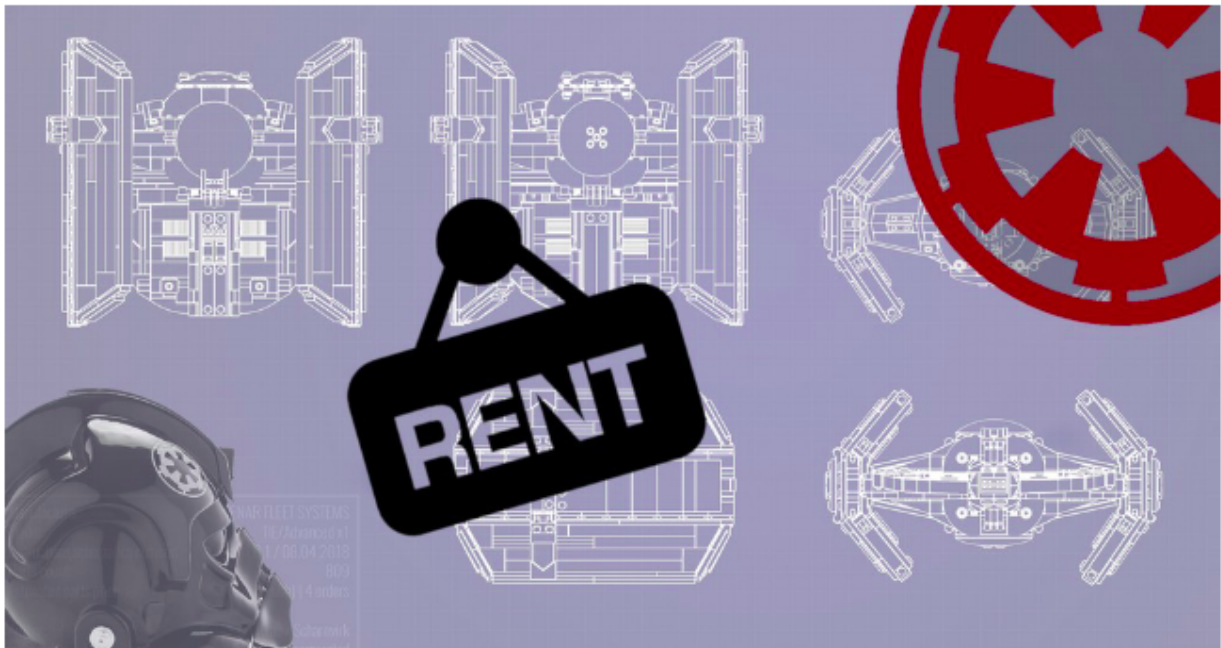


RAPPORT PROJET GÉNIE LOGICIEL

RENTaTIE4J



GHOUIRI Mazir GI4

LEMBA Mohamed GI4

TATAH NDOME Wilfried GI4

BOUAPHAKEO Benoit GI1

SELAMNIA Isra GI3

2021/2022



Sommaire

Sommaire	1
I) INTRODUCTION	2
La répartition des tâches	2
Les choix techniques & outils de travail avec leur justification:	3
II) CAHIER DES CHARGES	4
III) DIAGRAMMES UML	5
Diagramme des cas d'utilisation	5
Diagramme de classe	6
IV) MODÈLE CONCEPTUEL DES DONNÉES	7
V) IMPLÉMENTATION DES CLASSES ET DE LA BASE DE DONNÉES	8
VI) TESTS UNITAIRES	12
VII) INTERFACE GRAPHIQUE	17
La classe Controller	17
Les méthodes de Controller	18
Les actions de certains éléments (Relations Controleur - Vue)	19
La gestion des requêtes dans les controllers (Relations Contrôleur - Modèle)	19
Les requêtes de lectures de données	19
Les requêtes d'écritures / de modifications	20
Les contrôleurs qui implémentent Initializable / cacher des éléments	22
VIII) APPLICATION FINALE	23
IV) CONCLUSION	39

I) INTRODUCTION

Dans le cadre de notre première année de cycle ingénieur en Génie Informatique, il nous a été demandé de réaliser un projet de groupe autour du sujet de la Programmation Orientée Objet (POO). Notre groupe est donc composé de GOUHIRI Mazir (GI4), TATAH NDOME Wilfried (GI4), SELAMNIA Isra (GI3), BOUAPHAKEO Benoit (GI3) et LEMBA Mohamed (GI4), et l'ensemble du projet est encadré par Monsieur Renaud VERIN.

Parmi les sujets que nous pouvions choisir, nous avons choisi le sujet de Rent A Tie. Nous avons fait ce choix car le sujet nous intéressait et était vraiment adapté pour les personnes ayant une faible expérience dans le domaine de la POO.

L'objectif de ce rapport est donc de présenter de manière concise le travail réalisé au cours de ces derniers mois. Une première partie de ce rapport traitera de la partie conception du projet en présentant brièvement une version réduite et en français du cahier des charges, puis les diagrammes UML pertinents pour la réalisation de la version finale de notre projet.

Une seconde partie du rapport présentera les détails techniques, avec le modèle conceptuel de données, l'implémentation des classes et de la BDD, les tests unitaires, l'interface graphique, et l'application finale.

La répartition des tâches

- Réalisation de la partie conception:** Tout le monde
- Gestion de la base de données:** Benoit, Mazir
- Gestion de la partie Modèle de l'application:** Benoit, Isra
- Gestion des tests:** Isra, Mazir
- Gestion du cryptage des mots de passe:** Mazir
- Gestion du design de l'interface:** Isra, Wilfried
- Gestion de la partie vue de l'application:** Mohamed, Wilfried
- Gestion de la partie Contrôleur de l'application:** Mohamed, Wilfried

Les choix techniques & outils de travail avec leur justification:

-**IntelliJ** : Environnement de travail utilisé par la majeure partie des membres de l'équipe.

-**SceneBuilder** : Permet de réaliser les fichiers de vue FXML et de gérer avec une interface graphique les interactions entre "contrôleur/vue".

-**Architecture MVC** : Architecture que nous avons déjà étudiée en cours de IHM, et qui correspond aux besoins demandés par le projet.

-**JUnit (version 5)** : Sur conseil de notre tuteur, nous avons utilisé JUnit pour les tests sur le modèle.

-**bibliothèque JavaFX** : Bibliothèque graphique que nous avons étudiée en cours d'IHM cette année.

-**Star UML** : Utilisé pour générer la documentation de la partie conception, avant la partie réalisation.

-**plug-in IntelliJ diagrams** : Utilisé pour générer les diagrammes UML finaux de l'application, permet d'avoir une bonne cohérence entre le code et le diagramme final.

-**jdbc-connector** : Situé dans la catégorie "possibles ressources" dans le sujet de l'application, utilisé pour les connexions à la base de données.

-**phpMyadmin** : Utilisé pour la gestion des bases de données, c'est le seul SGBD qui a déjà été utilisé par les différents membres de l'équipe.

-**Git** : Utilisé pour les dépôts de fichiers.

-**classe BCrypt** : Classe récupérée sur Internet, utilisée pour la gestion des cryptages de mots de passe. (Voir commentaires dans la classe du projet pour la source)

II) CAHIER DES CHARGES

Dans l'optique d'aider les officiers de l'empire à gérer les locations de Tie Fighters et leurs pilotes, nous avons développé une application reliée à une base de données.

