# Marieteam Java - Documentation

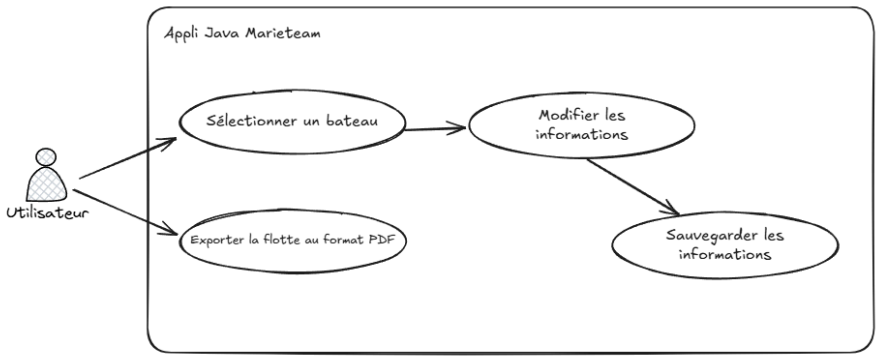


## Contexte

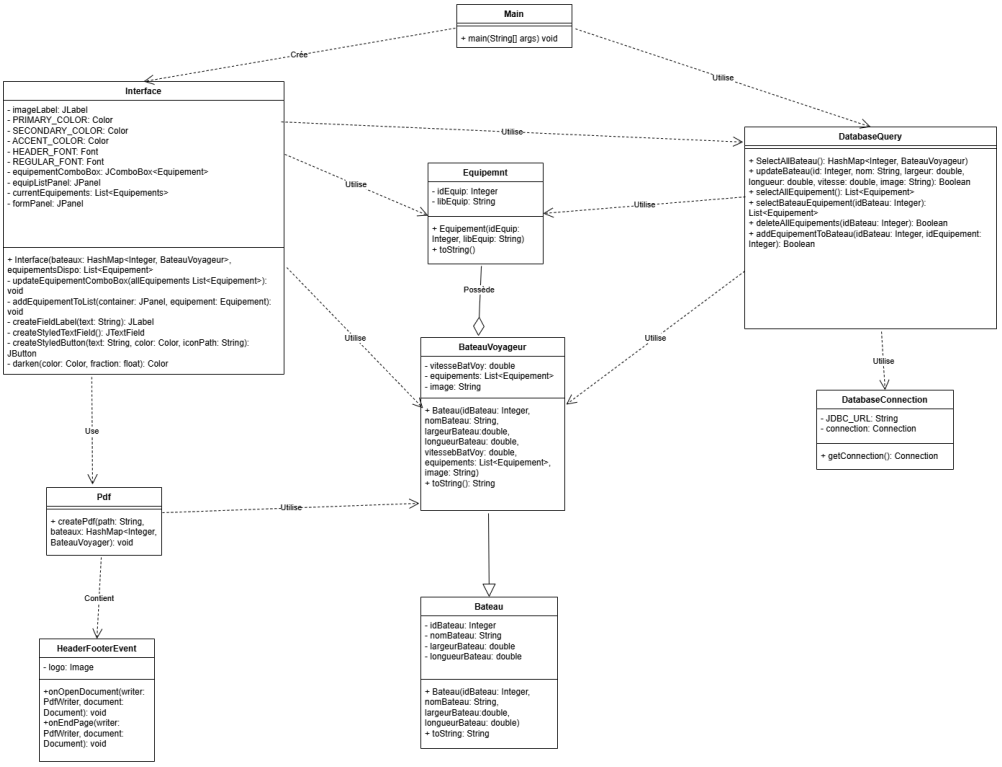
Dans le cadre de l'épreuve E6, il fallait réaliser un client lourd permettant l'édition de brochure récapitulant la flotte de bateaux de Marieteam. Il fallait aussi permettre la modification par les utilisateurs, des informations d'un bateau. Le projet devait utiliser la même base de données que l'application web, de même que fournir une interface graphique pour l'utilisateur.

