Réalisation par :

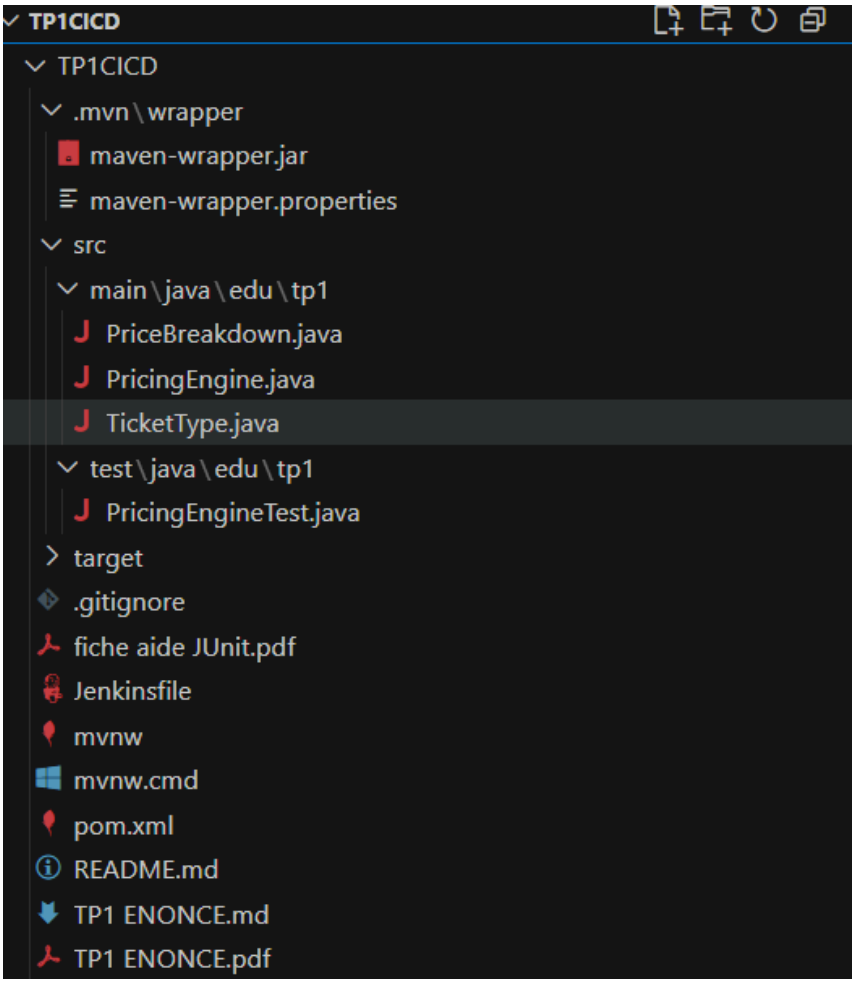
Floyd LAFORGE	Nathan VANDENBOSSCHE
Crystal BOUDAUD	Maxime LANGRAND

README :

Projet Mini Moteur de Tarification Cinéma :