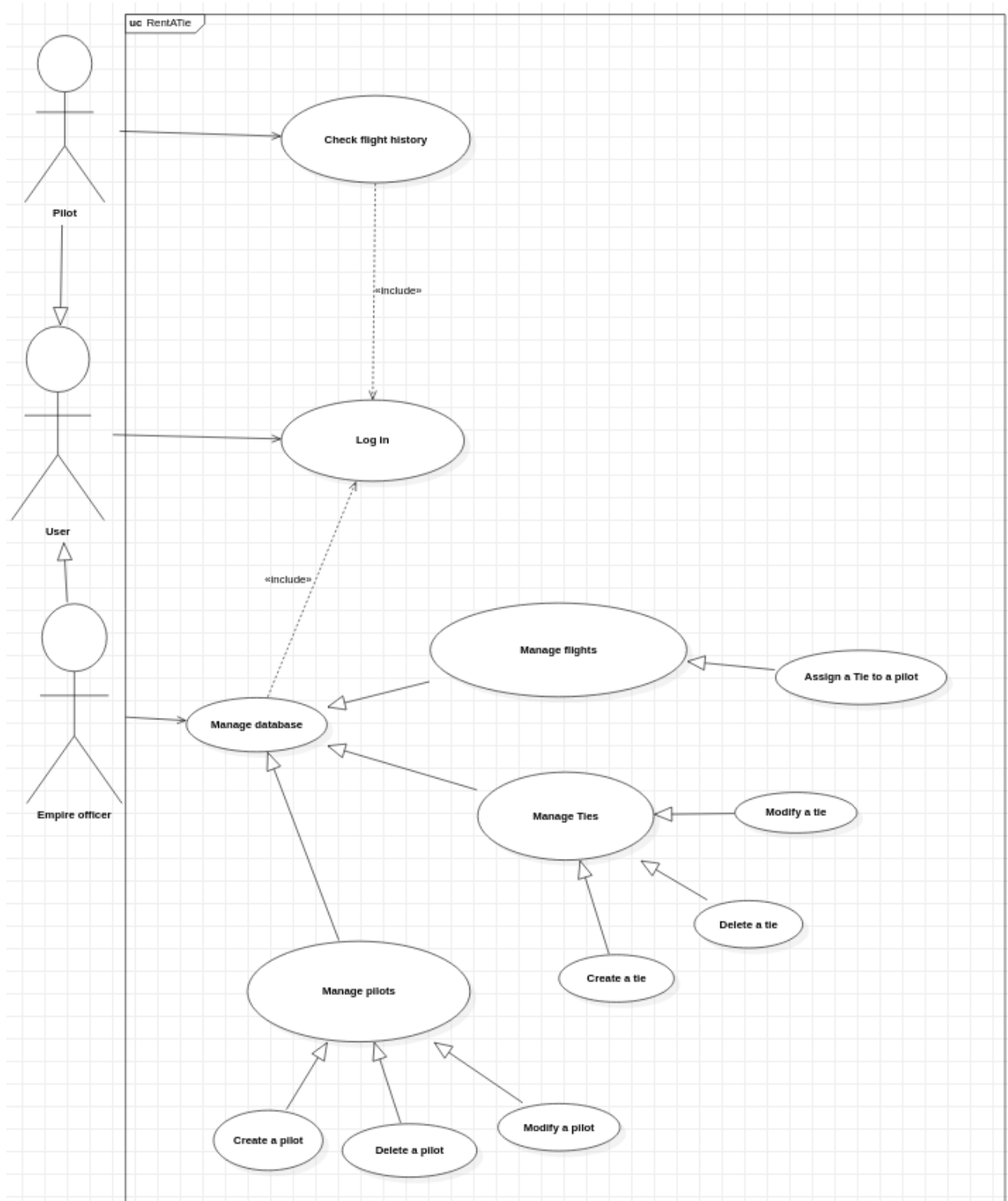
Dans un premier temps, nous avons à construire les différents diagrammes UML ainsi que le modèle conceptuel des données afin d'avoir une vision globale du projet. Puis, nous devons coder les diagrammes de classe en Java et également créer une base de données contenant les différents acteurs de l'empire.

Notre application doit pouvoir gérer les exceptions sans crasher. L'accès à l'application nécessite une authentification sécurisée. L'utilisateur peut être un pilote ou un officier.

Un officier peut accéder aux données de la base de données triées par préférence tandis qu'un pilote ne peut que accéder à son historique de vols. L'officier peut également changer ou supprimer des données. Il doit être possible de voir la durée d'un vol ; si un pilote dépasse la date de rendu du vaisseau, l'officier a le choix de le punir de la peine de mort.

III) DIAGRAMMES UML

a. Diagramme des cas d'utilisation



Le diagramme de cas d'utilisation est un outil qui permet de représenter les différentes actions qu'un acteur peut effectuer avec le système. Ce diagramme nous a permis d'avoir une première approche de notre application.

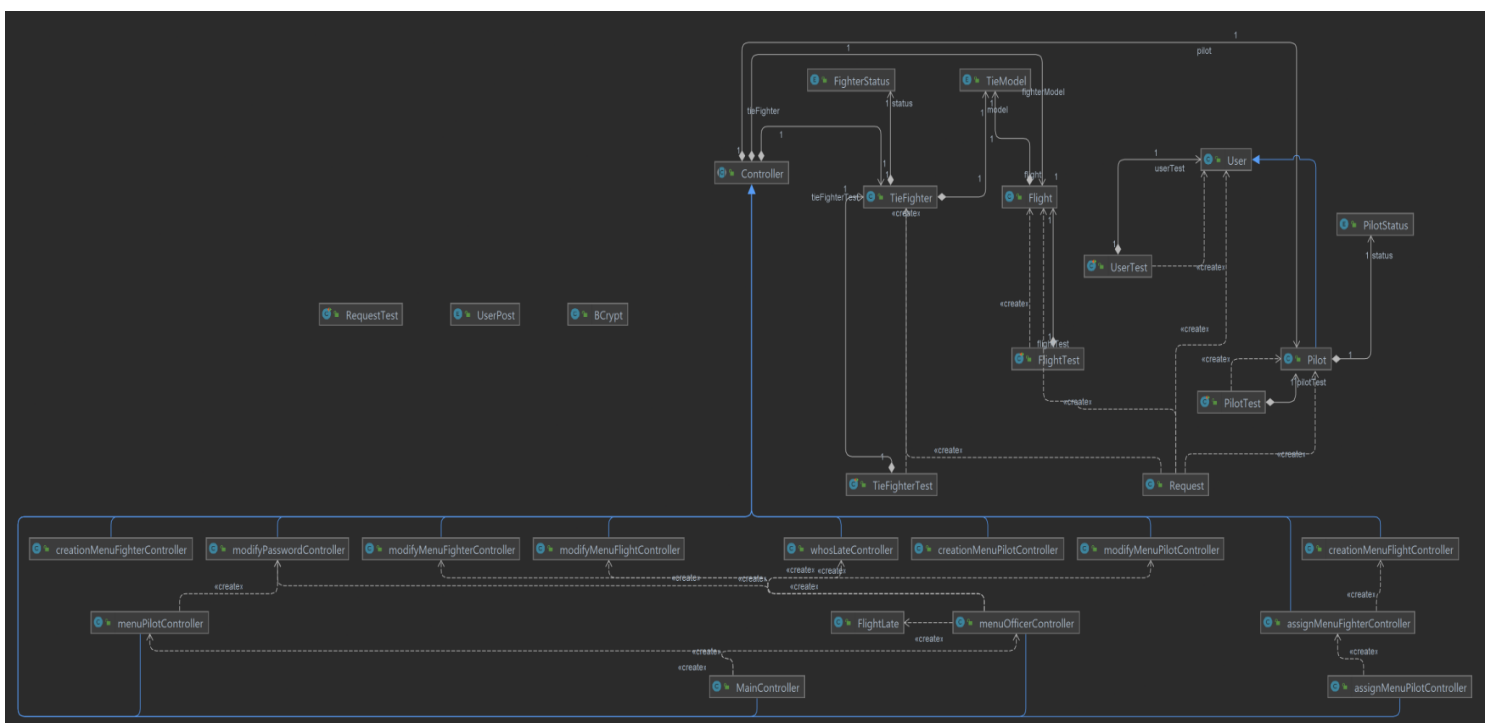
Les deux acteurs principaux sont le pilote et l'officier qui sont une spécialisation du user. Le user peut se connecter sur l'application. Selon le profil, le user peut avoir accès à son historique de vols ou gérer la base de données.

b. Diagramme de classe

Le diagramme de classe sert à décrire la structure de notre application. Il complète le diagramme de cas d'utilisation en modélisant des classes, des attributs, des méthodes et des relations. Nous nous sommes appuyés sur une conception MVC, c'est-à-dire Modèle-Vue-Contrôleur. La partie modèle contient les classes et leurs méthodes. La partie vue est l'interface avec laquelle l'utilisateur interagit. Et le contrôleur permet de faire le lien entre les deux.