## I. Les technologies

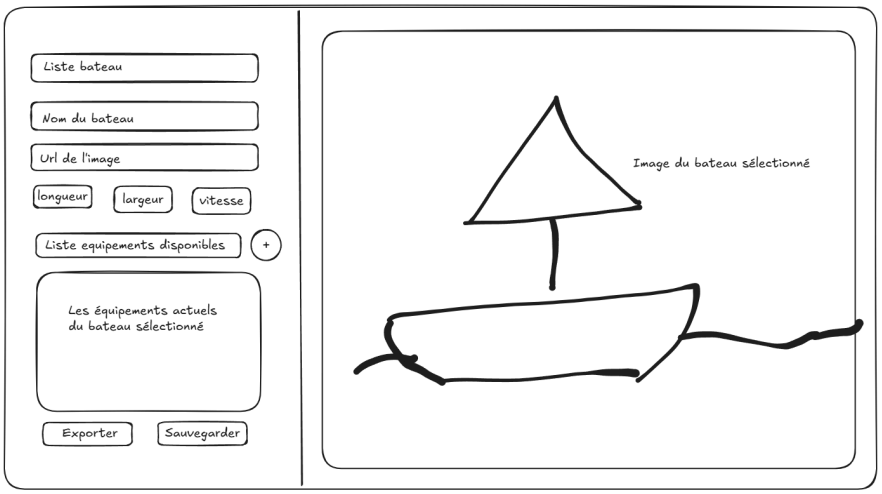
Le projet est développé en Java. Il utilise les bibliothèques ItextPDF pour la création de PDF et Swing pour les interfaces, Junit pour les test unitaires, postgresql pour faire les requêtes à la bdd.




## II. Diagramme de classe




# III. Interface




L’interface est géré dans la classe Interface. La librairie Swing est utilisé.

Voici comment on la déclare dans la méthode Main:

```
// === LANCEMENT DE L'INTERFACE GRAPHIQUE ===
// SwingUtilities.invokeLater() garantit que l'interface graphique sera créée
// sur l'Event Dispatch Thread (EDT), ce qui est obligatoire pour Swing
SwingUtilities.invokeLater(new Runnable() {
    @Override
    public void run() {
        // Création et affichage de la fenêtre principale de l'application
        // Passage des données chargées (bateaux et équipements) au constructeur
        new Interface(finalBateaux, equipementDispo);
```

```
    }
});
```

Pour instancier une Interface, il faut mettre en arguement une HashMap de bateaux et une List d'équipement. Dans notre cas dans la méthode Main, le finalBateaux est la totalité de la flotte en base de données et la liste equipementDispo est la liste de tous les équipements disponibles en base de données.

Ils sont instanciés ainsi:

```
// === CHARGEMENT DES BATEAUX DEPUIS LA BASE DE DONNÉES ===
// HashMap pour stocker les bateaux avec leur ID comme clé
HashMap<Integer, BateauVoyageur> bateaux = new HashMap<>();

// Récupération de tous les bateaux depuis la base de données
bateaux = databaseQuery.SelectAllBateau();

// Création d'une référence finale pour utilisation dans la classe anonyme
// (nécessaire car les variables locales utilisées dans les classes anonymes
// doivent être finales ou effectivement finales)
HashMap<Integer, BateauVoyageur> finalBateaux = bateaux;

// === CHARGEMENT DES ÉQUIPEMENTS DISPONIBLES ===
// Récupération de la liste de tous les équipements depuis la base de données
// Méthode statique, pas besoin d'instance de DatabaseQuery
List<Equipement> equipementDispo = DatabaseQuery.SelectAllEquipement();
```

(Pour plus de précision sur databaseQuery, voir la partie sur la base de données)

# IV. Base de données

La base de données est la même que le projet web, celle sur Supabase. Pour y accèder et faire des requêtes dessus, il a fallu installer la dépendance postgresql pour Java.

On effectue sa connexion dans la classe DatabaseConnection de la manière suivant:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

/**
 * Classe utilitaire pour gérer la connexion à la base de données
 *
 * Cette classe implémente le pattern SINGLETON pour la gestion de connexion
 * à une base de données PostgreSQL hébergée sur Supabase.
 *
 * PATTERN SINGLETON : Assure qu'une seule instance de connexion existe
 * dans toute l'application, évitant ainsi la création multiple de connexions
 * coûteuses en ressources.
 */
public class DatabaseConnection {

    // === CONFIGURATION DE LA BASE DE DONNÉES ===

    /**
     * URL de connexion JDBC vers la base de données PostgreSQL
     *
```

```
     * Structure de l'URL :
     * - jdbc:postgresql:// : Protocole JDBC pour PostgreSQL
     * - aws-0-eu-west-2.pooler.supabase.com:6543 : Serveur et port Supabase
     * - postgres : Nom de la base de données
     * - user=postgres.vlqfkkxckflqiqdhjaur : Nom d'utilisateur
     * - password=MarieteamEpreuve : Mot de passe
     *
     */
    private static final String JDBC_URL = "jdbc:postgresql://aws-0-eu-west-2.pooler.supabase.com:6543/pos
tgres?user=postgres.vlqfkkxckflqiqdhjaur&password=MarieteamEpreuve";

