

EPCF3

26 Septembre

2024

Dossier projet – Fresh ! La préparation du déploiement d'une application sécurisée.

EXPERNET



CENTRE DE FORMATION
INFORMATIQUE

Table des matières

I.	Liste des compétences du référentiel qui sont couvertes par le projet	2
II.	Expression des besoins du projet pour définir les objectifs et les limites du projet	3
III.	Environnement technique	5
IV.	Réalisations permettant la mise en œuvre des compétences.	6
	Étapes à réaliser pour créer une course :	7
	Jeu de donnée :	7
	Tests unitaires :	8
	Étapes à réaliser pour ajouter des produits à une course :	9
	Jeu de donnée :	9
	Code :	12
	Tests unitaires :	13
	Signature de l'application :	14
	Tests de déploiement :	15
V.	Annexes	17
	Annexe 1 : méthode « buildAlertDialog() »	17
	Annexe 2 : méthode « add() » et méthode « persist() »	18
	Annexe 3 : méthode « canAddOrUpdate() »	19

I. Liste des compétences du référentiel qui sont couvertes par le projet

Préparer le déploiement d'une application sécurisée	Préparer et exécuter les plans de tests d'une application
	Préparer et documenter le déploiement d'une application
	Contribuer à la mise en production dans une démarche DevOps

Le projet couvre obligatoirement les compétences suivantes :

- Préparer et exécuter les plans de tests d'une application
- Préparer et documenter le déploiement d'une application
- Contribuer à la mise en production dans une démarche DevOps

II. Expression des besoins du projet pour définir les objectifs et les limites du projet

Le présent cahier des charges définit les exigences et les spécifications pour le développement d'une application mobile de gestions de courses.

- Contexte

Fresh souhaite mettre en place une solution permettant à n'importe qui, la possibilité de créer des listes de courses depuis son téléphone.

- Fonctionnalités

Les fonctionnalités permettent de savoir ce qui est fonctionnel pour l'utilisateur

- I. L'application a une page d'accueil.

- Un bouton « Créer une course » est cliquable et permet de créer une course dans un pop-up.
- Listage des courses à faire.
 - Chaque élément à une date à faire (sauf cas contraire)
 - Chaque élément est cliquable qui permet d'ajouter la date à faire (si elle n'est pas encore attribuée) avec un pop-up ou afficher un pop-up récapitulatif de la course, où l'on peut modifier la course, manager les produits de la course et marquer la course comme « terminer »
- Un bouton « Voir les courses déjà effectuées » est cliquable et permet de voir la liste de courses ayant été terminées.

- II. L'application a une page de management des produits de course

- Un bouton « Retour » est cliquable et permet de revenir à la page d'accueil.
- Un récapitulatif de la course est disponible avec la date de création de la course, la date à faire et la date quand la course est terminée (sauf cas contraire)
- Un bouton « Modifier » est cliquable et permet de modifier le nom de la course, la date à faire et de marquer la course comme « terminer »
- Listage des produits.
 - Chaque élément affiche la date de création de la course
 - Chaque élément qui n'a pas encore été pris est cliquable et permet d'afficher un pop-up qui contient un bouton « Modifier », qui permet de modifier la ligne du produit et un bouton « Marquer comme pris » qui permet de catégoriser le produit comme pris avec la date à laquelle il a été pris.
- Un bouton « Ajouter un produit » est cliquable et permet d'afficher un pop-up qui contient des champs libres pour indiquer le nom du produit, sa quantité et son unité (champ facultatif)
- Un bouton « Marquer comme terminer » qui permet de classer la course comme terminer avec la date à laquelle elle a été terminée.