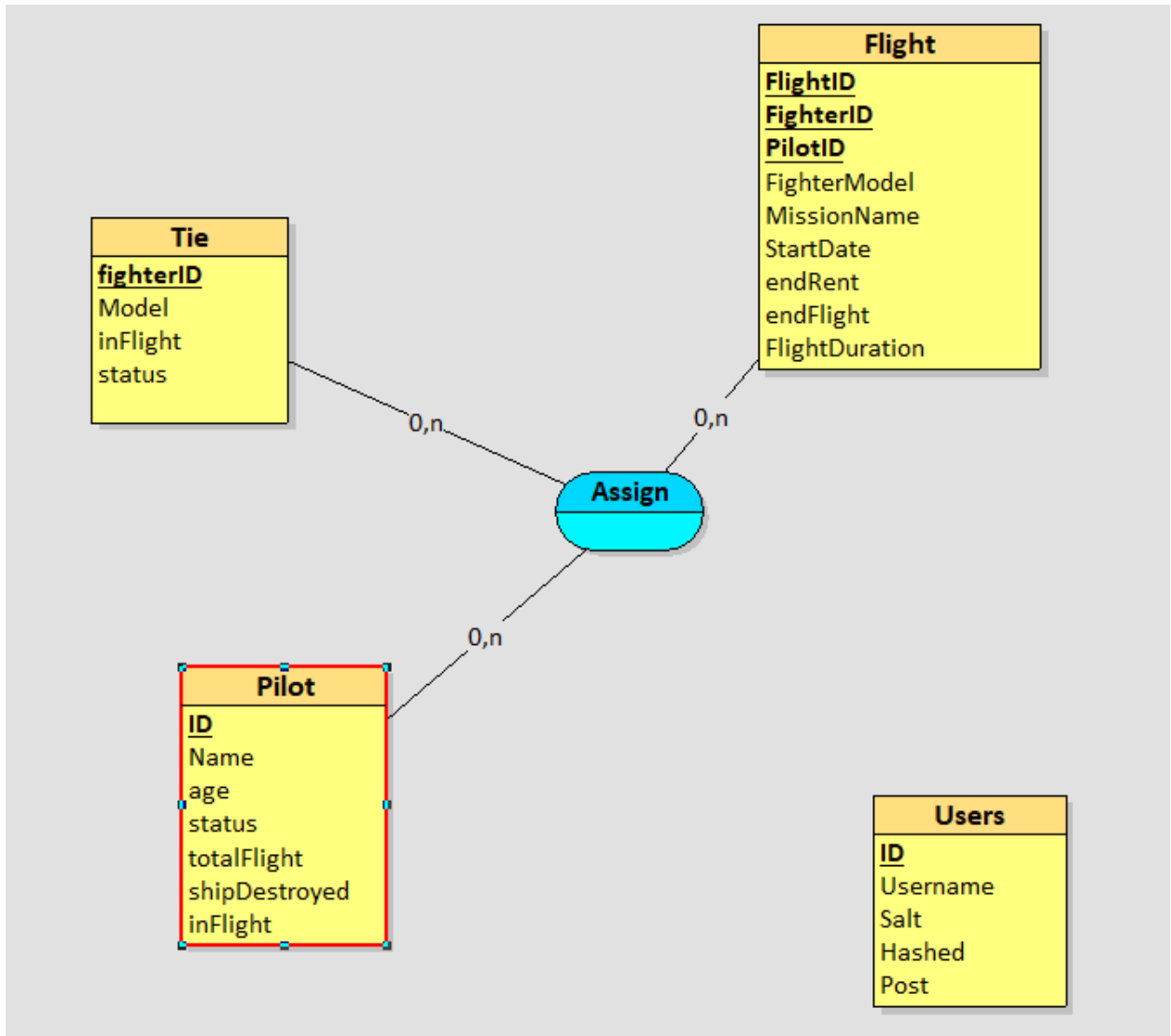
Nous avons réalisé une première version du diagramme de classe dans StarUML. Puis nous avons généré un diagramme de classe avec le plugin "Diagrams" d'IntelliJ. Des versions haut et bas niveau se trouvent dans le dossier UML du projet.

Voici une version du diagramme de classe de l'application (haut niveau) généré grâce au plugin "Diagrams" d'IntelliJ.



IV) MODÈLE CONCEPTUEL DES DONNÉES

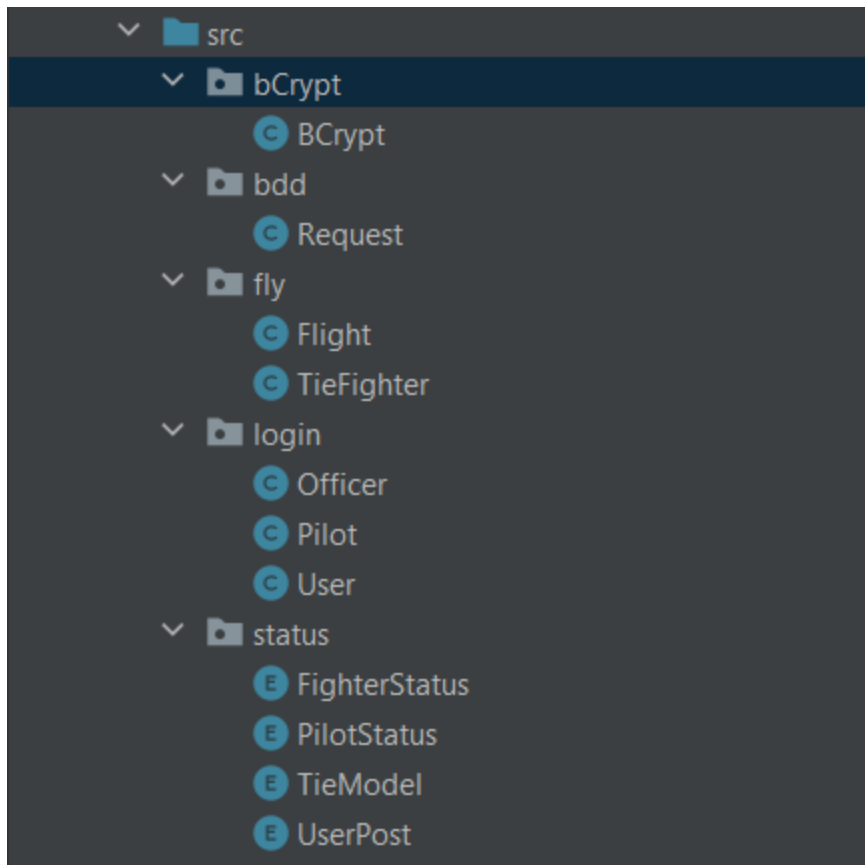
Pour construire la base de données, nous nous sommes basés sur la partie modèle du diagramme de classe.



V) IMPLÉMENTATION DES CLASSES ET DE LA BASE DE DONNÉES

Après avoir modélisé notre application à l'aide des diagrammes UML, nous avons réalisé l'intégration des classes en java.

Voici nos différentes classes et énumérations:



Nous verrons quelques méthodes de la classe request qui contient le cœur de notre code.

Dans cet exemple, nous avons la méthode `displayHistory` qui permet de retourner une liste de tous les vols effectués par un pilote. Nous avons décidé d'utiliser une `ObservableList` car ce type de liste peut ensuite être utilisé dans l'interface de l'application.

```

532
533 public static ObservableList displayHistory(int pilotID) {
534
535     ObservableList<Flight> flights = FXCollections.observableArrayList();
536
537     try {
538         Connection con = getConnection();
539         Statement stmt = con.createStatement();
540         ResultSet rst = stmt.executeQuery( "SELECT flightID FROM Flight WHERE endFlight IS NULL AND pilotID="+pilotID);
541
542         while (rst.next()) {
543             int flightID = rst.getInt( columnIndex: 1);
544             updateFlightDuration(flightID);
545         }
546
547         rst = stmt.executeQuery( "SELECT * FROM Flight WHERE pilotID="+pilotID);
548
549         while (rst.next()) {
550             String model = rst.getString( columnIndex: 3);
551             TieModel tm = TieModel.valueOf(model);
552             Flight flight = new Flight(rst.getInt( columnIndex: 1), rst.getInt( columnIndex: 2), tm, rst.getInt( columnIndex: 4), rst.getString( columnIndex: 5),
553                                     rst.getString( columnIndex: 6), rst.getString( columnIndex: 7), rst.getString( columnIndex: 8), rst.getInt( columnIndex: 9));
554             flights.add(flight);
555         }
556         con.close();
557
558         return flights;
559     } catch (Exception e) {
560         e.printStackTrace();
561         return flights;
562     }
563 }
564

```

Méthode Request.displayHistory

Dans cet exemple, nous avons la méthode assignFlight qui consiste à créer un vol en associant un pilote à un vaisseau. Cette méthode s'assure que le pilote et le vaisseau soient disponibles et en état de voler. Puis, elle met à jour les disponibilité des deux.

```

273 //Assigne un vol en prenant un pilote et un vaisseau
274
275 public static boolean assignFlight(int pilotID, int fighterID, String mission, String sStartDate, String sEndRent) throws Exception{
276     try{
277         LocalDate startDate = LocalDate.parse(sStartDate);
278         LocalDate endRent = LocalDate.parse(sEndRent);
279
280         Connection con = getConnection();
281
282         //on vérifie la disponibilité du vaisseau
283
284         Statement stmt = con.createStatement();
285         ResultSet rst = stmt.executeQuery( "SELECT inFlight,Status FROM TieFighter WHERE fighterID="+fighterID);
286
287         rst.next();
288         if (!rst.getString( columnIndex: "status").equals("Functional") || rst.getInt( columnIndex: "inFlight")!=0){
289             System.out.println("Tie fighter not available");
290             return false;
291         }
292
293         //on vérifie la disponibilité du pilote
294
295         Statement stmt3 = con.createStatement();
296         ResultSet rst3 = stmt3.executeQuery( "SELECT COUNT(id) FROM Pilote WHERE id="+pilotID);
297         rst3.next();
298
299         Statement stmt1 = con.createStatement();
300         ResultSet rst1 = stmt1.executeQuery( "SELECT inFlight,Status FROM Pilote WHERE id="+pilotID);
301
302         if (rst3.getInt( columnIndex: 1)==0){
303             System.out.println("Le user correspondant n'est pas un pilote");
304             return false;
305         }else {
306             rst1.next();
307             if (!rst1.getString( columnIndex: "status").equals("Safe") || rst1.getInt( columnIndex: "inFlight") != 0) {
308                 System.out.println("Pilot not available");
309                 return false;
310             }
311         }
312     }
313 }