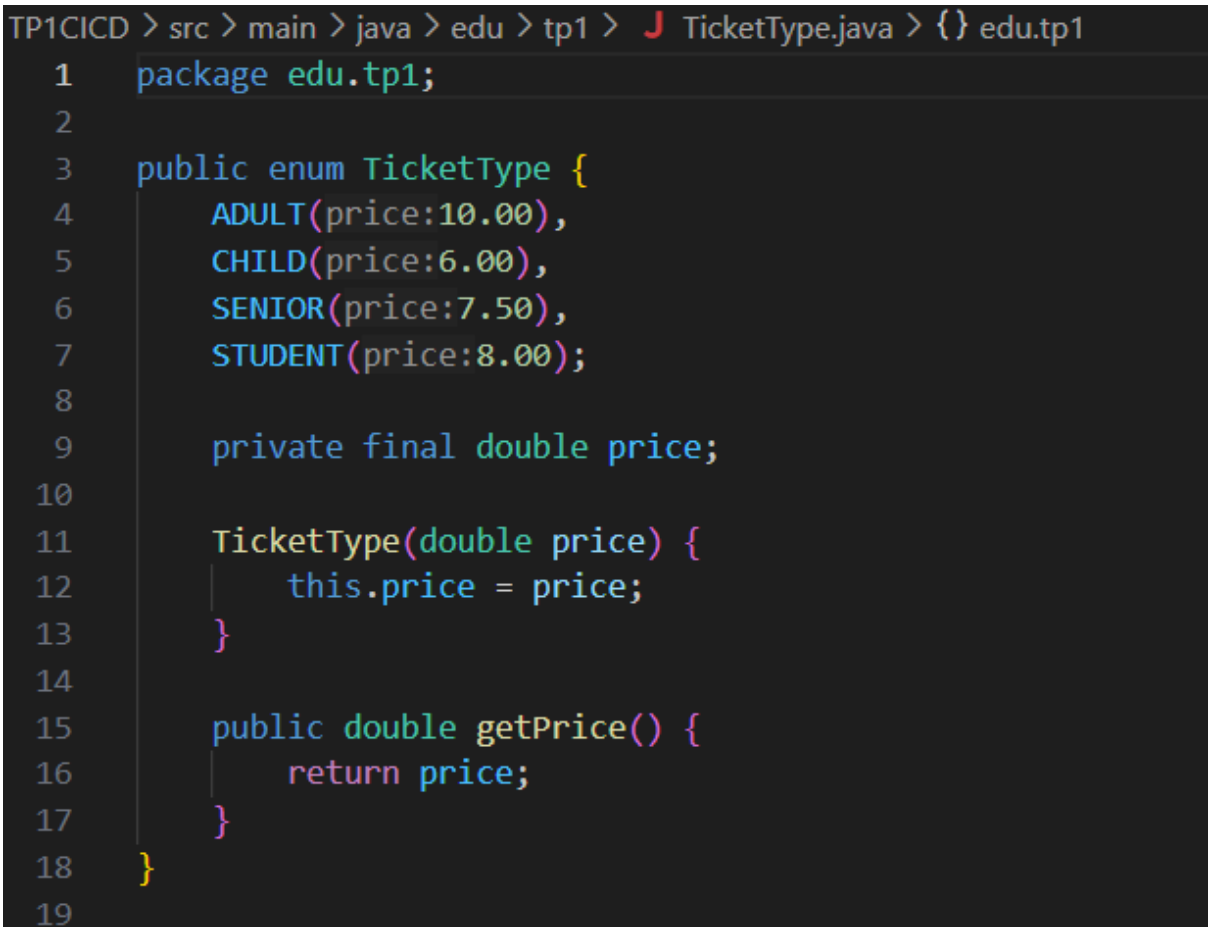
Ce projet a pour objectif de développer un module Java pour calculer le prix total d'une commande de billets de cinéma selon différentes règles tarifaires. Les critères pris en compte sont le type de billet (adulte, enfant, sénior, étudiant), l'utilisation éventuelle de la 3D, le jour de la séance (avec une remise spécifique le mercredi) et l'achat en groupe (à partir de 4 billets). Ce moteur de tarification doit être structuré avec un code clair, testé avec JUnit 5 et permettre la mesure de la couverture de tests via JaCoCo. Une intégration avec Jenkins via Maven est prévue en option.

Arborescence :



Fichiers Java et leurs rôles :

1. TicketType.java
Ce fichier définit une énumération qui représente les différents types de billets disponibles : ADULT, CHILD, SENIOR, STUDENT. Chaque valeur correspond à un type spécifique de ticket.



2. PriceBreakdown.java
Cette classe encapsule le résultat détaillé du calcul du prix d'une commande. Elle contient :

subtotal : la somme des prix de base sans remises ni suppléments.

wednesdayDisc : la remise appliquée si la séance est le mercredi (-20%).

threeDSurcharge : le supplément appliqué par billet si la séance est en 3D (+2€ par billet).

groupDisc : la remise de groupe si 4 billets ou plus (-10€).

total : le montant final, arrondi au centime.