- Technologies utilisées

- Développement de l'application
 - Java
 - Gradle Kotlin
 - Android SDK API 34

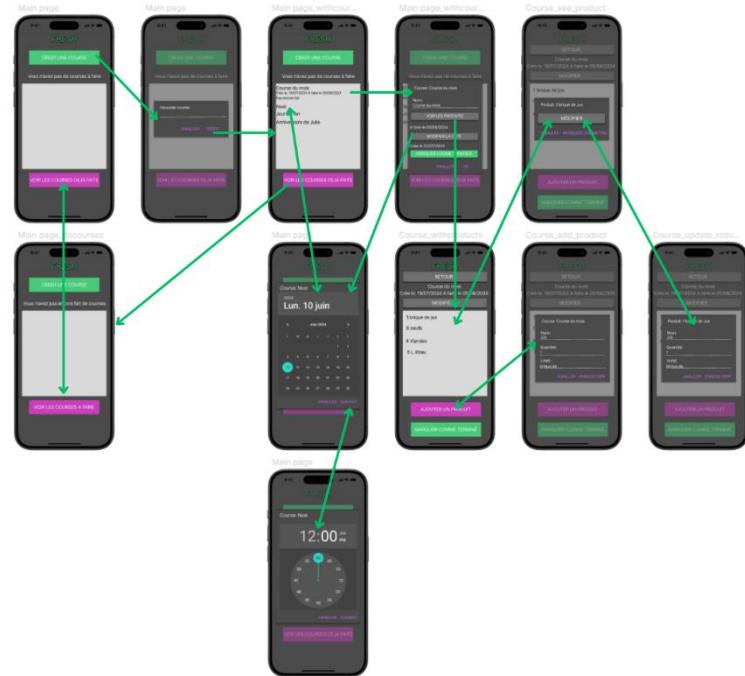
- Livrable et délais

- L'application complète et fonctionnelle doit être livrée conformément aux spécifications.
- La date de livraison prévue pour la plateforme est le 26/07/2024.

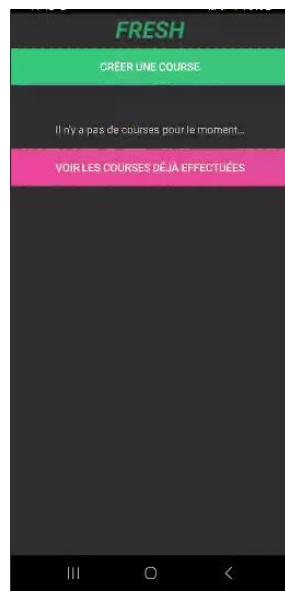
III. Environnement technique

- Développement de l'application
 - Android Studio Koala
- Déploiement de l'application
 - Git 2.45.1 sur Github : [SteveHoareau18/MSP3 \(github.com\)](https://github.com/SteveHoareau18/MSP3)
- Tests de l'application
 - JUnit 4 et tests manuels