```

```

111     }
112
113     // si les informations précédentes sont ok, on insert le nouveau vol
114
115     ResultSet rst2 = stmt.executeQuery( sql: "SELECT model FROM TieFighter WHERE fighterID="+fighterID);
116     rst2.next();
117     String model = rst2.getString( columnIndex: 1);
118
119     PreparedStatement pstmt = con.prepareStatement( sql: "INSERT INTO Flight ('pilotID', 'fighterModel', 'fighterID', 'missionName', 'start', 'endRent') " +
120     "VALUES (" +pilotID+", "+model+", "+fighterID+", "+mission+", "+startDate+", "+endRent+")");
121     pstmt.executeUpdate();
122
123     con.close();
124
125     InFlightUpdate(fighterID, pilotID);
126
127     return true;
128
129 } catch (Exception e){
130     System.out.println(e);
131     return false;
132 }
133
134 finally {
135     System.out.println("Assignment Completed.");
136 }
137 }

```

Méthode Request.assignFlight

Dans le constructeur de la classe User, le mot de passe pris en argument est crypté avant d'être stocké dans les attributs "salt" et "hashed". En effet, grâce au package bcrypt, on génère un "sel" (chaîne de caractère) que l'on associera à l'utilisateur. On génère ensuite un hachage de la combinaison "mot de passe + sel" grâce à la méthode hashpw de BCrypt. Dans la méthode Request.createUser, ces données (sel et hash) seront alors stockées dans la table user de la base de données, associées à l'id de l'utilisateur.

```

21     public User(int id,String username, String password) {
22         this.id=id;
23         this.username = username;
24         this.salt = BCrypt.gensalt();
25         this.hashed = BCrypt.hashpw(password,this.salt);
26     }

```

Constructeur de la classe User.

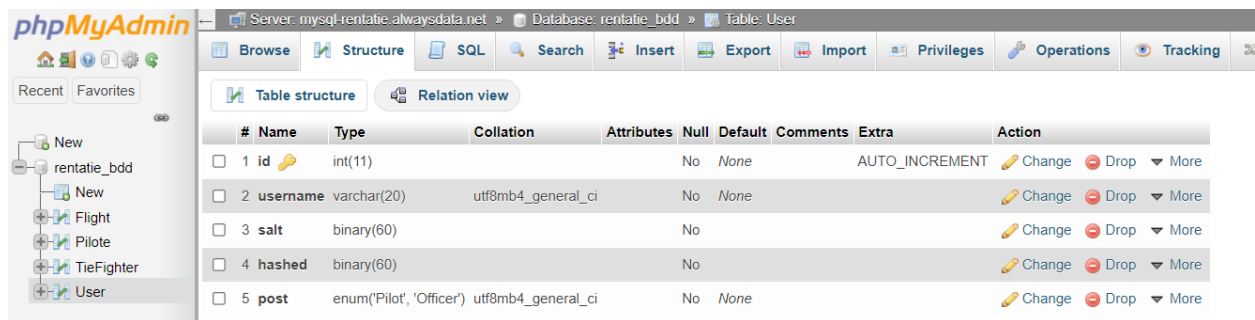
```

246     User user = new User(userID,username, pwd);
247     String salt = user.getSalt();
248     String hashed = user.getHashed();
249
250     PreparedStatement pstmt2 = con.prepareStatement( sql: "UPDATE User SET salt="+salt+", hashed="+hashed+" WHERE id="+userID);
251     pstmt2.executeUpdate();

```

Partie de code de la méthode Request.createUser qui concerne le cryptage du mot de passe.

Pour la base de données, nous avons décidé de la créer sur le site d'hébergement `alwaysdata`.



The screenshot shows the phpMyAdmin interface. The left sidebar displays a tree view of the database structure, including 'rentatie_bdd' and its tables: 'Flight', 'Pilot', 'TieFighter', and 'User'. The main panel shows the 'Table structure' view for the 'User' table. The table has five columns: 'id' (int(11), AUTO_INCREMENT), 'username' (varchar(20)), 'salt' (binary(60)), 'hashed' (binary(60)), and 'post' (enum('Pilot', 'Officer')).

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	username	varchar(20)	utf8mb4_general_ci		No	None			Change Drop More
3	salt	binary(60)			No				Change Drop More
4	hashed	binary(60)			No				Change Drop More
5	post	enum('Pilot', 'Officer')	utf8mb4_general_ci		No	None			Change Drop More

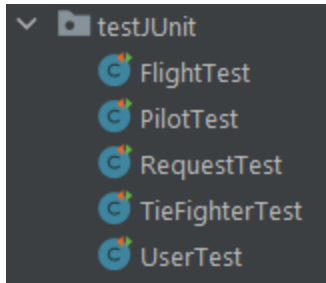
La connexion à cette dernière se fait grâce à la méthode `Request.getConnection` qui utilise les données stockées dans le fichier `conf.properties` du dossier `ressources`.

```
36 @ public static Connection getConnection() throws Exception{
37     Properties props = new Properties();
38     try (FileInputStream fis = new FileInputStream( name: "RentATie/ressources/conf.properties")) {
39         props.load(fis);
40         String driver = props.getProperty("jdbc.driver.class");
41         String url = props.getProperty("jdbc.url");
42         String username = props.getProperty("jdbc.username");
43         String password = props.getProperty("jdbc.password");
44         Class.forName(driver);
45
46         Connection conn = DriverManager.getConnection(url,username,password);
47         System.out.println("Connected");
48         return conn;
49     } catch(Exception e){System.out.println(e);}
50     return null;
51 }
```

Méthode `Request.getConnection`

VI) TESTS UNITAIRES

A l'aide de JUnit 5, nous avons effectué des tests unitaires permettant de vérifier le bon fonctionnement de nos classes.



Nous avons réalisé des classes Test dans le package testJUnit pour les classes des packages login, fly et bdd.

Nous allons expliquer certains tests de la classe RequestTest qui nous paraissent les plus pertinents.

Avant chaque test, la méthode setUp est appelée. Après chaque test, la méthode tearDown est appelée.

La méthode setUp sert à insérer des éléments (un pilote, un vaisseau et un vol) dans la base de données qui serviront dans les tests.

La méthode tearDown supprime de la base de données les éléments insérés par setUp et réinitialise les paramètres "Auto_increment" des id de la base de données à la bonne valeur.

On utilise ces méthodes pour que chaque test soit indépendant.

```
29      @BeforeEach
30      void setUp() throws Exception {
```

```
70      @AfterEach
71      void tearDown() throws Exception {
```

Pour tester la méthode assignFlight, on a réalisé 5 méthodes de test afin de vérifier son bon fonctionnement dans plusieurs cas possibles :

```
Test assignFlight.  
Throws: Exception – If there was a problem in the execution of the methods used.  
test.case Pilot already in flight.  
test.result Flight not created.  
  
MazirG  
307 @Test  
308 void assignFlightTestNotOkPilotInFlight() throws Exception {  
309     Connection con = getConnection();  
310     assertNotNull(con);  
311  
312     PreparedStatement pstmt = con.prepareStatement( sql: "UPDATE Pilote SET inFlight=1 WHERE id=999");  
313     pstmt.executeUpdate();  
314  
315     boolean verif = assignFlight( pilotID: 999, fighterID: 9999, mission: "missionTest2", sStartDate: "2000-01-01", sEndRent: "2023-01-01" );  
316  
317     con.close();  
318     assertFalse(verif);  
319 }
```

Premier cas : le pilote est déjà en vol.

```
Test assignFlight.  
Throws: Exception – If there was a problem in the execution of the methods used.  
test.case Pilot not safe.  
test.result Flight not created.  
  
MazirG  
327 @Test  
328 void assignFlightTestNotOkPilotNotSafe() throws Exception {  
329     Connection con = getConnection();  
330     assertNotNull(con);  
331  
332     PreparedStatement pstmt = con.prepareStatement( sql: "UPDATE Pilote SET status='Lost' WHERE id=999");  
333     pstmt.executeUpdate();  
334  
335     boolean verif = assignFlight( pilotID: 999, fighterID: 9999, mission: "missionTest2", sStartDate: "2000-01-01", sEndRent: "2023-01-01" );  
336  
337     con.close();  
338     assertFalse(verif);  
339 }
```

Deuxième cas : le pilote est mort ou blessé.

```

Test assignFlight.
Throws: Exception – If there was a problem in the execution of the methods used.
test.case TieFighter already in flight.
test.result Flight not created.

MazirG
@Test
void assignFlightTestNotOKFighterInFlight() throws Exception {

    Connection con = getConnection();
    assertNotNull(con);

    PreparedStatement pstmt = con.prepareStatement( sql: "UPDATE TieFighter SET inFlight=1 WHERE fighterID=9999 ");
    pstmt.executeUpdate();

    boolean verif = assignFlight( pilotID: 999, fighterID: 9999, mission: "missionTest2", sStartDate: "2000-01-01", sEndRent: "2023-01-01" );

    con.close();
    assertFalse(verif);
}

```

Troisième cas : le vaisseau est déjà en vol..

```

Test assignFlight.
Throws: Exception – If there was a problem in the execution of the methods used.
test.case TieFighter not functional.
test.result Flight not created.

MazirG
@Test
void assignFlightTestNotOKFighterDestroyed() throws Exception {

    Connection con = getConnection();
    assertNotNull(con);

    PreparedStatement pstmt = con.prepareStatement( sql: "UPDATE TieFighter SET status='Destroyed' WHERE fighterID=9999 ");
    pstmt.executeUpdate();

    boolean verif = assignFlight( pilotID: 999, fighterID: 9999, mission: "missionTest2", sStartDate: "2000-01-01", sEndRent: "2023-01-01" );

    con.close();
    assertFalse(verif);
}

