e) {
        System.out.println("code généré disponible");
    }
    return code;
}
```

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 boolean create(Piscine piscine) {
    boolean succes=true;
    try {
        Adresse adresse = piscine.getAdresse();

        String requete = "INSERT INTO "+TABLE+" (" +NOM+", "+HORAIREOUV+", "+HORAIREFERM+", "+ADRESSE+") VALUES (?, ?, ?, ?)";
        PreparedStatement pst = Connexion.getInstance().prepareStatement(requete, Statement.RETURN_GENERATED_KEYS);
        pst.setString(1, piscine.getNom());
        pst.setString(2, piscine.getHoraireOuv());
        pst.setString(3, piscine.getHoraireFerm());
        pst.setInt(4, adresse.getIdAdresse());
        pst.executeUpdate();
        ResultSet rs = pst.getGeneratedKeys();
        if (rs.next()) {
            piscine.setIdPiscine(rs.getInt(1));
        }
    } catch (SQLException e) {
        succes=false;
        e.printStackTrace();
        // selon les erreurs si clé étrangères inexistantes
        if (piscine.getAdresse().getIdAdresse() ==-1) {
            //afficher un message d'erreur
            System.out.println("Adresse inexistante");
        }
    }
    return succes;
}

```

GESTION DES TABLES D'ASSOCIATION

Nous avons deux tables d'association : participe et travail

La table travail est géré dans EmployeDAO.java

```

public class EmployeDAO extends DAO<Employe> {
    private static final String CLE_PRIMAIRE = "idEmp";
    private static final String TABLE = "employe";

    private static final String NOM = "nom";
    private static final String PRENOM = "prenom";
    private static final String FONCTION = "fonction";
    private static final String DATENAISSANCE = "dateNaissance";
    private static final String ADRESSE = "idAdresse";
    private static final String TRAVAILLEPOUR = "travail";
    private static final String ID_EMP_TRAVAILLEPOUR = "idEmp";
    private static final String ID_PISCINE_TRAVAILLEPOUR = "idPiscine";

    // Boucle pour sur la liste de piscines dans l'objet employé et insertion dans TRAVAILLEPOUR des id uniquement
    int idEmp = employe.getIdEmp();
    requete = "INSERT INTO " + TRAVAILLEPOUR + " (" + ID_EMP_TRAVAILLEPOUR + ", " + ID_PISCINE_TRAVAILLEPOUR + ") VALUES (?, ?)";
    for (Piscine piscine : employe.getLesPiscines()) {
        pst = Connexion.getInstance().prepareStatement(requete);
        pst.setInt(1, idEmp);
        pst.setInt(2, piscine.getIdPiscine());
        pst.executeUpdate();
    }
}

```

La table participe est gérée dans CodeDAO.java

```
public class CodeDAO extends DAO<Code> {
    private static final String CLE_PRIMAIRE = "idCode";
    private static final String TABLE = "code";
    private static final String ACHAT = "dateAchat";
    private static final String ECHEANCE = "dateEcheance";
    private static final String OFFRE = "idOffre";
    private static final String PARTICIPE = "participe";
    private static final String ID_CODE_PARTICIPE = "idCode";
    private static final String ID_COURS_PARTICIPE = "idCours";
}
```

```
//ajouter une participation a un cours
public boolean ajouterParticipation(Cours cours, Code code) {
    boolean succes=true;
    try {
        String requete = "INSERT INTO "+PARTICIPE+" (" +ID_CODE_PARTICIPE+", "+ID_COURS_PARTICIPE+" ) VALUES (?, ?)";
        PreparedStatement pst = Connexion.getInstance().prepareStatement(requete, Statement.RETURN_GENERATED_KEYS);
        pst.setString(1, code.getIdCode());
        pst.setInt(2, cours.getIdCours());
        pst.executeUpdate();
    } catch (SQLException e) {
        succes=false;
        e.printStackTrace();
    }
    return succes;
}
```