    // === INSTANCE SINGLETON ===

    /**
     * Instance unique de la connexion à la base de données
     *
     * - static : Appartient à la classe, pas aux instances
     * - Connection : Type d'objet représentant une connexion JDBC
     * - Initialement null, sera créée lors du premier appel à getConnection()
     */
    private static Connection connection;

    // === MÉTHODE D'ACCÈS À LA CONNEXION ===

    /**
     * Méthode statique pour obtenir la connexion à la base de données
     *
     * Implémente le pattern LAZY INITIALIZATION :
     * - La connexion n'est créée que lors du premier appel
     * - Les appels suivants réutilisent la même connexion
     * - Vérifie si la connexion existe et est toujours active
     *
     * @return Connection L'objet de connexion à la base de données
     * @throws SQLException Si une erreur survient lors de la connexion
     *         (serveur inaccessible, identifiants incorrects, etc.)
     */
    public static Connection getConnection() throws SQLException {

        // Vérification de l'état de la connexion
        // Deux conditions pour créer/recréer une connexion :
        // 1. connection == null : Aucune connexion n'a encore été créée
        // 2. connection.isClosed() : La connexion existante a été fermée
        if (connection == null || connection.isClosed()) {

            // Création d'une nouvelle connexion via DriverManager
            // DriverManager.getConnection() :
            // - Parse l'URL JDBC
            // - Charge le driver PostgreSQL automatiquement (JDBC 4.0+)
            // - Établit la connexion avec les paramètres fournis
            // - Peut lever une SQLException en cas d'échec
            connection = DriverManager.getConnection(JDBC_URL);
        }

        // Retourne la connexion (nouvelle ou existante)
        return connection;
```

```
    }
}
```

Ensuite dans la classe DatabaseQuery, on peut y déclarer toutes les requêtes nécessaires à l'appli Java, voici par exemple la requête pour avoir tous les bateaux:

```
/**
 * Récupère tous les bateaux de la base de données avec leurs équipements.
 *
 * Cette méthode effectue une requête complexe qui :
 * 1. Sélectionne tous les bateaux dans la table "bateau".
 * 2. Pour chaque bateau, récupère ses équipements via SelectBateauEquipment().
 * 3. Construit des objets BateauVoyageur complets.
 * 4. Les stocke dans une HashMap avec l'ID comme clé.
 *
 * @return HashMap<Integer, BateauVoyageur> Collection de bateaux indexée par ID.
 *         Retourne une HashMap vide en cas d'erreur ou si aucun bateau n'existe.
 */
public static HashMap<Integer, BateauVoyageur> SelectAllBateau() {
    // Initialisation de la collection de résultats
    HashMap<Integer, BateauVoyageur> bateaux = new HashMap<>();

    try {
        // Obtention de la connexion via la classe DatabaseConnection
        Connection conn = DatabaseConnection.getConnection();
        System.out.println("✅ Connexion réussie à Supabase !");

        // Création d'un Statement pour exécuter la requête
        Statement statement = conn.createStatement();

        // Requête SQL simple pour récupérer tous les bateaux
        String query = "SELECT * FROM bateau";

        // Exécution de la requête et récupération des résultats
        ResultSet resultSet = statement.executeQuery(query);

        try {
            // Parcours de tous les résultats
            // Vérification défensive : resultSet != null (bonne pratique)
            while (resultSet != null && resultSet.next()) {
                // Pour chaque bateau, récupération de ses équipements
                // Appel à une autre méthode de cette classe
                List<Equipement> equipements = SelectBateauEquipment(resultSet.getInt("id"));

                // Construction de l'objet BateauVoyageur avec toutes ses données
                BateauVoyageur bateauVoyageur = new BateauVoyageur(
                        resultSet.getInt("id"),          // ID du bateau
                        resultSet.getString("nom"),      // Nom du bateau
                        resultSet.getDouble("largeur"),  // Largeur en mètres
                        resultSet.getDouble("longueur"), // Longueur en mètres
                        resultSet.getInt("vitesse"),     // Vitesse (conversion int vers double automatique)
                        equipements,                     // Liste des équipements
                        resultSet.getString("ImageUrl")  // URL de l'image
                );
```