IV. Réalisations permettant la mise en œuvre des compétences.



 Enchaînement de maquette 

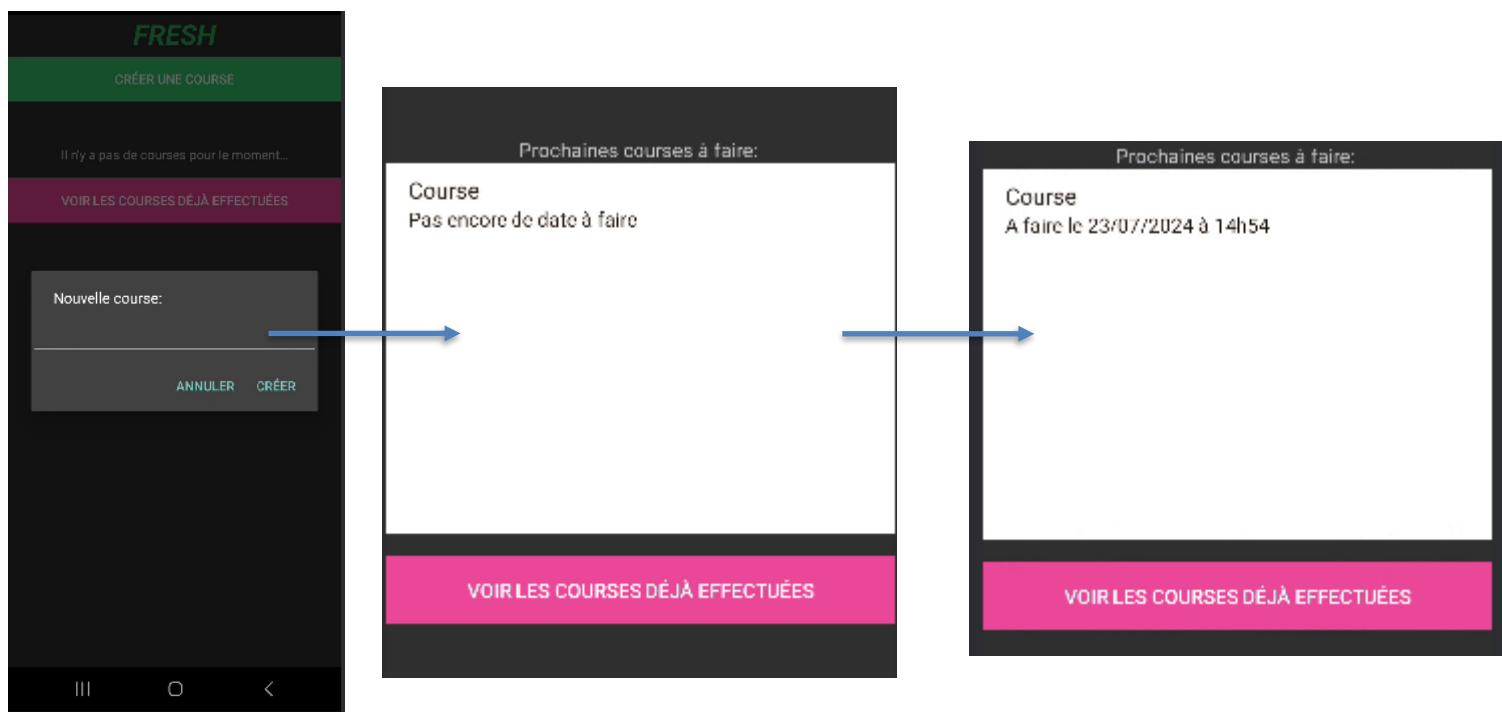


 Page d'accueil



Étapes à réaliser pour créer une course :

- Cliquer sur le bouton « Créer un course »
- Entrer le nom de la course (facultatif) puis cliquer sur le bouton « Créer »
- Définir la date et l'heure à laquelle la course doit être effectuée.



Jeu de donnée :

- Données attendues : Toutes types de données entre 2 et 30 caractères (dont 2 lettres obligatoires) ou Aucune

	Données en entrée	Données obtenues
Aucune donnée	Aucune	« Course »
Données ne correspondantes pas aux critères attendus	<ul style="list-style-type: none"> - « 12 » - « C » - « Coursequifaitplusdetrenteslettres » 	« Course »
Données correspondantes aux critères attendus	<ul style="list-style-type: none"> - « Course du mois » - « Course du 12 » - « Course du 123 » - « Espace » 	<ul style="list-style-type: none"> - « Course du mois » - « Course du 12 » - « Course du 123 » - « Espace »

Code :

```

    /**
     * Creates a new errand with the given name.
     * <p>
     * If the provided name is empty, the errand will be named "Errand" by
     * default.
     * After creation, the errand is added to the repository and a toast
     * message
     * is displayed to inform the user that the errand has been created.
     * Finally, the view is reloaded to reflect the changes.
     * </p>
     *
     * @param name the name of the errand to be created. If empty, defaults to
     * "Errand".
     */
    @Override
    public void create(String name) {
        Errand errand = new Errand();
        String regex = "^(?=(:.*[A-Za-z]){2})[A-Za-z0-9]{1,30}$";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(name);
        boolean matches = matcher.matches();
        if (matches) {
            errand.setName(name.trim());
        } else {
            errand.setName("Course");
            Toast.makeText(getActivity(), "La course ne peut pas être nommée
                ainsi, par défaut elle est donc nommée: Errand",
                Toast.LENGTH_LONG).show();
        }
        getRepository().add(errand);
        Toast.makeText(getActivity(), "La course: " + errand.getName() + " a
            été créée.", Toast.LENGTH_SHORT).show();
        reload();
    }
}

```

👉 Méthode « create() », permettant à créer une course. 🚀

[Voir annexe 2 pour la méthode « getRepository\(\).add\(\) »](#)