```

Quatrième cas : le vaisseau n'est pas en bon état.

```
Test assignFlight.  
Throws: Exception - If there was a problem in the execution of the methods used.  
test.case Pilot and TieFighter OK.  
test.result The method assignFlight was executed correctly and a flight was created.  
  
MazirG  
@Test  
void assignFlightTestOK() throws Exception {  
  
    boolean _verif = assignFlight( pilotID: 999, fighterID: 9999, mission: "missionTest2", sStartDate: "2000-01-01", sEndRent: "2023-01-01" );  
  
    assertTrue(verif);  
  
    Connection con = getConnection();  
    assertNotNull(con);  
  
    Statement stmt = con.createStatement();  
    ResultSet rst = stmt.executeQuery( sql: "SELECT COUNT(*) FROM Flight WHERE missionName='missionTest2' AND fighterID=9999" );  
    rst.next();  
    int inFlight = rst.getInt( columnIndex: 1);  
  
    assertEquals( expected: 1, inFlight);  
  
    PreparedStatement pstmt = con.prepareStatement( sql: "DELETE From Flight WHERE missionName='missionTest2' AND fighterID=9999" );  
    pstmt.executeUpdate();  
  
    con.close();  
}
```

Cinquième cas : tout est ok, le vol est créé.

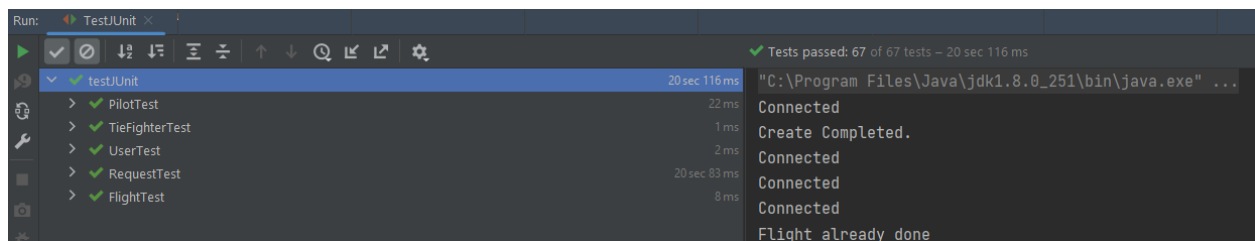
Dans chaque cas, on lance la méthode assignFlight en ayant réalisé auparavant les modifications voulues dans la BDD (pilote en vol, vaisseau endommagé ...). On vérifie ensuite que, pour les 4 premiers cas, la méthode renvoie “false” ce qui signifie que le vol n’a pas été créé. Pour le dernier cas, on vérifie que la méthode renvoie bien “true” et on vérifie ensuite que le vol à bien été créé dans la BDD.

Pour tester la méthode displayFlightTable on utilise une méthode qui va s’assurer que ce que renvoie displayFlightTable n’est pas une liste vide.

```
Test displayFlightTable.  
test.result The list returned is not empty.  
  
MazirG  
@Test  
void displayFlightTableTest() {  
    ObservableList flights = displayFlightTable();  
    assertFalse(flights.isEmpty());  
}
```

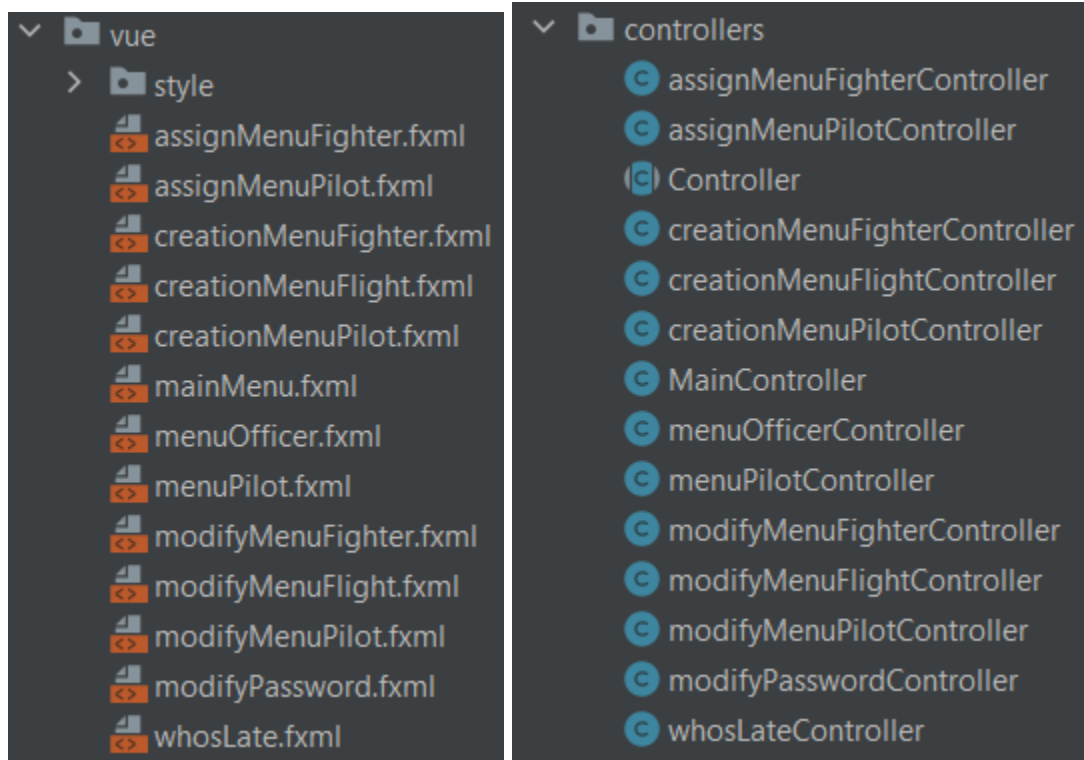
méthode displayFlightTableTest

Une fois tous les tests écrits, on lance un run JUnit qui va tester tous les tests du package “testJUnit” un par un.



résultat des tests JUnit

VII) INTERFACE GRAPHIQUE



Pour le développement de l'interface graphique, nous avons fait en parallèle la partie Controller (package "controllers") et la partie Vue (package "vue"). Chaque fichier FXML de la partie Vue possède directement son contrôleur dans le package controllers

La classe Controller

```
public abstract class Controller {  
  
    private Pilot pilot;  
    private TieFighter tieFighter;  
    private Flight flight;  
  
    public int UserId;  
    public ObservableList listLate;  
}
```

Dans le package controllers, l'ensemble des contrôleurs héritent de la classe Controller. Elle possède en champs les différentes informations qui peuvent être transmises d'une page à une autre. Ces informations peuvent être un Pilot, un TieFighter, un Flight. Par exemple, la fenêtre menuOfficer contenant l'ensemble des Pilots dans la table va récupérer un Pilot sélectionné par l'officier et le transmettre à une fenêtre modifyPilot à l'aide des méthodes de la classe Controller. Les deux derniers champs UserId et listLate concernent respectivement la modification de mot de passe et l'affichage des Flights ayant un retard pour leur location.

Les méthodes de Controller

```
public void open(String pageName){
    Parent fxmL;
    Stage home = new Stage();
    home.setResizable(false);
    try{
        fxmL = FXMLLoader.load(getClass().getResource(pageName));
        Scene scene = new Scene(fxmL);

        home.setScene(scene);
        home.show();
    }
    catch (Exception e){
        e.printStackTrace();
    }
}
```

La classe Controller possède un ensemble de méthodes pour la gestion des ouvertures et fermetures de pages. Elles prennent en général un String comme open() ci-dessus, qui va contenir l'adresse du fichier FXML à ouvrir dans l'application. Il y a également d'autres méthodes pour quitter une fenêtre à partir du bouton ayant enclenché la méthode (quit()), et aussi d'autres méthodes comme open_share() qui permettent de transmettre un Objet a une autre fenêtre à partir du contrôleur de la nouvelle fenêtre.

Les actions de certains éléments (Relations Contrôleur - Vue)

```
@FXML
void cbFighter_checked() {

    c1.setText("id");
    c2.setText("model");
    c3.setText("inFlight");
    c4.setText("status");
    c1.setVisible(true);
    c2.setVisible(true);
```

```
@FXML
private CheckBox cbFighter;
```

Certains éléments de la vue peuvent être amenés à effectuer des actions lancées par le contrôleur. Les éléments du fichier FXML peuvent être associés à des actions comme par exemple cette CheckBox qui est associée à l'action cbFighter_checked() du contrôleur, qui est lancée à chaque fois que la CheckBox est modifiée. A l'aide des différents @FXML il est donc possible de faire les liens entre la partie Vue et Contrôleur où les éléments correspondent directement.