```
            // Ajout du bateau à la HashMap avec son ID comme clé
            bateaux.put(resultSet.getInt("id"), bateauVoyageur);
        }
    } catch (SQLException e) {
        // Gestion des erreurs lors du traitement des résultats
        System.out.println("❌ Erreur de connexion ou d'exécution de la requête : " + e.getMessage());
        e.printStackTrace();
    }

    // Fermeture de la connexion
    // Note : Peut poser problème avec le pattern Singleton de DatabaseConnection
    conn.close();

} catch (SQLException e) {
    // Gestion des erreurs de connexion initiale
    System.out.println("❌ Erreur de connexion : " + e.getMessage());
    e.printStackTrace();
}

return bateaux;
}
```

Dans la classe Main, il faut bien déclarer une instance de DatabaseQuery pour permettre à l'appli de faire des requêtes:

```
DatabaseQuery databaseQuery = new DatabaseQuery();
```

Dans la méthode main par exemple on s'en sert comme suit:

```
HashMap<Integer, BateauVoyageur> bateaux = new HashMap<>();
bateaux = databaseQuery.SelectAllBateau();
HashMap<Integer, BateauVoyageur> finalBateaux = bateaux;
List<Equipement> equipementDispo = DatabaseQuery.SelectAllEquipement();
```

# V. Création de PDF

La création d'un pdf contenant toute la flotte Marieteam est gérée par la classe Pdf. Elle est appelée dans la classe Interface, lors de l'évenement click sur le bouton créer un pdf.

```
// --- GESTIONNAIRE D'EXPORT PDF ---
buttonPdf.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Définir le chemin de sauvegarde sur le bureau
        String userHome = System.getProperty("user.home");
        String filePath = userHome + "/Desktop/FlotteMarieteam.pdf";

        try {
            // Création du PDF via la classe Pdf
            Pdf.createPdf(filePath, bateaux);

            // Demander à l'utilisateur s'il veut ouvrir le PDF
            int option = JOptionPane.showOptionDialog(
```

```
            Interface.this,
            "<html><div style='font-family:Segoe UI;font-size:12pt;padding:10px'>" +
                "PDF créé avec succès!<br>Souhaitez-vous ouvrir le document?</div></html>",
            "Export réussi",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,
            new Object[]{"Ouvrir", "Fermer"},
            "Ouvrir"
        );

        // Ouvrir le PDF si l'utilisateur le souhaite
        if (option == JOptionPane.YES_OPTION) {
            if (Desktop.isDesktopSupported()) {
                Desktop.getDesktop().open(new File(filePath));
            }
        }

    } catch (Exception exception) {
        // Affichage d'erreur en cas de problème
        JOptionPane.showMessageDialog(
            Interface.this,
            "<html><div style='font-family:Segoe UI;font-size:12pt;padding:10px'>" +
                "Erreur lors de la création du PDF:<br>" + exception.getMessage() + "</div></html>",
            "Erreur",
            JOptionPane.ERROR_MESSAGE
        );
        exception.printStackTrace();
    }
  }
});
```

La méthode prend en argument un chemin pour enregistrer le fichier et une HashMap contenant les bateaux. Ici le fichier est automatiquement enregistré sr le bureau avec:

```
String userHome = System.getProperty("user.home");
String filePath = userHome + "/Desktop/FlotteMarieteam.pdf";
```

Ici on prend en argument la HashMap de bateaux que l'on a récupérée lors de l'instanciation de notre interface. A noter que si elle est mise à jour par l'utilisateur, alors celle prise en compte sera elle aussi à jour pour la création du pdf.

# VI. Mettre à jour un bateau

Un utilisateur peut modifier les informations d'un bateau et les enregistrer en base de données. Cette logique est contenu dans Interface et fait appel à la requête update contenu dans DatabaseQuery.

Voici la méthode pour prendre en compte les nouvelles informations, vérifiées au préalable:

```
// --- GESTIONNAIRE DE SAUVEGARDE ---
buttonSave.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        DatabaseQuery databaseQuery = new DatabaseQuery();
        BateauVoyageur b = (BateauVoyageur) comboBox.getSelectedItem();

        // Vérifier qu'un bateau est sélectionné
```

```
        if (b == null) return;