CRÉATION DU CODE

GÉNÉRATION DU CODE

L'idCode est composé de 10 caractères (minuscules, majuscules et chiffres) aléatoires :

```
public static String generateRandomCode(){
    // Gamme ASCII - alphanumérique (0-9, a-z, A-Z)
    final String chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    SecureRandom random = new SecureRandom();
    StringBuilder sb = new StringBuilder();
    // chaque itération de la boucle choisit aléatoirement un caractère parmi les données
    // Plage ASCII et l'ajoute à l'instance `StringBuilder`
    for (int i = 0; i < 10; i++)
    {
        int randomIndex = random.nextInt(chars.length());
        sb.append(chars.charAt(randomIndex));
    }
    return sb.toString();
}
```

DATE D'ÉCHÉANCE D'UN CODE

La date d'échéance est automatiquement calculée en fonction de la durée de validité de l'offre sélectionnée :

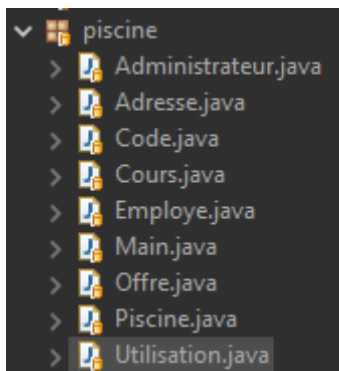
```
public static LocalDateTime dateEcheance(Offre offre) {
    LocalDateTime dureeVal = LocalDateTime.now().plusMonths(offre.getValidite()).plusDays(1);
    return dureeVal;
}
```

On y ajoute un jour pour que le nombre de jours accessibles correspondent bien à celui indiqué dans l'offre et que l'accès à la piscine soit toujours possible le dernier jour de validité.

Ces deux fonctions sont utilisées lors de la création du code :

```
// CREATE
public boolean create(Code code) {
    boolean succes = true;
    String generatedCode = generateRandomCode();
    try {
        String requete = "INSERT INTO " + TABLE + " (" + CLE_PRIMAIRE + ", " + ACHAT + ", " + ECHEANCE + ", " + OFFRE + ") VALUES (?, ?, ?, ?)";
        PreparedStatement pst = Connexion.getInstance().prepareStatement(requete);
        LocalDateTime dureeDeValidite = dateEcheance(code.getOffre());
        code.setIdCode(generatedCode);
        pst.setString(1, code.getIdCode());
        pst.setObject(2, code.getDateAchat());
        code.setDateEcheance(dureeDeValidite);
        pst.setObject(3, code.getDateEcheance());
        pst.setInt(4, code.getOffre().getIdOffre());
        pst.executeUpdate();
    } catch (SQLException e) {
        succes = false;
        e.printStackTrace();
        // recuper les erreurs si clé étrangères inexistantes
        if (code.getOffre().getIdOffre() == -1) {
            System.out.println("Offre inexistante");
        }
    }
    return succes;
}
```

PISCINE (MÉTIER)



Chaque classe métier contient ses attributs, ses constructeurs, ses getters et setters ainsi que les fonctions nécessaires au besoin de l'application.

Il y a au moins deux constructeurs par classe, un sans l'id pour le create car cet id est généré automatiquement à la création du tuple dans la base de données. Et un constructeur avec l'id pour la manipulation de l'objet.