La gestion des requêtes dans les controllers (Relations Contrôleur - Modèle)

Les requêtes de lectures de données

```
ObservableList<TieFighter> list = FXCollections.observableArrayList();
list= Request.displayTieFighterAvailable();

cId.setCellValueFactory(new PropertyValueFactory<TieFighter,Integer>("fighterId"));
cModel.setCellValueFactory(new PropertyValueFactory<TieFighter, TieModel>("model"));
cInFlight.setCellValueFactory(new PropertyValueFactory<TieFighter,Boolean>("inFlight"));
cStatus.setCellValueFactory(new PropertyValueFactory<TieFighter, FighterStatus>("status"));

tableItems.setItems(list);
```

La majeure partie des requêtes sql de lecture de données sont utilisées pour l'affichage de données dans les tableaux TableView. Pour chaque affichage spécifique, une requête sql a été mise en place (sauf pour les Flight en retard), et cette requête renvoie une liste qui peut être mise en tant que affichage de la table. Dans cet exemple ci dessus d'assignFighter, la requête displayTieFighterAvailable() renvoie dans une liste tous les TieFighter disponibles (ceux ayant status= functional et inFlight=false).

Les requêtes d'écritures / de modifications

```
if(status.equals("Wrecked")==false && status.equals("Functional")==false && status.equals("Destroyed")==false && st
    Alert errorAlert = new Alert(Alert.AlertType.ERROR);
    errorAlert.setHeaderText("Error");
    errorAlert.setContentText("You need to chose a status to modify !");
    errorAlert.showAndWait();
}
else{
    boolean succeed = true;
    try {
        Request.modifyFighterStatus(this.getTieFighter().getFighterId(), FighterStatus.valueOf(this.getStatus()));
    }catch (Exception e){
        succeed=false;
        Alert errorAlert = new Alert(Alert.AlertType.ERROR);
        errorAlert.setHeaderText("Error");
        errorAlert.setContentText("Request has failed !");
        errorAlert.showAndWait();
    }
    if(succeed){
        super.open_quit(btnValidate, pageName: "/vue/menuOfficer.fxml");
        Alert errorAlert = new Alert(Alert.AlertType.INFORMATION);
        errorAlert.setContentText("The Fighter has been modified !");
        errorAlert.show();
    }
}
```

Les requêtes d'écritures prennent en compte un ensemble d'erreurs à prendre en compte. Elles dépendent d'une part des inputs réalisées par l'utilisateur qui peut entrer des données faussées (par exemple des lettres pour un int), et d'autre part des résultats des requêtes elle-même. Pour cela, il faut donc émettre des conditions pour que les champs de données soient correctement remplis. Dans l'exemple ci-dessus, on attend à ce que l'utilisateur choisisse un statut pour modifier un TieFighter, mais il ne peut pas lancer la requête si il ne sélectionne pas de TieFighter.

```

        sqlResult=Request.deleteUser(pilotSelected.getPilotID());
    }catch (Exception e){
        succeed=false;
        Alert errorAlert = new Alert(Alert.AlertType.ERROR);
        errorAlert.setHeaderText("Error");
        errorAlert.setContentText("Request has failed !");
        errorAlert.showAndWait();
    }
    if(succeed && sqlResult==1){
        Alert errorAlert = new Alert(Alert.AlertType.INFORMATION);
        errorAlert.setContentText("The Pilot has been removed !");
        errorAlert.show();
    }

    else if(succeed && sqlResult==0){
        Alert errorAlert = new Alert(Alert.AlertType.ERROR);
        errorAlert.setHeaderText("Error");
        errorAlert.setContentText("Pilot already removed !");
        errorAlert.showAndWait();
    }
}

```

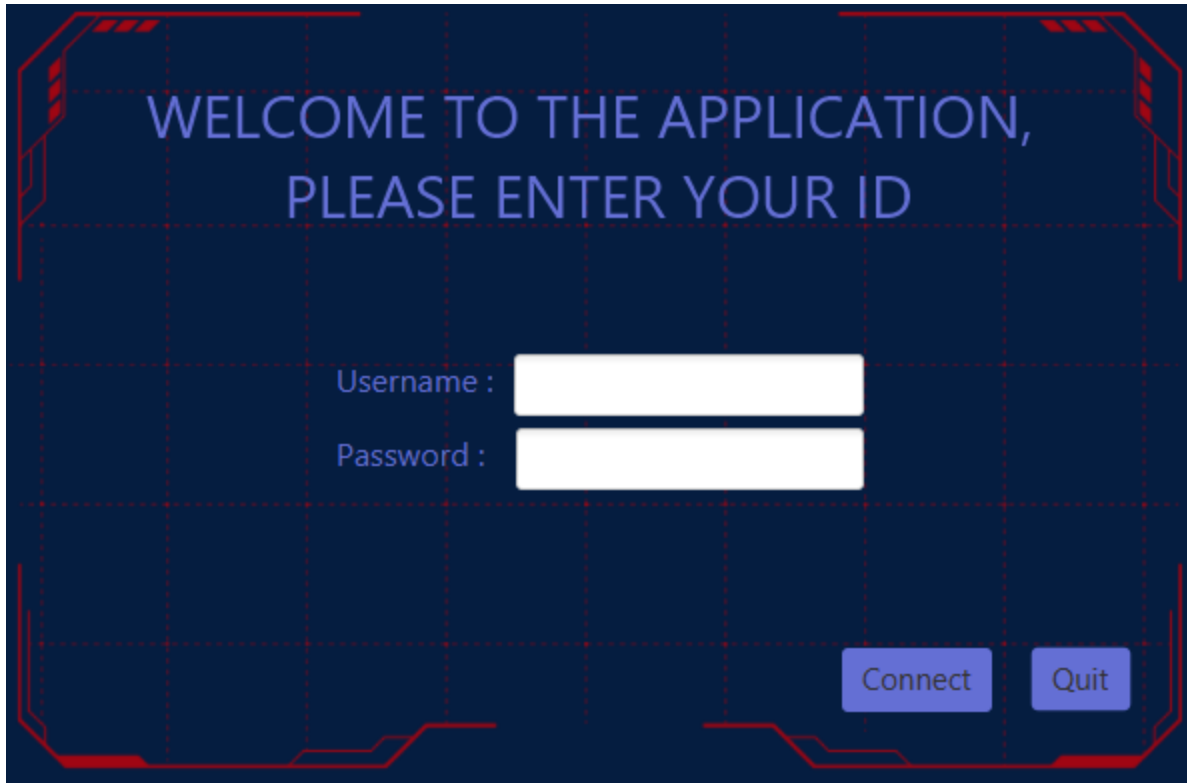
Dans ce second exemple, cette fois ci de deleteUser situé dans menuOfficer, on regarde le résultat renvoyé par la requête, et on affiche un message personnalisé en fonction du résultat reçu dans sqlResult (pouvant être indicateur du succès ou du type d'erreur trouvé dans la requête)

Les contrôleurs qui implémentent Initializable / cacher des éléments

```
@Override
public void initialize(URL location, ResourceBundle resources) {
    c1.setVisible(false);
    c2.setVisible(false);
    c3.setVisible(false);
    c4.setVisible(false);
    c5.setVisible(false);
}
```

La plupart des classes ayant des fonctions d'écriture dans la BDD sont souvent implémentées avec une méthode `initialize` surchargée. Cette méthode permet d'effectuer des actions dès que les fenêtres sont créées, et celle-ci va être utilisée pour cacher des composantes de l'interface. L'objectif est de cacher certains éléments ou certaines fonctionnalités à l'utilisateur pour qu'il ne puisse pas effectuer des actions amenant à des erreurs. Par exemple, dans menu Officer, un Pilot ou un TieFighter peut être ajouté directement alors qu'un Flight nécessite un assignment. Lorsque Flight sera sélectionné, l'affichage du bouton "Add an element" sera retiré

VIII) APPLICATION FINALE



The image shows a login screen with a dark blue background and a red dashed grid. The text "WELCOME TO THE APPLICATION, PLEASE ENTER YOUR ID" is displayed in a light blue, sans-serif font at the top. Below this, there are two input fields: "Username :" and "Password :". The "Username :" field is a white rectangle, and the "Password :" field is a white rectangle with a small black dot in the center. At the bottom right, there are two buttons: "Connect" and "Quit", both in a light blue color with a dark blue border.

page MainMenu

Cette page permet à l'utilisateur de se connecter ses identifiants et mot de passe. Selon son type, pilote ou officier, l'utilisateur aura accès à la page de menu officier ou du menu pilote. Il est aussi possible de quitter l'application grâce à au bouton "Quit"

Search by name...

Session of pilot

Refresh

id	mission	pilot-id	fighter-id	fighter-model	date-begin	date-end	date-back	duration(day)
18	BO2	22	13	Fighter	2022-05-20	2022-05-23	2022-05-22	2

Modify password

Disconnect

page MenuPilot

La page de menu du pilote est constituée d'une table contenant tous les vols effectués par celui-ci. Le pilote peut chercher un vol en entrant le nom de la mission dans la barre de recherche. Il peut aussi rafraîchir sa page, modifier son mot de passe ou se déconnecter.

Session of officer

Search a property ...