Tests unitaires :

```

    @Test
public void testCreateValidErrand() {
    errandCrud.create("Shopping");

    String toastMessage = ShadowToast.getTextOfLatestToast();
    assertEquals("La course: Shopping a été créée.", toastMessage);
}

@Test
public void testCreateInvalidErrand() {
    errandCrud.create("A");

    String toastMessage = ShadowToast.getTextOfLatestToast();
    assertEquals("La course: Course a été créée.", toastMessage);

    errandCrud.create("12");

    toastMessage = ShadowToast.getTextOfLatestToast();
    assertEquals("La course: Course a été créée.", toastMessage);

    errandCrud.create("1234567891011121314151617181920");

    toastMessage = ShadowToast.getTextOfLatestToast();
    assertEquals("La course: Course a été créée.", toastMessage);
}

```

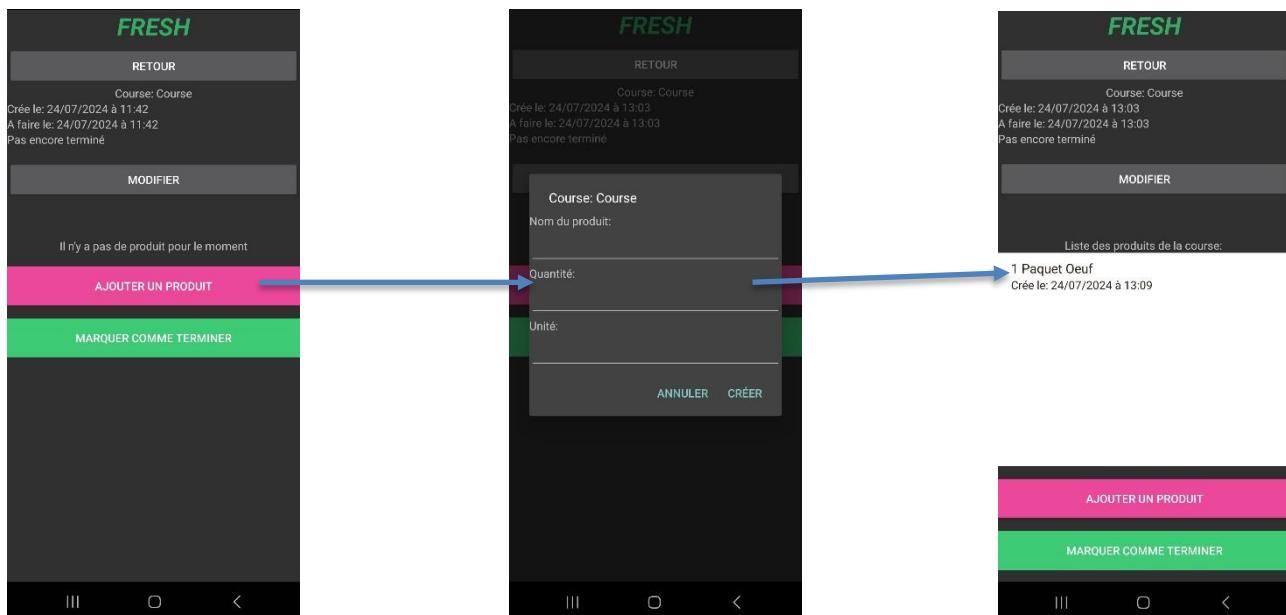
Test Results

Test	Time	Status
Test Results	3 sec 935 ms	Tests passed: 6 of 6 tests – 3 sec 935 ms
ErrandCrudTest	3 sec 935 ms	> Task :app:processDebugUnitTestJavaRes NO-SOURCE
testCreateInvalidErrand	3 sec 496 ms	> Task :app:testDebugUnitTest
testReadErrandNotFound	152 ms	Called loadFromPath(/system/framework/framework-res.apk, true); mode=binary sdk=28
testUpdateErrand	80 ms	resources.arsc in APK 'C:\Users\shoareau.cda.m2\repository\org\robolectric\android-all-instrumented\9-robolectric-4913185-2-i3\android-all-instrumented-9-roboelectric-4913185-2-i3\AndroidManifest.xml'
testSetTitleAndButtonText	72 ms	BUILD SUCCESSFUL in 6s
testGetSortedErrands	60 ms	24 actionable tasks: 1 executed, 23 up-to-date
testCreateValidErrand	75 ms	Build Analyzer results available

15:20:43: Execution finished ':app:testDebugUnitTest --tests "fr.steve.fresh.ErrandCrudTest"'.