Cette structure permet de présenter un récapitulatif clair et détaillé du calcul du prix.

TP1CICD > src > main > java > edu > tp1 > PriceBreakdown.java > {} edu.tp1

```java
package edu.tp1;

import java.util.StringJoiner;

public final class PriceBreakdown {
    private final double subtotal;
    private final double wednesdayDiscount;
    private final double threeDSurcharge;
    private final double groupDiscount;
    private final double total;

    public PriceBreakdown(double subtotal, double wednesdayDiscount, double threeDSurcharge, double groupDiscount,
        this.subtotal = subtotal;
        this.wednesdayDiscount = wednesdayDiscount;
        this.threeDSurcharge = threeDSurcharge;
        this.groupDiscount = groupDiscount;
        this.total = total;
    }

    public double getSubtotal() { return subtotal; }
    public double getWednesdayDiscount() { return wednesdayDiscount; }
    public double getThreeDSurcharge() { return threeDSurcharge; }
    public double getGroupDiscount() { return groupDiscount; }
    public double getTotal() { return total; }

    @Override
    public String toString() {
        return new StringJoiner(delimiter:", ", PriceBreakdown.class.getSimpleName() + "[", suffix:"]")
                .add("subtotal=" + subtotal)
                .add("wednesdayDiscount=" + wednesdayDiscount)
                .add("threeDSurcharge=" + threeDSurcharge)
                .add("groupDiscount=" + groupDiscount)
                .add("total=" + total)
                .toString();
    }
}
```

3. PricingEngine.java
Il s'agit du cœur du projet. Cette classe contient :

La méthode basePrice(TicketType) qui retourne le prix de base pour chaque type de billet.

La méthode computeTotal(List<TicketType>, boolean, DayOfWeek) qui calcule le prix total en appliquant dans l'ordre :

Calcul du prix total sans remises ni suppléments (subtotal).

Application de la remise du mercredi (-20% si applicable).

Ajout du supplément 3D (+2€ par billet si séance 3D).

Application de la remise groupe (-10% si 4 billets ou plus).

Un arrondi systématique des montants au centime près.

Cette organisation garantit la lisibilité, la maintenabilité et la possibilité d'ajouter facilement des règles tarifaires.

TP1CICD > src > main > java > edu > tp1 > PricingEngine.java > {} edu.tp1

```java
package edu.tp1;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.time.DayOfWeek;
import java.util.List;

public class PricingEngine {

    public double basePrice(TicketType type) {
        if (type == null) {
            throw new IllegalArgumentException(s:"TicketType cannot be null.");
        }
        return type.getPrice();
    }

    public PriceBreakdown computeTotal(List<TicketType> tickets, DayOfWeek day, boolean is3D) {
        if (tickets == null || day == null) {
            throw new IllegalArgumentException(s:"Tickets list and day cannot be null.");
        }

        double subtotal = tickets.stream()
                                 .mapToDouble(this::basePrice)
                                 .sum();

        double priceAfterWednesdayDiscount = subtotal;
        double wednesdayDiscount = 0;
        if (day == DayOfWeek.WEDNESDAY) {
            wednesdayDiscount = roundToCents(subtotal * 0.20);
            priceAfterWednesdayDiscount = subtotal - wednesdayDiscount;
        }

        double threeDSurcharge = 0;
        if (is3D) {
            threeDSurcharge = tickets.size() * 2.00;
        }
        double priceAfter3D = priceAfterWednesdayDiscount + threeDSurcharge;
```

```
38
39          double groupDiscount = 0;
40          if (tickets.size() >= 4) {
41              groupDiscount = roundToCents(priceAfter3D * 0.10);
42          }
43
44          double total = priceAfter3D - groupDiscount;
45
46          return new PriceBreakdown(
47              roundToCents(subtotal),
48              wednesdayDiscount,
49              threeDSurcharge,
50              groupDiscount,
51              roundToCents(total)
52          );
53      }
54
55      private double roundToCents(double value) {
56          return BigDecimal.valueOf(value).setScale(newScale:2, RoundingMode.HALF_UP).doubleValue();
57      }
58  }
59
```