Modify Password

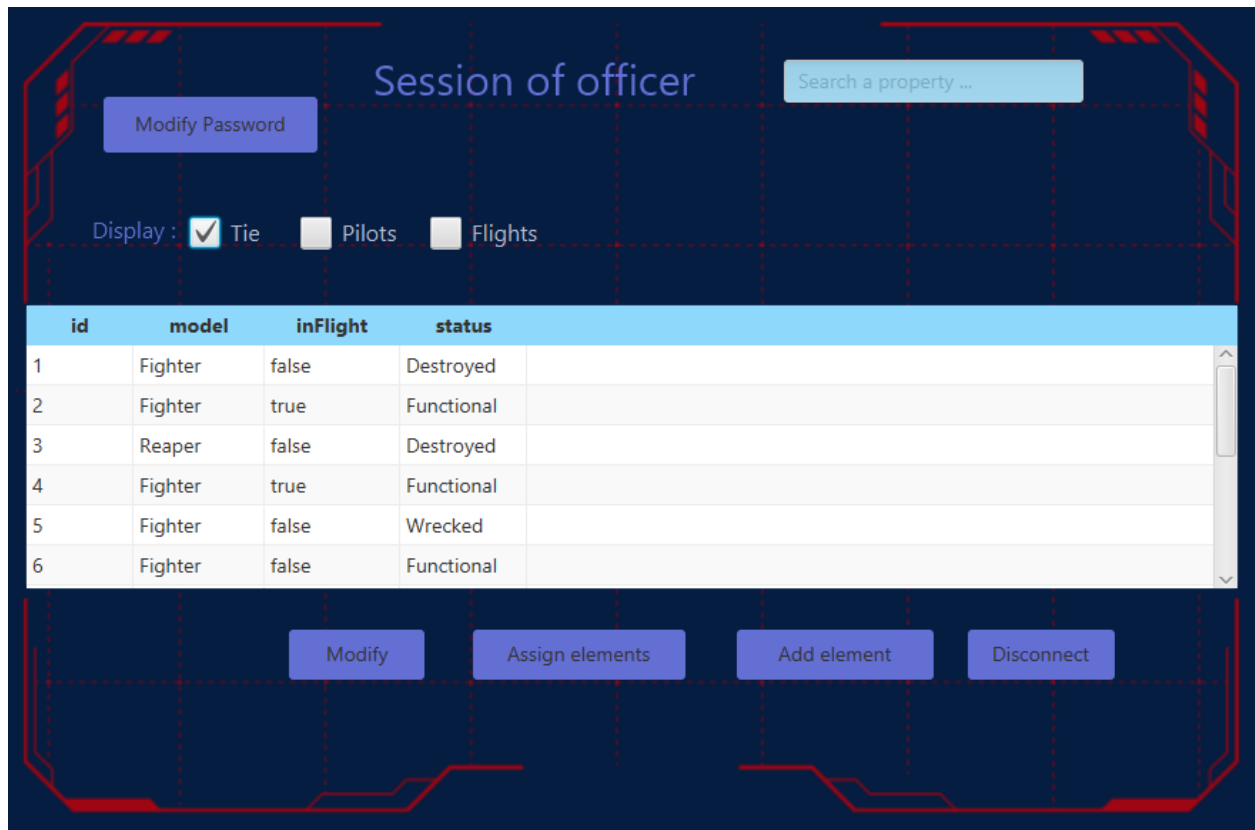
Display : ☐ Tie ☒ Pilots ☐ Flights

id	nom	age	inFlight	status	total-flights	ship-destroy...	
1	Jean	51	false	Lost	1	1	
2	Michelle	30	false	Dead	1	0	
5	Pseudo2	65	false	Safe	1	27	
7	Leonarda	26	false	Lost	2	17	
9	Naps	45	true	Safe	0	10	
10	Cyril Hanou...	46	true	Safe	0	0	

Delete User Modify Assign elements Add element Disconnect

page menuOfficer

L'officier peut choisir d'afficher les Tie Fighters, les pilotes ou les vols. Pour les Tie Fighter et les pilotes, l'officier peut les modifier, en ajouter ou les supprimer. La suppression d'un pilote revient à supprimer son identifiant et son mot de passe mais ses informations (âge, nom, nombre de vols..) restent dans la base de données.



page menuOfficer affichant les vaisseaux.

The screenshot shows a web interface titled "Session of officer". At the top left is a "Modify Password" button. To its right is a search bar labeled "Search a property ...". Below these are three checkboxes for "Display": "Tie" (unchecked), "Pilots" (unchecked), and "Flights" (checked). To the right of these checkboxes is a red button labeled "See flights that are late".

id	mission	pilot-id	tie-id	tie model	begin-date	end-date	back-date	duration(day)
3	Mission1	2	1	Fighter	2012-11-12	2021-11-23	2022-05-20	0
5	Mission2	1	3	Fighter	2021-12-30	2022-12-20	2022-05-20	0
6	OKAYOKAY	9	2	Reaper	2022-05-19	2022-05-25	2022-05-21	0
7	BABA	12	5	Reaper	2022-05-03	2022-05-05	2022-05-21	0
8	missiontahl...	7	4	Reaper	2018-05-05	2018-05-05	2022-05-21	0
9	ASSIGNTEST	5	6	Fighter	2022-05-05	2022-05-18	2022-05-22	12

At the bottom of the interface are three buttons: "Modify", "Assign elements", and "Disconnect".

page menuOfficer affichant les vols

Lorsque les vols sont affichés et si un vol a du retard, il est possible de les voir en cliquant sur le bouton “see flights that are late”.

Il est possible de créer un vol en cliquant sur le bouton “assign elements “qui permettra d’associer un pilote a un vaisseau.

L’officier peut aussi modifier son mot de passe ou se déconnecter.

Rents that are taking to much time

Refresh
Search flight...

id	begin-rent	end-rent	late-duration
17	2022-05-19	2022-05-20	2

Cancel

page whosLate

Cette page permet d’afficher les vols ayant du retard, il est possible de chercher un vol en entrant le nom de la mission dans la barre de recherche mais aussi de rafraîchir la table contenant les vols.

Choose a pilot to assign

Search a pilot by name

id	name	age	inFlight	status	total-flight	ships-destroyed
5	Pseudo2	65	false	Safe	1	27
18	yuyu	3	false	Safe	0	0
19	vivi	5	false	Safe	2	210
21	Nabil	26	false	Safe	0	0

CancelValidate

page assignPilot

Cette page permet de choisir le pilote à associer pour créer un vol. L'officier doit sélectionner le pilote voulu et valider son choix. Il peut aussi chercher un pilote dans la barre de recherche avec son prénom ou annuler l'association.

Choose a Tie to assign

Search a Tie ...

id	model	in Flight	status
6	Fighter	false	Functional
12	Fighter	false	Functional
13	Fighter	false	Functional

< >

CancelValidate


page assignFighter


Cette page permet de choisir le vaisseau à associer pour créer un vol. L'officier doit sélectionner le vaisseau voulu et valider son choix. Il peut aussi chercher un vaisseau dans la barre de recherche avec son id ou annuler l'association.

Enter Flight name and dates of rent

Flight ID : {Default}

Flight name :

Rent beginning date : 

Rent ending date : 

Fighter back date : TO BE DETERMINED {Default}

page createFlight

Après avoir choisi le pilote et le vaisseau, pour créer un vol, l'officier doit indiquer un nom pour la mission, la date de début de location ainsi que la date de fin. La date de retour du vaisseau sera ensuite ajoutée lorsque le pilote revient de sa mission.

Enter information for a new Pilot

Pilot username :	<input type="text" value="Wartek"/>
Pilot password :	<input type="password" value="•••••"/>
Pilot name :	<input type="text" value="Anil"/>
Pilot age :	<input type="text" value="32"/>
Pilot status	Safe {Default}
Pilot in flight :	No {Default}
Pilot status :	Safe {Default}
Number of flights :	0 {Default}
Ships destroyed	0 {Default}

page createPilot

Pour créer un pilote, l'officier doit lui créer un identifiant et un mot de passe et entrer ses informations personnelles (son prénom et son âge). Par défaut, le pilote est opérationnel, disponible et il n'a pas encore effectué de vol ou détruit un vaisseau.

Choose the model of the new Tie

Tie ID :	{Default}
Tie model :	<input checked="" type="radio"/> Fighter <input type="radio"/> Reaper
Tie status :	Functional {default}
Tie in Flight : :	No {Default}

Validate Cancel

page createTie

Pour créer un Tie, l'officier doit seulement choisir son modèle. Par défaut, le tie est opérationnel et disponible.

Modify Flight

Flight ID :	18
Flight name:	BO2
Rent start date :	2022-05-20
Rent end date :	2022-05-23
Rent back date :	2022-05-22
Pilot ID :	22
Tie ID :	13
Tie new status :	<input checked="" type="radio"/> Functional <input type="radio"/> Damaged <input type="radio"/> Wrecked <input type="radio"/> Destroy...
Pilot new status :	<input checked="" type="radio"/> Safe <input type="radio"/> Wounded <input type="radio"/> Lost <input type="radio"/> Dead
Enemy ships destroyed by the pilot :	<input type="text" value="2420"/>

page modifyFlight

Lors du rendu du vaisseau, l'officier peut modifier le vol pour indiquer le nouveau statut du Tie ainsi que le statut du pilote. La date de rendu est automatiquement changée avec la date du jour de la modification.