Étapes à réaliser pour ajouter des produits à une course :

- Cliquer sur la course
- Cliquer sur le bouton « Voir les produits »
- Cliquer sur le bouton « Ajouter un produit »
- Entrer obligatoirement le nom et la quantité du produit, l'unité est facultative.
- Cliquer sur le bouton « Créer »



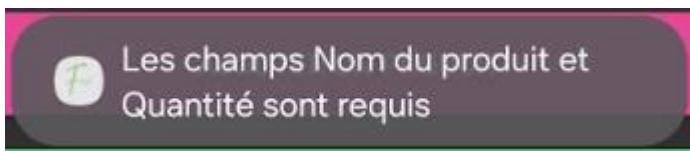
Jeu de donnée :

- Données attendues :
 - o Pour le champ « Nom du produit » : toutes types de données entre 2 et 30 caractères (dont 2 lettres obligatoires)
 - o Pour le champ « Quantité » : un entier strictement supérieur à 0 et inférieur à 1000
 - o Pour le champ « Unité » : toutes types de données entre 1 et 15 caractères (dont 1 lettre obligatoire) ou Aucune

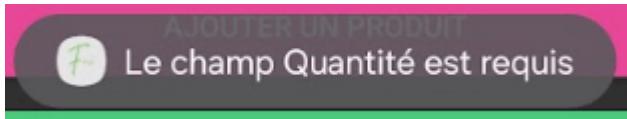
	Données en entrée	Données obtenues
Aucune donnée	Nom du produit : Aucune Quantité : Aucune Unité : Aucune	Aucune - Erreur avec affichage en Toast (1)
Aucun nom du produit	Nom du produit : Aucune Quantité : 1 Unité : L	Aucune - Erreur avec affichage en Toast (1)
Aucune quantité	Nom du produit : « Jus » Quantité : Aucune Unité : L	Aucune - Erreur avec affichage en Toast (2)

Nom du produit ne correspondant pas aux critères	<ul style="list-style-type: none"> - Nom du produit : « J » - Nom du produit : « J1 » - « Nomduproduitquidépassentrentecharactères » <p>Quantité : 1 Unité : L</p>	Aucune <ul style="list-style-type: none"> - Erreur avec affichage en Toast (3)
Quantité du produit ne correspondant pas aux critères	<ul style="list-style-type: none"> - Quantité : 0 - Quantité : -1 - Quantité : 1000 <p>Unité : L</p>	Aucune <ul style="list-style-type: none"> - Erreur avec affichage en Toast (4)
Unité du produit ne correspondant pas aux critères	<ul style="list-style-type: none"> - Unité : « 1 » - Unité : « Unitéquidépassentrentecharactères » 	Aucune <ul style="list-style-type: none"> - Erreur avec affichage en Toast (5)
Données correspondantes aux critères attendus	<ul style="list-style-type: none"> - Nom du produit : « Jus » - Quantité : 1 - Unité : « L » - Nom du produit : « Jus » - Quantité : 1 - Unité : « L » - Nom du produit : « 7up » - Quantité : 999 - Unité : « cl » - Nom du produit : « Oeuf cat 1 » - Quantité : 1 - Unité : « paquet » - Nom du produit : « Carreaux » - Quantité : 15 - Unité : « m2 » 	<ul style="list-style-type: none"> - Nom du produit : « Jus » - Quantité : 1 - Unité : « L » - Nom du produit : « Jus » - Quantité : 1 - Unité : « L » - Nom du produit : « 7up » - Quantité : 999 - Unité : « cl » - Nom du produit : « Oeuf cat 1 » - Quantité : 1 - Unité : « paquet » - Nom du produit : « Carreaux » - Quantité : 15 - Unité : « m2 »

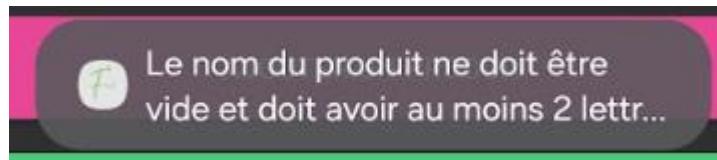
Toast (1) :



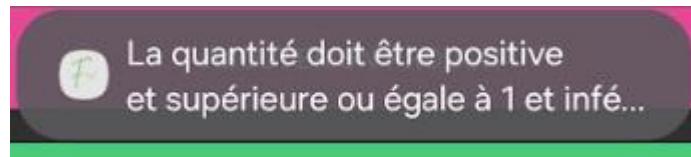
Toast (2) :