4. PricingEngineTest.java
Ce fichier fournit une suite de tests unitaires avec JUnit 5 pour vérifier le bon fonctionnement de la classe PricingEngine. Les tests couvrent :

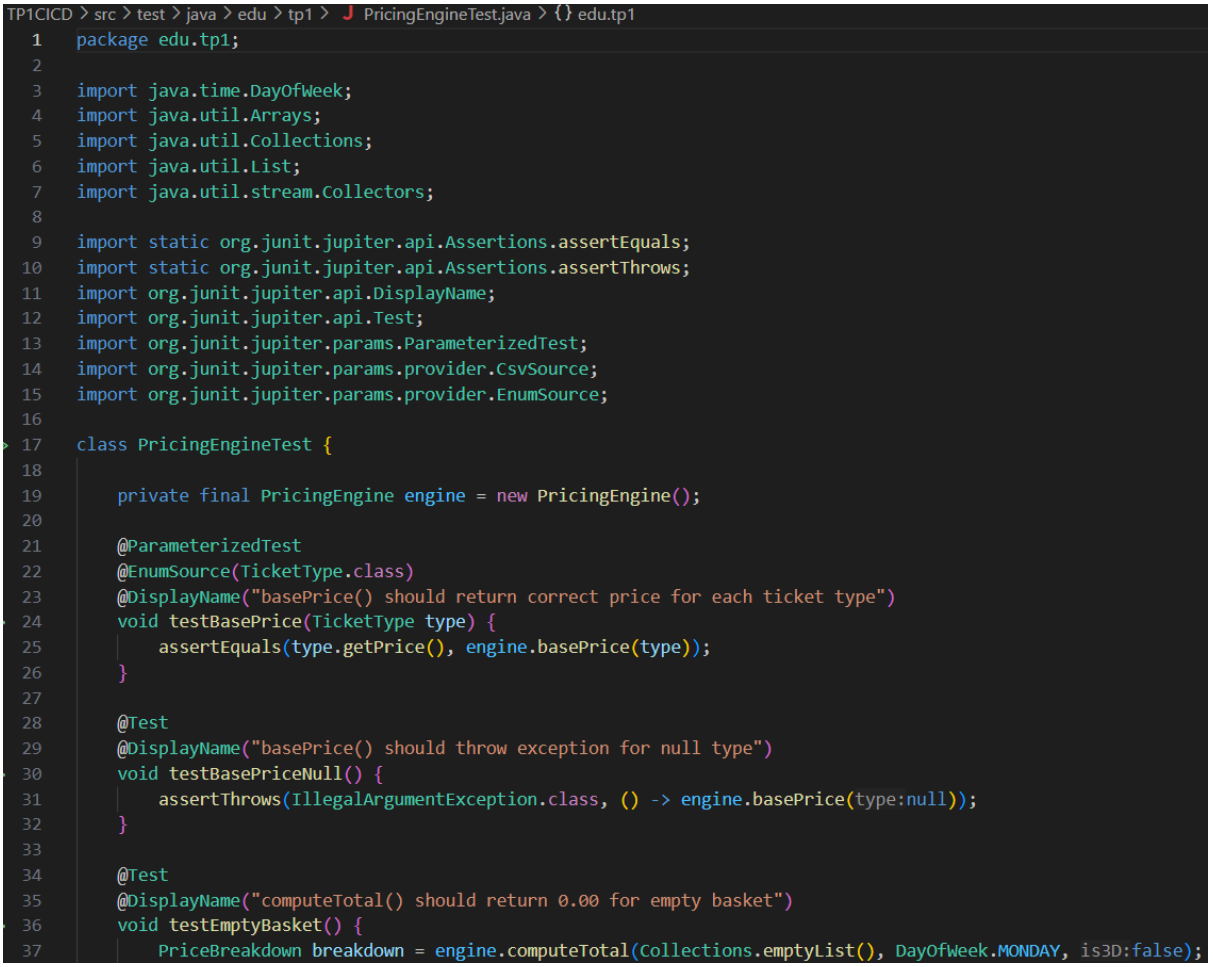
La validation des prix de base selon les types de billets.