        // Récupération des valeurs du formulaire
        String nomBateau = nameField.getText().trim();

        try {
            // Conversion et validation des valeurs numériques
            Double longueurBateau = Double.parseDouble(longueurField.getText().trim());
            Double largeurBateau = Double.parseDouble(largeurField.getText().trim());
            Double vitesseBatVoy = Double.parseDouble(vitesseField.getText().trim());
            String image = urlField.getText().trim();

            // Validation des données
            if (nomBateau.isEmpty() || longueurBateau < 0 || largeurBateau < 0 || vitesseBatVoy < 0) {
                JOptionPane.showMessageDialog(
                    Interface.this,
                    "<html><div style='font-family:Segoe UI;font-size:12pt;padding:10px'>" +
                        "Veuillez remplir tous les champs.</div></html>",
                    "Erreur",
                    JOptionPane.ERROR_MESSAGE
                );
                return;
            }

            // MISE À JOUR EN BASE DE DONNÉES
            Boolean update = databaseQuery.updateBateau(
                b.getIdBateau(),
                nomBateau,
                largeurBateau,
                longueurBateau,
                vitesseBatVoy,
                image
            );

            if (update) {
                // Mise à jour de l'objet en mémoire
                b.setNomBateau(nomBateau);
                b.setLongueurBateau(longueurBateau);
                b.setLargeurBateau(largeurBateau);
                b.setVitesseBatVoy(vitesseBatVoy);
                b.setImage(image);

                // GESTION DES ÉQUIPEMENTS
                // Supprimer tous les équipements existants pour ce bateau
                databaseQuery.deleteAllEquipements(b.getIdBateau());

                // Ajouter les nouveaux équipements
                boolean allEquipmentsSaved = true;
                for (Equipement equip : currentEquipements) {
                    boolean saved = databaseQuery.addEquipementToBateau(
                        b.getIdBateau(),
                        equip.getIdEquip()
                    );
                    if (!saved) {
                        allEquipmentsSaved = false;
```

```
                }
            }

            // Mettre à jour la liste des équipements dans l'objet
            b.setEquipements(new ArrayList<>(currentEquipements));

            // Mettre à jour l'objet dans la HashMap principale
            bateaux.put(b.getIdBateau(), b);

            // Rafraîchir la ComboBox pour afficher le nouveau nom
            int selectedIndex = comboBox.getSelectedIndex();
            comboBox.removeItemAt(selectedIndex);
            comboBox.insertItemAt(b, selectedIndex);
            comboBox.setSelectedIndex(selectedIndex);

            // Message de confirmation
            String message = "<html><div style='font-family:Segoe UI;font-size:12pt;padding:10px'>" +
                    "Les modifications ont été sauvegardées.";

            if (!allEquipmentsSaved) {
                message += "<br><br><span style='color:orange'>Attention: Certains équipements n'ont pas pu
être sauvegardés.</span>";
            }

            message += "</div></html>";

            JOptionPane.showMessageDialog(
                Interface.this,
                message,
                "Sauvegarde",
                JOptionPane.INFORMATION_MESSAGE
            );
        } else {
            // Erreur de sauvegarde
            JOptionPane.showMessageDialog(
                Interface.this,
                "<html><div style='font-family:Segoe UI;font-size:12pt;padding:10px'>" +
                        "Une erreur est survenue lors de la sauvegarde des modifications.</div></html>",
                "Erreur",
                JOptionPane.ERROR_MESSAGE
            );
        }
    } catch (NumberFormatException ex) {
        // Erreur de format des nombres
        JOptionPane.showMessageDialog(
            Interface.this,
            "<html><div style='font-family:Segoe UI;font-size:12pt;padding:10px'>" +
                    "Veuillez entrer des valeurs numériques valides pour la longueur, largeur et vitesse.</div></ht
ml>",
            "Erreur",
            JOptionPane.ERROR_MESSAGE
        );
    }
```

```
    }
});
```

L’appel à la base de données se fait ici :

```
Boolean update = databaseQuery.updateBateau(b.getIdBateau(), nomBateau, largeurBateau, longueurBateau, vitesseBatVoy, image);
```

Et dans la classe DatabaseQuery est utilisé de ce fait cette méthode:

```
/**
 * Met à jour les informations d'un bateau existant
 *
 * @param id       Identifiant du bateau à modifier
 * @param nom      Nouveau nom du bateau
 * @param largeur  Nouvelle largeur du bateau
 * @param longueur Nouvelle longueur du bateau
 * @param vitesse  Nouvelle vitesse du bateau
 * @param image    Nouvelle URL d'image du bateau
 * @return Boolean true si la mise à jour a réussi (au moins une ligne modifiée),
 *                 false en cas d'erreur ou si aucune ligne n'a été modifiée
 */
public static Boolean updateBateau(int id, String nom, double largeur, double longueur, double vitesse, String image) {
    try {
        // Obtention de la connexion
        Connection conn = DatabaseConnection.getConnection();
        System.out.println("✅ Connexion réussie à Supabase !");
        System.out.println(id); // Debug : affichage de l'ID

        String query = "UPDATE bateau SET nom = '" + nom +
                    "', largeur = " + largeur +
                    ", longueur = " + longueur +
                    ", vitesse = " + vitesse +
                    ", ImageUrl = '" + image + "'" +
                    " WHERE id = " + id;

        try {
            // Utilisation de Statement au lieu de PreparedStatement (non sécurisé)
            Statement statement = conn.createStatement();

            // Exécution de la requête de mise à jour
            // executeUpdate() retourne le nombre de lignes affectées
            int rowsAffected = statement.executeUpdate(query);

            conn.close(); // Fermeture de la connexion

            // Retourne true si au moins une ligne a été modifiée
            return rowsAffected > 0;

        } catch (SQLException e) {
            System.err.println("❌ Erreur lors de la mise à jour : " + e.getMessage());
            e.printStackTrace();
```

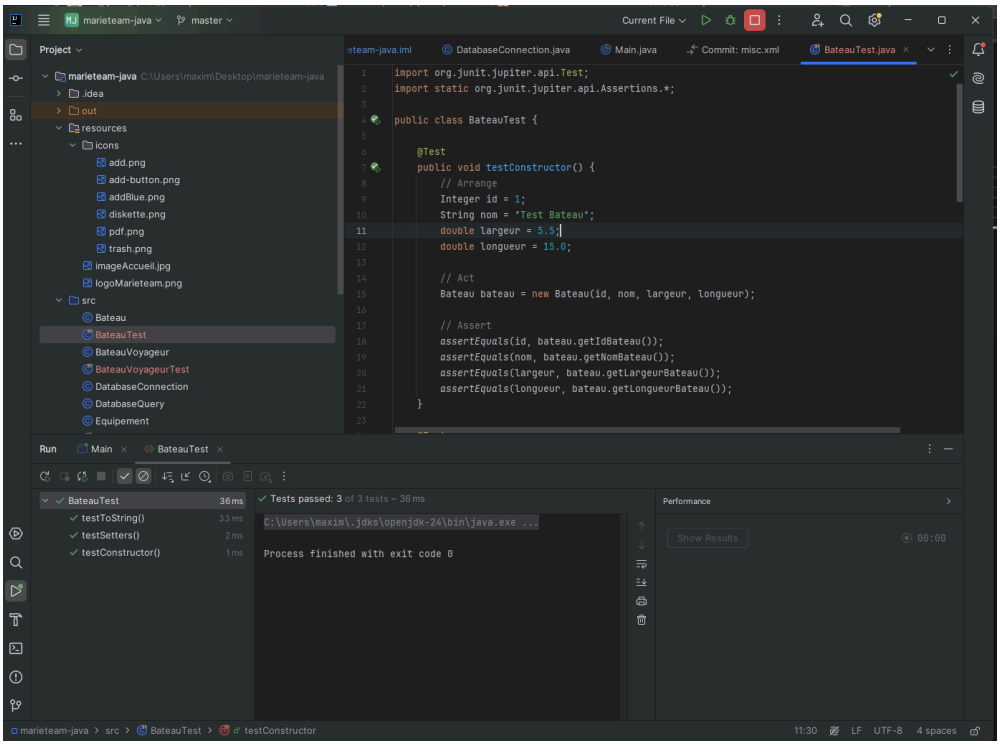
```
            return false; // Échec de l'opération
        }