Toast (3) :



Toast (4) :



Toast (5) :



Code :

```


/**
 * Updates a product in the repository.
 *
 * This method receives a product supplier, extracts the product from
 * the supplier, and checks if the product can be added or
 * updated based on the specified criteria (name, quantity,
 * unit). If the product is valid, the following operations are
 * performed:
 * <ul>
 *   <li>Trims any extra whitespace around the product's name and
 *       unit.</li>
 *   <li>Searches for an existing product with the same name in the
 *       repository.</li>
 *   <li>If an existing product is found with a different ID, it is
 *       deleted and its quantity is added to the current product's
 *       quantity.</li>
 *   <li>Adds or updates the product in the repository.</li>
 *   <li>Displays a confirmation message to the user.</li>
 *   <li>Reloads the data after the update.</li>
 * </ul>
 *
 * @param productSupplier A supplier of the product. The product
 * provided by this supplier is the one that will be updated.
 */
@Override
public void update(Supplier<Product> productSupplier) {
    Product product = productSupplier.get();

    if (!canAddOrUpdate(product.getName(), product.getQuantity(),
        product.getUnit())) return;

    product.setUnit(product.getUnit().trim());
    product.setName(product.getName().trim());

    AtomicBoolean find = new AtomicBoolean(true);
    Product findProduct = ((ProductRepository)
        getRepository()).findByNameInCourse(product.getName(),
        errand).orElseGet(() -> {
        find.set(false);
        return product;
    });

    int new_quantity = 0;
    if (find.get() && findProduct.getId() != product.getId()) {
        new_quantity += findProduct.getQuantity();
        delete(findProduct);
    }
    product.setQuantity(product.getQuantity() + new_quantity);
    getRepository().add(product);
    Toast.makeText(getActivity(), "Le produit " + product.getName() +
        " a été mis à jour", Toast.LENGTH_LONG).show();
    reload();
}


```

[Voir annexe 3 pour la méthode « canAddOrUpdate\(\) »](#)

Tests unitaires :

```

@Test
public void testCanAddOrUpdate_withValidInput_shouldReturnTrue() {
    assertTrue(productCrud.canAddOrUpdate("Jus", 1, ""));
    assertTrue(productCrud.canAddOrUpdate("Jus", 1, "cl"));
    assertTrue(productCrud.canAddOrUpdate("Jus", 1, ""));
}

@Test
public void testCanAddOrUpdate_withInvalidQuantity_shouldReturnFalse() {
    assertFalse(productCrud.canAddOrUpdate("Jus", -1, ""));
    assertFalse(productCrud.canAddOrUpdate("Jus", 10000, ""));
}

@Test
public void testCanAddOrUpdate_withInvalidName_shouldShowToast() {
    productCrud.canAddOrUpdate("", 1, "");
    String toastMessage = ShadowToast.getTextOfLatestToast();
    assertTrue(toastMessage.contains("Le nom du produit ne doit pas être
        vide et doit avoir au moins 2 lettres et ne doit pas dépasser 30
        caractères"));

    productCrud.canAddOrUpdate("J", 1, "");
    toastMessage = ShadowToast.getTextOfLatestToast();
    assertTrue(toastMessage.contains("Le nom du produit ne doit pas être
        vide et doit avoir au moins 2 lettres et ne doit pas dépasser 30
        caractères"));
}

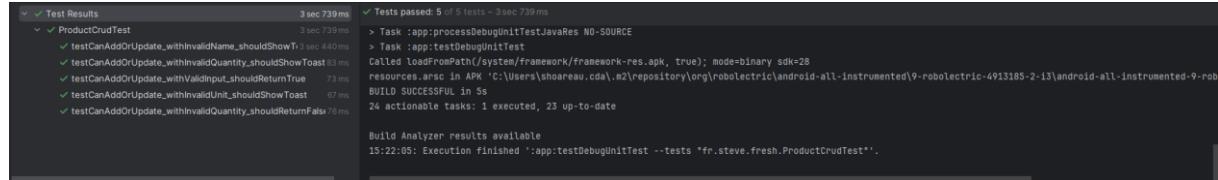
@Test
public void testCanAddOrUpdate_withInvalidUnit_shouldShowToast() {
    assertTrue(productCrud.canAddOrUpdate("Jus", 1, "x"));

    productCrud.canAddOrUpdate("Jus", 1, "abcdefghijklmnopqrstuvwxyz");
    String toastMessage = ShadowToast.getTextOfLatestToast();
    assertTrue(toastMessage.contains("L'unité du produit peut être au moins
        1 lettre et ne doit pas dépasser 15 caractères"));
}

@Test
public void testCanAddOrUpdate_withInvalidQuantity_shouldShowToast() {
    productCrud.canAddOrUpdate("Jus", -1, "");
    String toastMessage = ShadowToast.getTextOfLatestToast();
    assertTrue(toastMessage.contains("La quantité doit être positive et
        supérieure ou égale à 1 et inférieure à 1000"));

    productCrud.canAddOrUpdate("Jus", 10000, "");
    toastMessage = ShadowToast.getTextOfLatestToast();
    assertTrue(toastMessage.contains("La quantité doit être positive et
        supérieure ou égale à 1 et inférieure à 1000"));
}