Le comportement du moteur pour un panier vide.

L'application isolée des différentes remises et suppléments (mercredi, 3D, groupe).

La gestion combinée de plusieurs règles tarifaires.

La gestion des erreurs, notamment la présence d'un type de billet nul.




```java
package edu.tp1;

import java.time.DayOfWeek;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.stream.Collectors;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.EnumSource;

class PricingEngineTest {

    private final PricingEngine engine = new PricingEngine();

    @ParameterizedTest
    @EnumSource(TicketType.class)
    @DisplayName("basePrice() should return correct price for each ticket type")
    void testBasePrice(TicketType type) {
        assertEquals(type.getPrice(), engine.basePrice(type));
    }

    @Test
    @DisplayName("basePrice() should throw exception for null type")
    void testBasePriceNull() {
        assertThrows(IllegalArgumentException.class, () -> engine.basePrice(type:null));
    }

    @Test
    @DisplayName("computeTotal() should return 0.00 for empty basket")
    void testEmptyBasket() {
        PriceBreakdown breakdown = engine.computeTotal(Collections.emptyList(), DayOfWeek.MONDAY, is3D:false);
```

```
        assertEquals(0.00, breakdown.getTotal());
    }

    @Test
    @DisplayName("computeTotal() with Wednesday discount only")
    void testWednesdayDiscount() {
        List<TicketType> tickets = List.of(TicketType.ADULT, TicketType.SENIOR, TicketType.CHILD);
        PriceBreakdown breakdown = engine.computeTotal(tickets, DayOfWeek.WEDNESDAY, is3D:false);
        assertEquals(18.80, breakdown.getTotal());
    }

    @Test
    @DisplayName("computeTotal() with 3D surcharge only")
    void test3DSurcharge() {
        List<TicketType> tickets = List.of(TicketType.ADULT, TicketType.CHILD);
        PriceBreakdown breakdown = engine.computeTotal(tickets, DayOfWeek.MONDAY, is3D:true);
        assertEquals(20.00, breakdown.getTotal());
    }

    @Test
    @DisplayName("computeTotal() with Group discount only")
    void testGroupDiscount() {
        List<TicketType> tickets = Collections.nCopies(n:4, TicketType.STUDENT);
        PriceBreakdown breakdown = engine.computeTotal(tickets, DayOfWeek.MONDAY, is3D:false);
        assertEquals(28.80, breakdown.getTotal());
    }

    @ParameterizedTest
    @CsvSource({
        // On utilise des apostrophes pour que la liste de tickets soit lue comme un seul argument
        "'ADULT,CHILD',        true,  WEDNESDAY, 16.80",
        "'ADULT:4',            true,  WEDNESDAY, 36.00",
        "'STUDENT:4',          false, MONDAY,    28.80",
        "'ADULT,SENIOR,CHILD', false, WEDNESDAY, 18.80"
    })
    @DisplayName("computeTotal() should handle combinations of rules correctly")
    void testCombinedRules(String ticketsStr, boolean is3D, DayOfWeek day, double expectedTotal) {
        List<TicketType> tickets;
        // Gère la notation "TYPE:QUANTITE"
        if (ticketsStr.contains(s:":")) {
            String[] parts = ticketsStr.split(regex:":");
            TicketType type = TicketType.valueOf(parts[0]);
            int count = Integer.parseInt(parts[1]);
            tickets = Collections.nCopies(count, type);
        } else {
            // Gère la notation "TYPE1,TYPE2,..." pour n'importe quel nombre de tickets
            tickets = Arrays.stream(ticketsStr.split(regex:","))
                            .map(TicketType::valueOf)
                            .collect(Collectors.toList());
        }

        PriceBreakdown breakdown = engine.computeTotal(tickets, day, is3D);
        // On utilise une tolérance (delta) pour les comparaisons de doubles
        assertEquals(expectedTotal, breakdown.getTotal(), 0.01);
    }
}
```

Webhook :

Mise en place d'un Webhook pour avoir un véritable flux de travail d'intégration continue (CI).