Exemple sur la classe Cours.java :

```

public class Cours {
    private int idCours;
    private String intitule;
    private LocalDateTime horaireDebut;
    private LocalDateTime horaireFin;
    private int nombrePlacesInitiales;
    private Employee employee; //cls STANESSE
    private Piscine piscine; //cls STANESSE
    // private List<Code> lesCodes = new ArrayList<Code>(); //lien tables d'association "participe"

    //constructeur sans idCours :
    public Cours(String intitule, LocalDateTime horaireDebut, LocalDateTime horaireFin, int nombrePlacesInitiales, Employee employee, Piscine piscine) {
        super();
        this.horaireFin = horaireFin;
        this.horaireDebut = horaireDebut;
        this.intitule = intitule;
        this.nombrePlacesInitiales = nombrePlacesInitiales;
        this.employee = employee;
        this.piscine = piscine;
    }

    //constructeur avec idCours :
    public Cours(int idCours, String intitule, LocalDateTime horaireDebut, LocalDateTime horaireFin, int nombrePlacesInitiales, Employee employee, Piscine piscine) {
        super();
        this.idCours = idCours;
        this.horaireFin = horaireFin;
        this.horaireDebut = horaireDebut;
        this.intitule = intitule;
        this.nombrePlacesInitiales = nombrePlacesInitiales;
        this.employee = employee;
        this.piscine = piscine;
    }
}

```

CALCUL DU SOLDE

```

//consulter le solde du code
public int getSoldeCode() {
    String idCode = this.getIdCode();
    // System.out.println(idCode);
    int nombreUtilisation = UtilisationDAO.getInstance().getNombreUtilisation(idCode);
    // System.out.println(nombreUtilisation);
    Offre offre = this.getOffre();
    int nbPlaces = offre.getNbPlaces();
    // System.out.println(nbPlaces);
    int solde = nbPlaces - nombreUtilisation;
    return solde;
}

```

Dans la classe Code.java, on récupère le nombre d'entrée dans la table « utilisation » ayant l'id du code et on soustrait ce nombre au nombre de places initiales de l'abonnement.

CALCUL DES PLACES RESTANTES À UN COURS

```

public int getPlacesRestantes() {
    int idCours = this.getIdCours();
    int nombreParticipant = CodeDAO.getInstance().getNombreParticipant(idCours);
    int nbPlacesInitiales = this.getNombrePlacesInitiales();
    int nbPlacesRestantes = nbPlacesInitiales - nombreParticipant;
    return nbPlacesRestantes;
}

```

Dans la classe Cours.java on calcule le nombre d'entrée dans la table « participe ». Ces entrées sont créées dès l'achat d'un code pour un cours.

La fonction getNombreParticipant() est composée d'une requête « count * » dans la table d'association « participe ».

Le nombre de places restantes correspond au nombre de places initiales d'un cours moins le nombre de participants.

AFFICHAGE DES DATES

Pour afficher les dates dans un format adapté à la vue, on a mis en place des fonctions qui les transforment en chaîne de caractères :

```
//pour afficher les dates "joliment" :  
public String toStringDate() {  
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("EEEE dd MMMM yyyy");  
    return horaireDebut.format(formatter);  
}  
  
public String toStringHoraireDebut() {  
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm");  
    return horaireDebut.format(formatter);  
}  
  
public String toStringHoraireFin() {  
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm");  
    return horaireFin.format(formatter);  
}
```

classe Cours.java

```
public String toStringDateAchat() {  
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd MMMM yyyy");  
    return dateAchat.format(formatter);  
}  
  
public String toStringDateEcheance() {  
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd MMMM yyyy");  
    return dateEcheance.format(formatter);  
}
```

classe Code.java

VUE

JAVA FX

INSTALLATION

JavaFX est une plate-forme de développement d'interface utilisateur graphique (GUI) pour Java qui offre des fonctionnalités avancées pour créer des applications interactives pour des applications de bureau, mobiles et web. Nous avons choisi JavaFX pour notre projet car il permet de créer des interfaces utilisateur modernes, dynamiques et personnalisables facilement avec des outils simples. L'installation de JavaFX est simple, en téléchargeant et en installant la dernière version du kit de développement Java SE (JDK) et en ajoutant les bibliothèques JavaFX à notre projet en configurant les propriétés du projet, il est cependant important lorsque l'on travaille en mode projet de tous être sur la même version de JavaFX afin de ne pas avoir d'erreurs de compatibilité.