    } catch (SQLException e) {
        System.out.println("❌ Erreur de connexion : " + e.getMessage());
        e.printStackTrace();
        return false;
    }
}
```

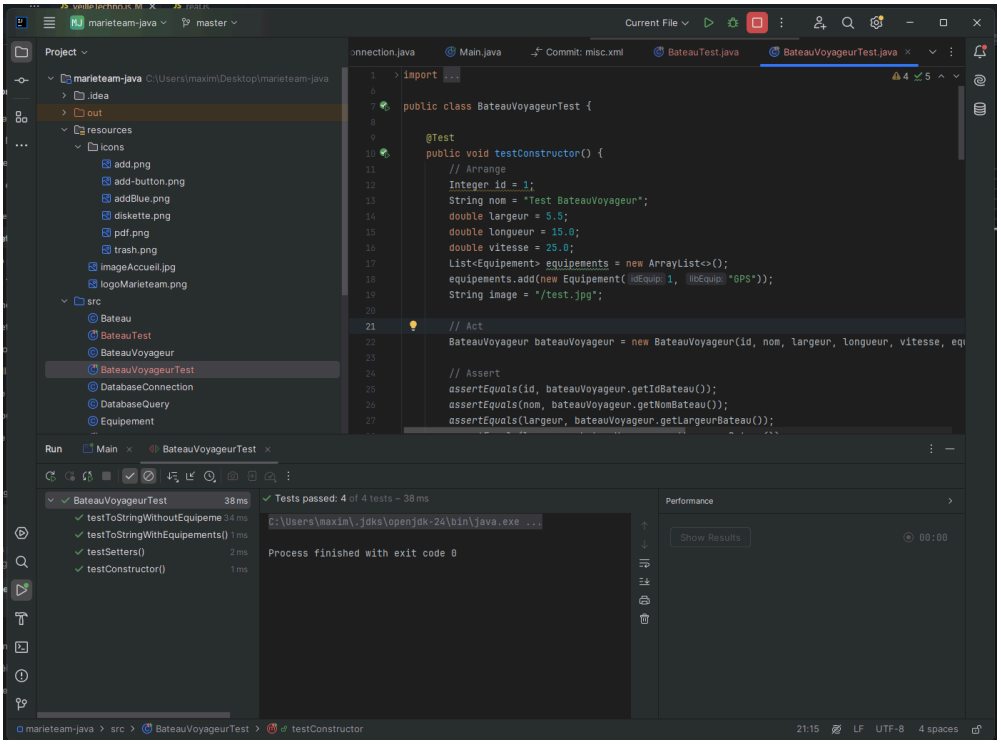
# VI. Test Unitaires

Voici l’ensemble des tests unitaires réalisés avec Junit:

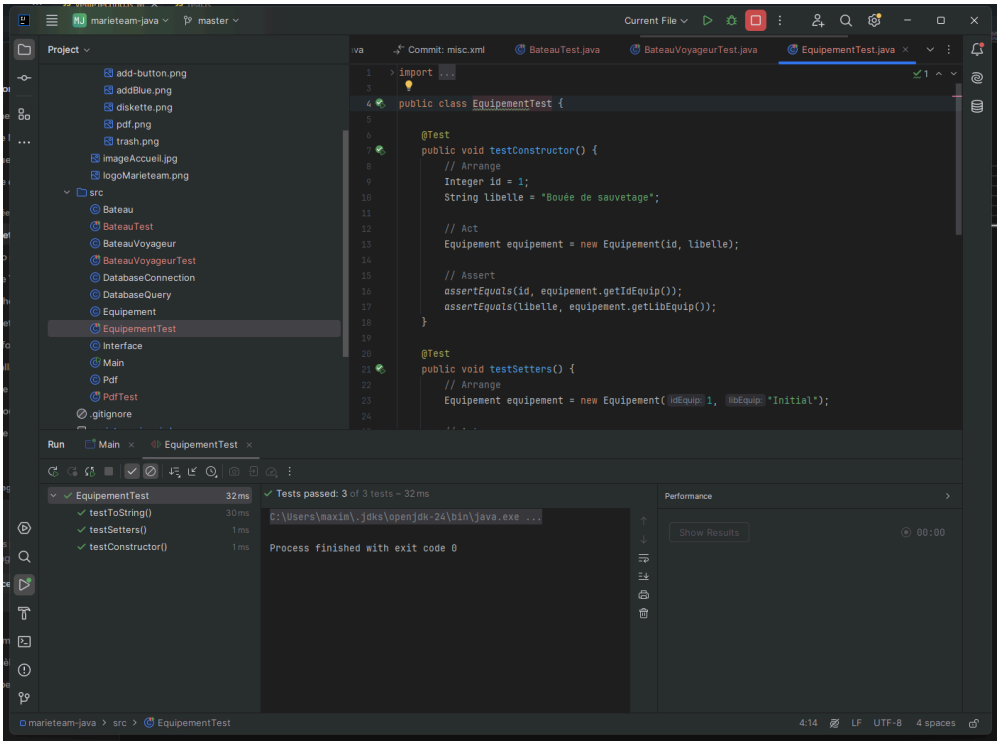
BateauTest:



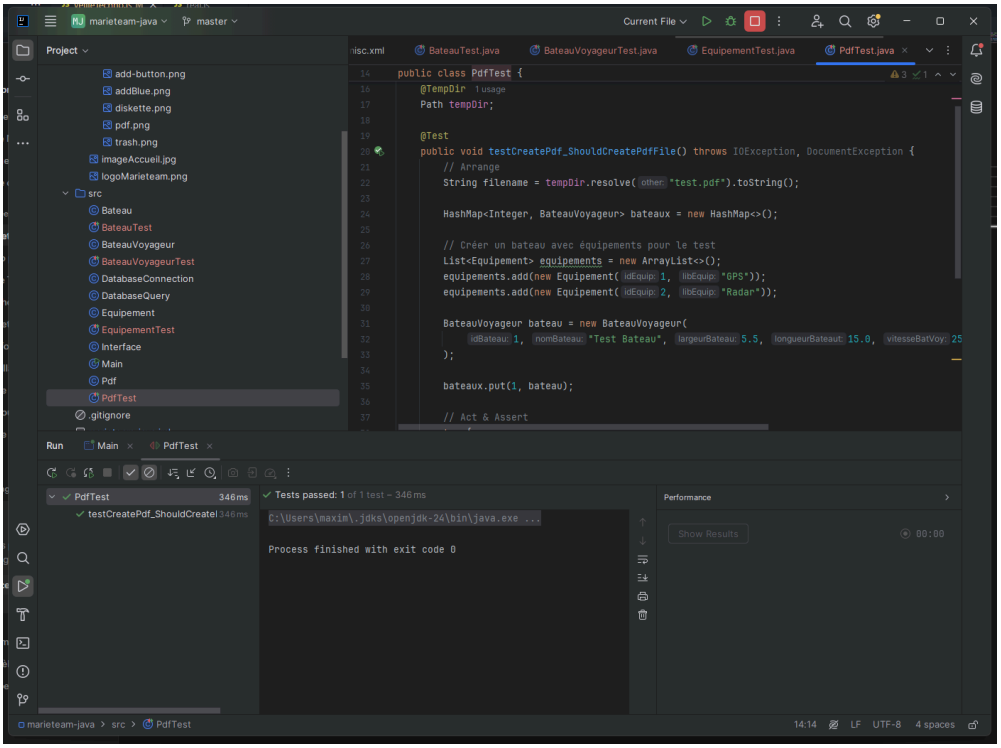

BateauVoyageurTest:




EquipementTest:




PdfTest:


marieteam-java
master
Current File
Project
add-button.png
addBlue.png
diskette.png
pdf.png
trash.png
imageAccueil.jpg
logoMarieteam.png
src
Bateau
BateauTest
BateauVoyageur
BateauVoyageurTest
DatabaseConnection
DatabaseQuery
Equipement
EquipementTest
Interface
Main
Pdf
PdfTest
.gitignore
BateauTest.java
BateauVoyageurTest.java
EquipementTest.java
PdfTest.java
public class PdfTest {
@TempDir 1 usage
Path tempDir;
@Test
public void testCreatePdf_ShouldCreatePdfFile() throws IOException, DocumentException {
// Arrange
String filename = tempDir.resolve( other: "test.pdf").toString();
HashMap<Integer, BateauVoyageur> bateaux = new HashMap<>();
// Créer un bateau avec équipements pour le test
List<Equipement> equipements = new ArrayList<>();
equipements.add(new Equipement( idEquip: 1, libEquip: "GPS"));
equipements.add(new Equipement( idEquip: 2, libEquip: "Radar"));
BateauVoyageur bateau = new BateauVoyageur(
idBateau: 1, nomBateau: "Test Bateau", largeurBateau: 5.5, longueurBateau: 15.0,
);
bateaux.put(1, bateau);
// Act & Assert
Run
Main
PdfTest
PdfTest 346 ms
testCreatePdf_ShouldCreatel 346 ms
Tests passed: 1 of 1 test – 346 ms
C:\Users\maxim\.jdks\openjdk-24\bin\java.exe ...
Process finished with exit code 0
Performance
Show Results
00:00
marieteam-java › src › PdfTest
14:14
LF
UTF-8
4 spaces