Phase 1 : Préparation du Projet Local et GitHub

1. Initialisation du projet local :
    - Le projet Java Maven existant a été transformé en dépôt Git avec la commande `git init -b main`.
    - Tous les fichiers du projet ont été ajoutés (`git add .`) et sauvegardés dans un premier commit (`git commit -m "Premier commit"`).

2. Création du dépôt distant :
    - Un nouveau dépôt entièrement vide a été créé sur GitHub. Aucune option (`README`, `.gitignore`) n'a été sélectionnée pour éviter les conflits d'historiques.

3. Connexion et premier `push` :
    - Le dépôt local a été lié au dépôt distant GitHub avec `git remote add origin <URL_du_dépôt>`.
    - Le contenu local a été envoyé vers GitHub avec `git push -u origin main`.

Phase 2 : Configuration de Jenkins

1. Installation des plugins :
    - Les plugins essentiels ont été installés via `Administrer Jenkins > Plugins` :
        - `GitHub Integration`
        - `Pipeline`
        - `HTML Publisher plugin`

2. Création du `Jenkinsfile` :
    - Un fichier `Jenkinsfile` a été ajouté à la racine du projet pour décrire le pipeline.

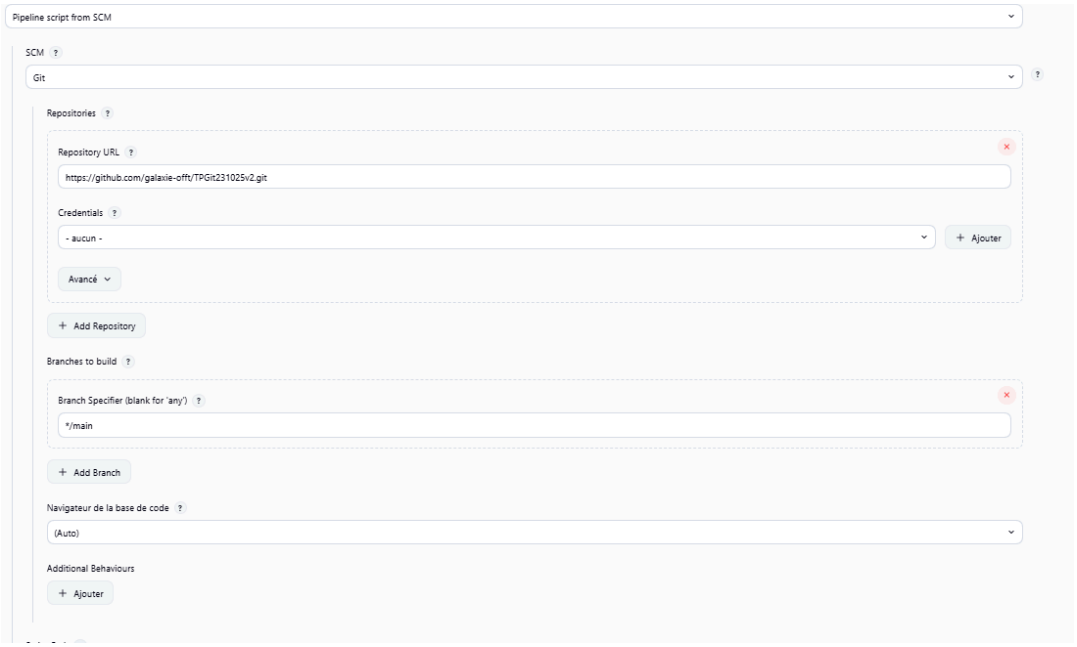
Jenkinsfile

```groovy
pipeline {
  agent any
  stages {
    stage('Build & Test') {
      steps {
        script {
          if (isUnix()) {
            sh './mvnw -B -V clean verify'
          } else {
            bat '.\\mvnw.cmd -B -V clean verify'
          }
        }
      }
    }
    stage('Archive') {
      steps {
        archiveArtifacts artifacts: 'target/*.jar', fingerprint: true
        publishHTML(target: [allowMissing: true, alwaysLinkToLastBuild: true, keepAll: true,
          reportDir: 'target/site/jacoco', reportFiles: 'index.html', reportName: 'JaCoCo'])
      }
    }
  }
}
```

