

**Préambule :** Avez-vous des questions sur les enseignements dispensés précédemment ?

**Objectif :**


- Mettre en place des tests end-to-end
- Améliorer la couverture de test

Sur ce dernier TP, nous allons mettre en place des tests end-to-end. Ces tests automatiques testent l'intégralité de l'application, en l'exerçant depuis son interface la plus extérieure, à savoir ici l'IHM, comme un utilisateur le ferait. Cela va nous permettre de vérifier que scénario nominal et cas d'erreurs sont bien implémentés dans l'application, et que le bon résultat s'affiche dans l'IHM. Au travers de ces tests nous vérifions aussi que l'interaction avec la web API fonctionne correctement.

**Travail à faire :**

1. L'application se base sur JavaFX. Nous allons utiliser la librairie TestFX qui facilite le test de ces applications. Dans le fichier *build.gradle*, ajouter les dépendances TestFX dans la section *dependencies* :  

```
testImplementation 'org.testfx:testfx-core:4.0.16-alpha'
testImplementation 'org.testfx:testfx-junit5:4.0.16-alpha'
testImplementation 'org.testfx:openjfx-monocle:jdk-11+26'
```
2. Ajouter une dépendance sur *Awaitility* dans la section *dependencies* :  

```
testImplementation 'org.awaitility:awaitility:4.2.0'
```
3. Rafraichir le projet dans la vue Gradle dans  IntelliJ
4. Créer un fichier de test *PokeBagarreAppTest.java* (*Ctrl+Maj+T* depuis la classe *PokeBagarreApp*)
5. Remplacer la classe avec le code fourni ci-dessous
6. Ecrire autant de tests que nécessaire pour couvrir les différents scénarios

Code de base de la classe *PokeBagarreAppTest.java* :

```

package com.montaury.pokebagarre.ui;

import java.util.concurrent.TimeUnit;
import javafx.stage.Stage;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.testfx.api.FxRobot;
import org.testfx.framework.junit5.ApplicationExtension;
import org.testfx.framework.junit5.Start;

import static org.assertj.core.api.Assertions.assertThat;
import static org.awaitility.Awaitility.await;

@ExtendWith(ApplicationExtension.class)
class PokeBagarreAppTest {

    private static final String IDENTIFIANT_CHAMP_DE_SAISIE_POKEMON_1 = "#nomPokemon1";
    private static final String IDENTIFIANT_CHAMP_DE_SAISIE_POKEMON_2 = "#nomPokemon2";
    private static final String IDENTIFIANT_BOUTON_BAGARRE = ".button";

    @Start
    private void start(Stage stage) {
        new PokeBagarreApp().start(stage);
    }

    @Test
    void nom_du_test(FxRobot robot) {
        //robot.clickOn(IDENTIFIANT);
        //robot.write("Text");

        //await().atMost(5, TimeUnit.SECONDS).untilAsserted(() ->
        //    assertThat(...).isEqualTo(...)
        // );
    }

    private static String getResultatBagarre(FxRobot robot) {
        return robot.lookup("#resultatBagarre").queryText().getText();
    }
}

```

```
private static String getMessageErreur(FxRobot robot) {  
    return robot.lookup("#resultatErreur").queryLabeled().getText();  
}  
  
}
```

Note : Pour que les tests passent puissent s'exécuter au sein des GitHub Actions, qui tournent sur des machines sans interface graphique, il faudra ajouter dans la section *test* du *build.gradle* :

```
systemProperties['java.awt.headless'] = true  
systemProperties['testfx.robot'] = 'glass'  
systemProperties['testfx.headless'] = 'true'
```

Cela a pour effet de jouer les tests sans montrer l'interface, faire ces changements uniquement une fois l'écriture des tests terminés.