```



Signature de l'application :

Generate Signed App Bundle or APK

Module

Key store path

Key store password

Key alias

Key password

Remember passwords

Tests de déploiement :

```
name: Android CI

on:
  push:
    branches: [ "master" ]
  pull_request:
    branches: [ "master" ]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4

      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: '17'
          distribution: 'temurin'
          cache: gradle

      - name: Install Android SDK
        uses: android-actions/setup-android@v2
        with:
          api-level: '34'
          target: 'android-34'
          build-tools: '34.0.0'

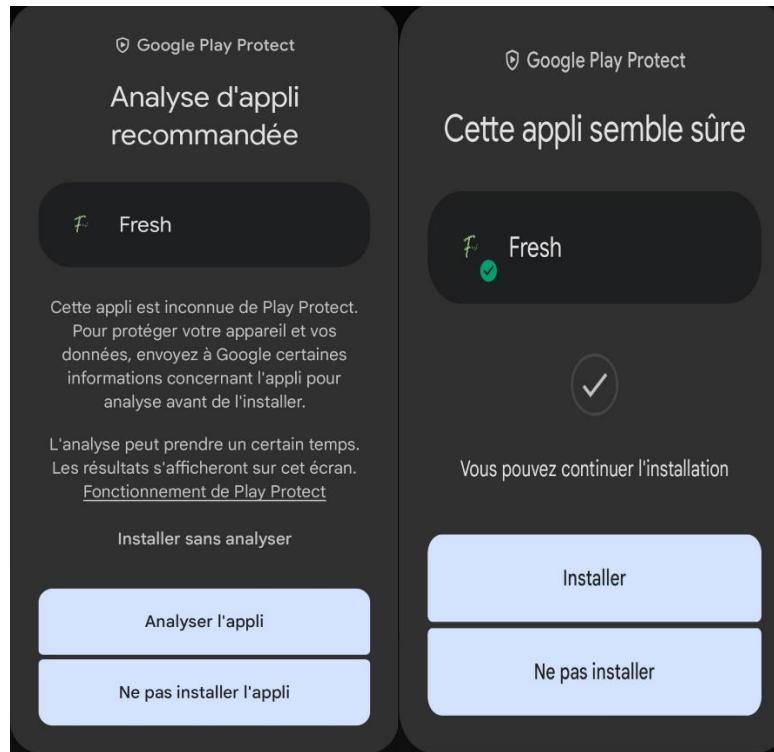
      - name: Grant execute permission for gradlew
        run: chmod +x gradlew

      - name: Build with Gradle
        run: ./gradlew build

      - name: Run unit tests
        run: ./gradlew test
```

Workflow de déploiement sur Github





⚠️ Lors de l'installation de l'application ⚠️

V. Annexes

Annexe 1 : méthode « buildAlertDialog() »