UTILISATION



FIGURE A : EXEMPLE DE PAGE AVEC SCENE BUILDER

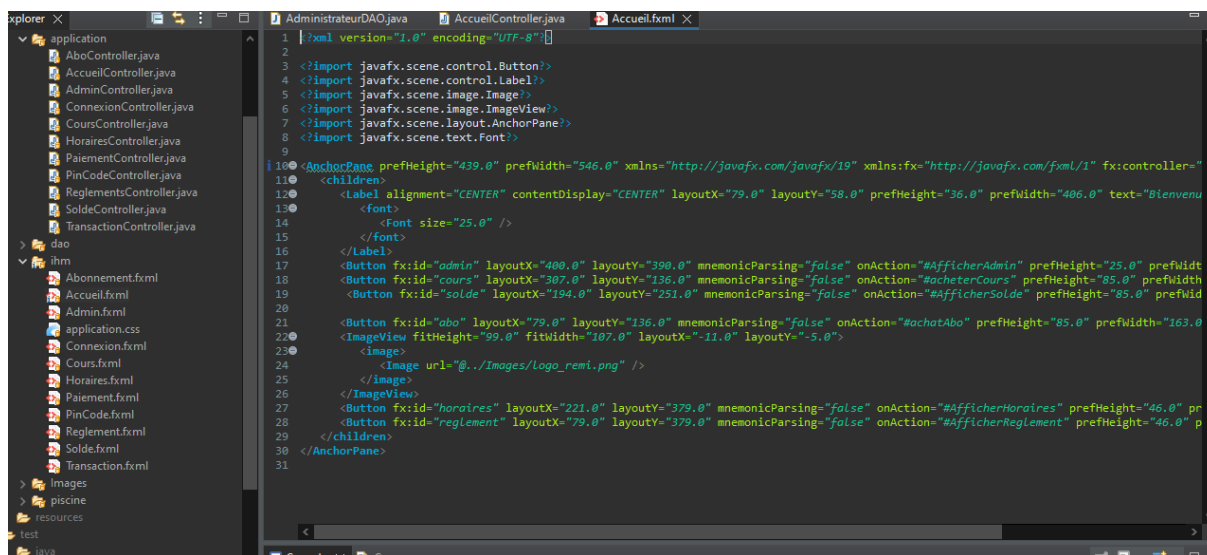


FIGURE B : ACCUEIL.FXML

```

@FXML
private Button solde;

@FXML
void AfficherSolde(ActionEvent event) {
    try {
        Parent root = FXMLLoader.load(getClass().getResource("../ihm/Solde.fxml"));
        Scene scene = new Scene(root);
        Main.stage.setScene(scene);
        Main.stage.show();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

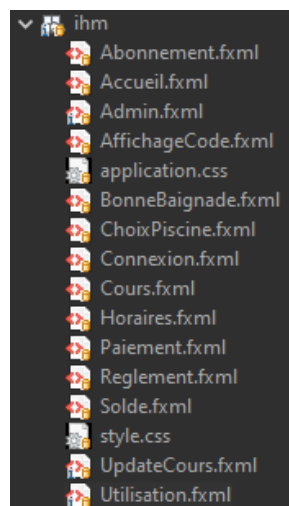
@FXML
private Button reglement;

```

FIGURE C : EXEMPLE DE CODE POUR AFFICHER LE SOLDE

Ici afin d'associer une fonction à une action (exemple ici au clic d'un bouton), il faut renseigner le nom de la fonction du contrôleur dans « On Action » ainsi que « solde » qui correspond au bouton ci-dessus dans fx:id

IHM

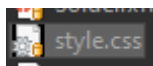


Les vues sont générées par des fichiers au format fxml et sont adaptées avec le logiciel Scene Builder.

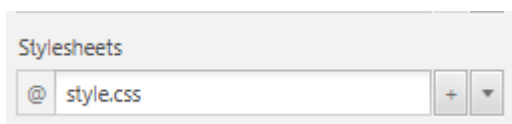
Il y a un fichier pour chaque vue.