Modify Pilot

Pilot ID : 1

Pilot name: Jean

Number of flights : 1

Pilot age : 51

Pilot status : ☒ Safe ☐ Wounded ☐ Lost ☐ Dead

Pilot in flight : No

Enemy ships destroyed : 1

Refresh Validate Cancel

page modifyPilot

L'officier peut modifier le statut et l'âge du pilote ou annuler la modification.

Modify Tie

Tie ID : 5

Tie model : Fighter

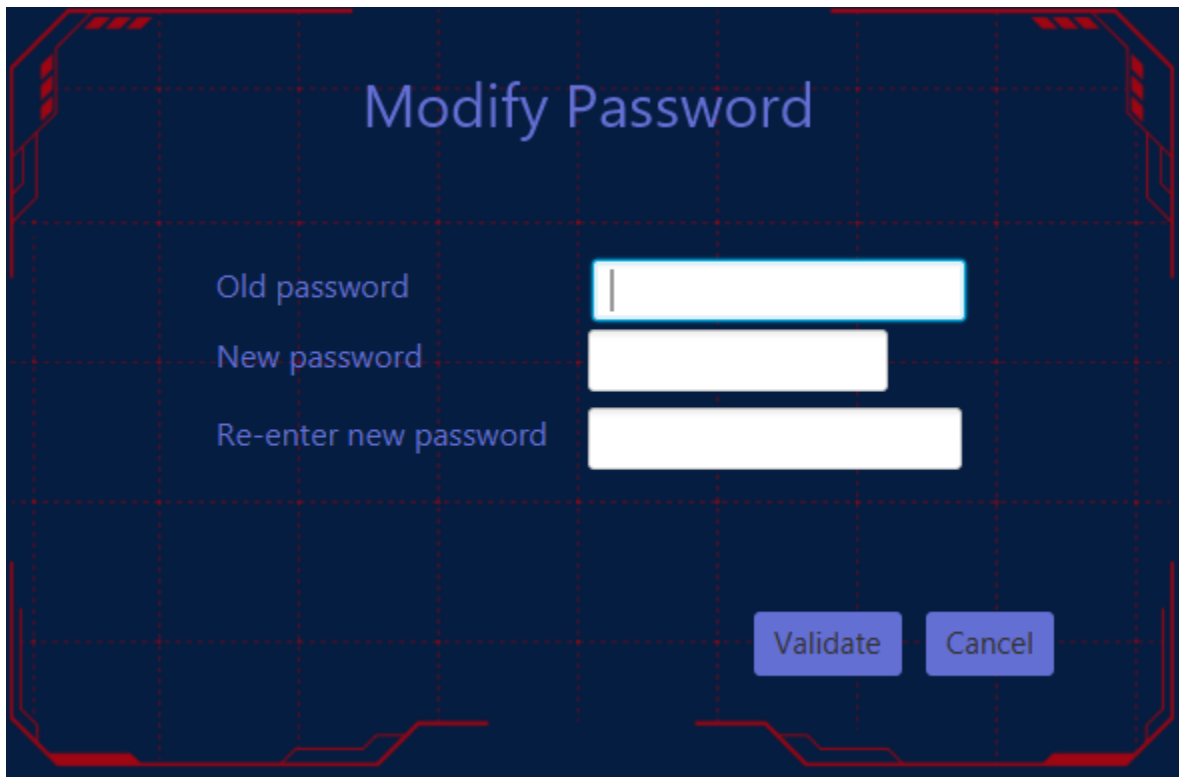
Tie in flight : No

Tie status : ☐ Functional ☐ Damaged ☐ Wreck ☐ Destroyed

Refresh Validate Cancel

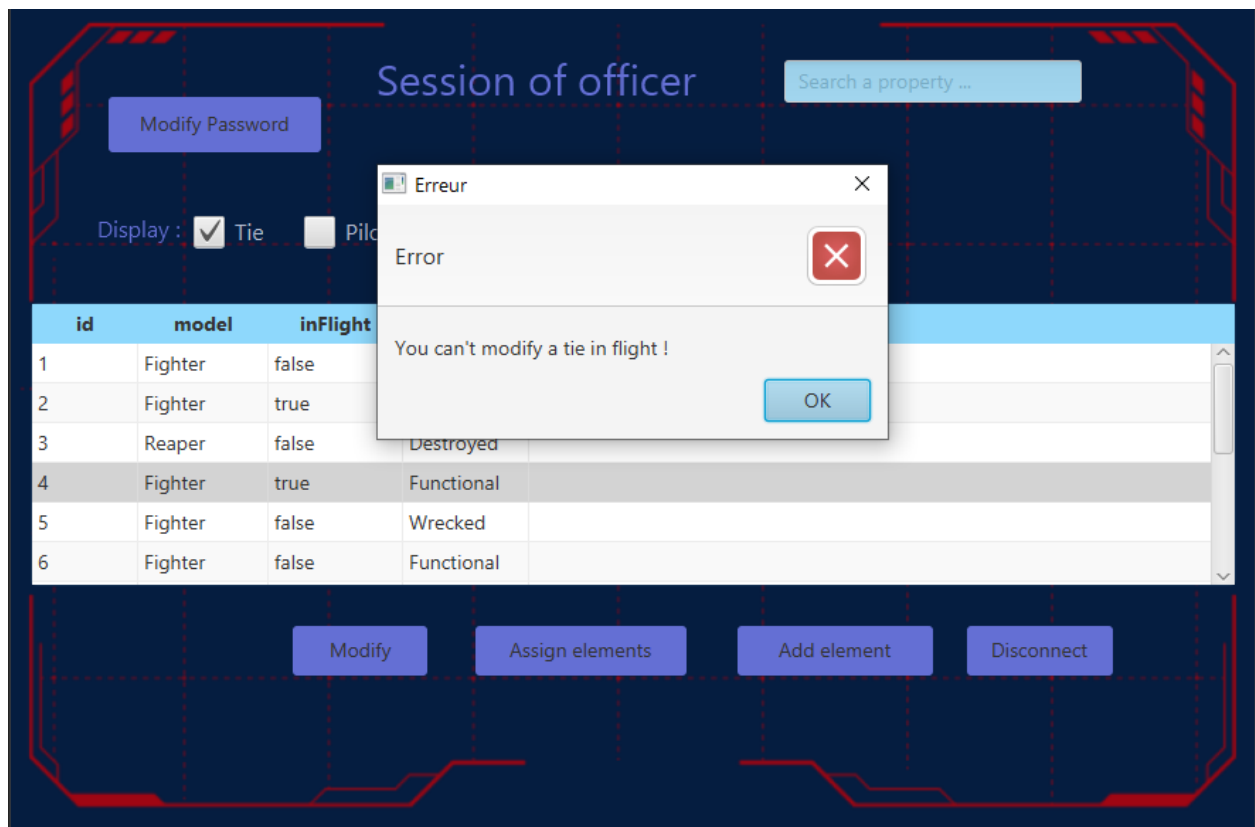
page modifyTie

L'officier peut seulement modifier le statut du tie ou annuler la modification.

The image shows a 'Modify Password' form with a dark blue background and a red dashed grid. The title 'Modify Password' is at the top center in a light blue font. Below it are three input fields: 'Old password', 'New password', and 'Re-enter new password'. The 'Old password' field has a light blue border and a cursor. The 'New password' and 'Re-enter new password' fields have white borders. At the bottom right are two light blue buttons labeled 'Validate' and 'Cancel'.

page modifyPassword

L'utilisateur peut modifier son mot de passe en entrant son ancien mot de passe puis son nouveau deux fois et valider.



exemple d'une erreur : ici on veut modifier un vaisseau qui est en vol

Il y a des erreurs personnalisées pour chaque erreur possible liée aux entrées de l'utilisateur ou à la lecture dans la base de données.

IV) CONCLUSION

Pour conclure, nous pensons avoir réussi à répondre à la problématique du sujet. L'ensemble des fonctionnalités de bases demandées ont réussi à être implémentées, et nous avons accordé un temps considérable aux différents tests du modèle et conditions d'erreur dans l'interface du projet. Néanmoins il reste certains points noirs comme par exemple au niveau de l'interface avec des problèmes mineurs de version de JavaFX et autres que nous n'avons pas réussi à régler qui fait que nous avons un warning à chaque affichage d'une nouvelle fenêtre. Nous trouvons aussi que l'application aurait gagné à implémenter des fonctionnalités supplémentaires permettant d'améliorer son ergonomie, avec des fonctionnalités comme le fait de voir les Flight en retard qui auraient pu être intégrées avec plus d'efficacité. Toujours dans l'interface, il y a certaines parties du code qui auraient pu être mieux optimisées, notamment sur certaines répétitions de code. Malgré ces légers regrets vis à vis de ces certains points, nous sommes totalement satisfait du travail que nous avons effectué pour ce projet, en considérant notamment le fait que ce projet soit un premier projet pour la majeure partie des membres de l'équipe (4/5).