La méthode « buildAlertDialog() » se situe dans la classe « Dialog<P extends IPage> » et permet de générer un pop-up.

```
/**  
 * Builds and shows an alert dialog with the specified title, layout, and  
 * buttons.  
 * <p>  
 * This method creates an alert dialog with a custom view provided by the  
 * `layoutSupplier`,  
 * and configures the positive and negative buttons with their respective  
 * callbacks.  
 * </p>  
 *  
 * @param title the title of the dialog.  
 * @param layoutSupplier a supplier that provides the custom view to be  
 * set in the dialog.  
 * @param positiveButton the text for the positive button.  
 * @param positiveCallback the callback to be executed when the positive  
 * button is clicked.  
 * @param negativeButton the text for the negative button.  
 * @param negativeCallback the callback to be executed when the negative  
 * button is clicked.  
 */  
public void buildAlertDialog(String title, Supplier<View> layoutSupplier,  
String positiveButton, DialogInterface.OnClickListener positiveCallback,  
String negativeButton, DialogInterface.OnClickListener negativeCallback) {  
    new AlertDialog.Builder(activity)  
        .setMessage(title)  
        .setView(layoutSupplier.get())  
        .setPositiveButton(positiveButton, positiveCallback)  
        .setNegativeButton(negativeButton, negativeCallback)  
        .show();  
}
```

Annexe 2 : méthode « add() » et méthode « persist() »

Les méthodes « add() » et « persist() » se trouvent dans la classe « Repository<T extends Entity> ».

La méthode « add() » permet d'ajouter une entité à la liste, tandis que la méthode « persist() » permet d'enregistrer l'entité.

L'entité peut être une course ou un produit, et chaque type d'entité possède sa propre liste associée.

```
/**
 * Adds an entity to the list. If an entity with the same ID already exists
 * in the list,
 * it is removed before the new entity is added.
 * <p>
 * This method ensures that the list does not contain duplicate entities
 * based on their IDs.
 * After adding the entity to the list, the entity is persisted.
 * </p>
 *
 * @param entity the entity to be added to the list. If an entity with the
 * same ID exists,
 * it is replaced by the new entity.
 */
public void add(T entity) {
    this.list.stream().filter(x -> x.getId() ==
        entity.getId()).findFirst().ifPresent(this.list::remove);
    this.list.add(entity);
    persist(entity);
}

/**
 * Persists the given entity by converting it to a JSON string and storing
 * it
 * in the shared preferences using a unique key.
 * <p>
 * The key is constructed using the class name and the ID of the entity to
 * ensure uniqueness.
 * </p>
 *
 * @param entity the entity to be persisted. It is converted to a JSON
 * string and stored
 * in the shared preferences.
 */
private void persist(T entity) {
    String key = entity.getClass().getName() + "_" + entity.getId();
    String json = gson.toJson(entity);
    MainActivity.getActivityReference().get().edit().putString(key,
        json).apply();
}
```

La méthode « persist() » utilise le Shared Preferences.

Annexe 3 : méthode « canAddOrUpdate() »

```


/**
 * Checks if a product can be added or updated.
 *
 * @param name      the name of the product
 * @param quantity  the quantity of the product
 * @param unit      the unit of the product
 * @return true if the product can be added or updated, false
 * otherwise
 */
public boolean canAddOrUpdate(String name, int quantity, String
unit) {
    name = name.trim();
    String regex = "^(?=.*[A-Za-z])\\d{1,30}$";
    Pattern pattern = Pattern.compile(regex);
    Matcher matcherName = pattern.matcher(name);

    String regexUnit = "^(?=.*[A-Za-z])\\d{1,15}$";
    Pattern patternUnit = Pattern.compile(regexUnit);
    Matcher matcherUnit = patternUnit.matcher(unit);

    boolean matchesName = matcherName.matches();
    boolean matchesUnit = matcherUnit.matches();
    if (!matchesName) {
        Toast.makeText(getActivity(), "Le nom du produit ne doit pas
être vide et doit avoir au moins 2 lettres et ne doit pas dépasser
30 caractères", Toast.LENGTH_LONG).show();
        return false;
    }
    if (!unit.isEmpty() && !matchesUnit) {
        Toast.makeText(getActivity(), "L'unité du produit peut être
au moins 1 lettre et ne doit pas dépasser 15 caractères",
Toast.LENGTH_LONG).show();
        return false;
    }
    if (quantity < 1 || quantity > 1000) {
        Toast.makeText(getActivity(), "La quantité doit être positive
et supérieure ou égale à 1 et inférieure à 1000",
Toast.LENGTH_LONG).show();
        return false;
    }
    return true;
}


```