FICHER CSS

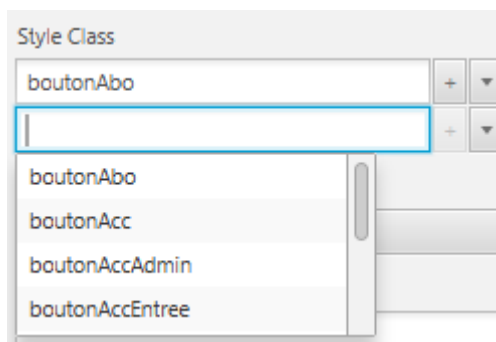
Un fichier style.css permet de centraliser le code css et uniformiser le visuel.



On applique ce fichier à la fenêtre d'une vue :

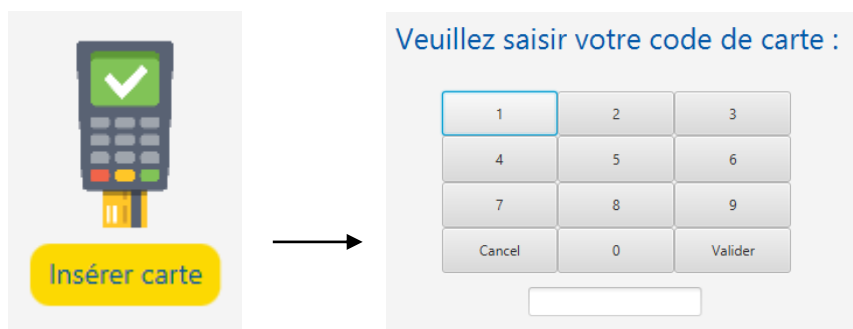


Dès que l'on sélectionne un élément de la page on nous propose tous les styles définis dans le fichier style.css :

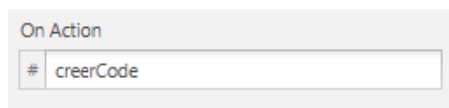


FENÊTRES « POPUP »

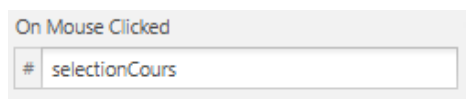
La vue paiement contient la saisie du code de carte bleue dans un pane qui apparaît au clique sur « insérer carte ».



Pour générer des événements au clic sur un bouton on utilise « On Action » sur Scene Builder qui va générer la fonction appropriée.



Pour les observables de la vue Cours.fxml on paramètre une fonction pour répondre au clique sur une ligne du tableau :



Intitulé	Date	Debut	Fin
Natation	vendredi 28 juillet 2023	15:00	16:00
Natation	vendredi 28 juillet 2023	16:00	17:00
Aquagym	vendredi 28 juillet 2023	16:00	17:00

Une fois la ligne cliquée on affiche un pane qui récapitule le cours sélectionné et permet de le réserver et payer :

Cours sélectionné :

Intitulé :

Natation

Date :

vendredi 28 juillet 2023

Heure de début :

15:00

Heure de fin :

16:00

Tarif d'un cours : 20€

Annuler

Réserver et payer

MESSAGES PERSONNALISÉS

Lors du test du code dans la page « consulter sole » différents messages peuvent apparaître :

Si le code est valide :

Valider

Date d'achat : 09 mai 2023

Date d'échéance : 10 mars 2024

Modalité de l'offre : solo

Nombre d'entrée(s) restante(s) : 9

Valider

Offre correspondante : cours

Date d'achat : 09 mai 2023

Intitulé	Date	Debut	Fin	Piscine
Aquagym	vendredi 28 juillet 2023	16:00	17:00	Vanocea

Valider

Code expiré le 01 mai 2023

Pour la simulation d'entrée c'est la même chose :

Pour accéder au bassin veuillez saisir votre code ou scanner votre carte :

Valider

Votre abonnement a expiré le 01 mai 2023

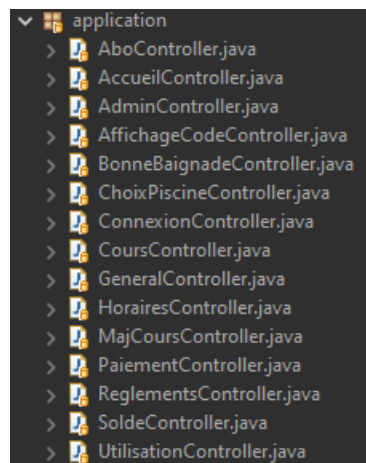
Valider

Votre code n'est valable que pour un cours

Vous n'avez plus d'entrées disponibles

CONTRÔLEUR

APPLICATION



Le package application contient tous les contrôleurs nécessaires pour interagir avec la vue et le métier.

GENERALCONTROLLER

Ce contrôleur est la classe mère des autres classes de l'application.

Il permet de stocker la fonction retour qui est utilisée dans la plupart des contrôleurs.

```
//fonction de retour a la page d'accueil
//stockee ici pour ne pas la repeter dans tous les controller
@FXML
void retourAcc(ActionEvent event) {
    try {
        Parent root = FXMLLoader.Load(getClass().getResource("../ihm/Accueil.fxml"));
        Scene scene = new Scene(root);
        Main.stage.setScene(scene);
        Main.stage.show();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Il permet aussi de stocker l'objet Piscine afin de pouvoir le récupérer dans les classes qui en ont besoin, en les « extends » :

```
public void setLaPiscine(Piscine laPiscine) {
    GeneralController.laPiscine = laPiscine;
}

//recupere la piscine courante pour qu'elle puisse etre appele depuis controleurs enfants :
public void setInfoPiscine(Piscine unePiscine) {
    setLaPiscine(unePiscine);
}
```



```
public class AboController extends GeneralController{

    Piscine laPiscine = GeneralController.getLaPiscine();
```

OBSERVABLES

```
private ObservableList<Cours> coursData = FXCollections.observableArrayList();

public ObservableList<Cours> getCoursData() {
    return coursData;
}
```

On crée une liste des cours sélectionnés avec la fonction « readAllCoursDispo » (cours postérieur à la date du jour, avec des places disponibles ».

On envoie ensuite cette liste dans l'observable « coursData », après l'avoir vidé.

Puis on envoie les données dans le tableau, on envoie l'observable dans le tableau « tableCours ».

```
Offre uneOffre = OffreDAO.getInstance().readModalite("cours");
tarifCours.setText(String.valueOf(uneOffre.getTarif()) + "€");
tarif.setVisible(true);
tableCours.setVisible(true);
intitule.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getIntitule()));
date.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().toStringDate()));
heureDebut.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().toStringHoraireDebut()));
heureFin.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().toStringHoraireFin()));
CoursDAO coursDAO = CoursDAO.getInstance();
List<Cours> lesCours = coursDAO.readAllCoursDispo(laPiscine);
coursData.clear();
coursData.addAll(lesCours);
tableCours.setItems(coursData);
```

À chaque clique sur un tuple cela ouvre un « pop up » en récupérant l'objet concerné, que l'on récupère avec cette fonction :

```
void selectionCours() {
    // Récupérer la ligne sélectionnée
    coursSelectionne = tableCours.getSelectionModel().getSelectedItem();
    System.out.println(coursSelectionne);
    Cours cours = coursSelectionne;
    if (cours != null) {
        reserverCours.setVisible(true);
        intituleCours.setText(cours.getIntitule());
        dateCours.setText(cours.toStringDate());
        horaireDebut.setText(cours.toStringHoraireDebut());
        horaireFin.setText(cours.toStringHoraireFin());
    }
}
```