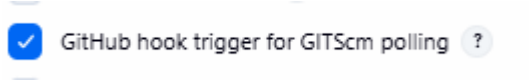
Contenu du fichier Jenkinsfile :

- Ce fichier a été versionné et poussé sur GitHub.

3. Création du Pipeline Jenkins :
   - Un nouvel item de type Pipeline a été créé.
   - Dans la configuration, la Définition a été passée à "Pipeline script from SCM".
   - Le SCM a été configuré sur Git, pointant vers l'URL du dépôt GitHub.

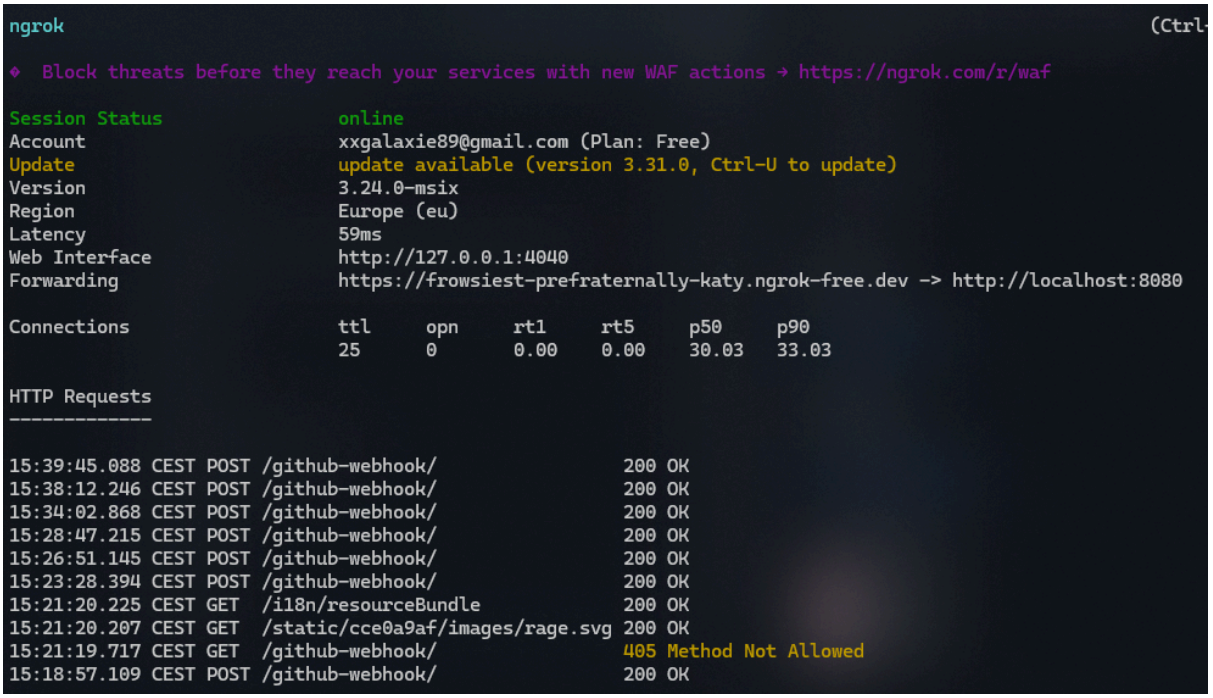


Le déclencheur "GitHub hook trigger for GITScm polling" a été activé.

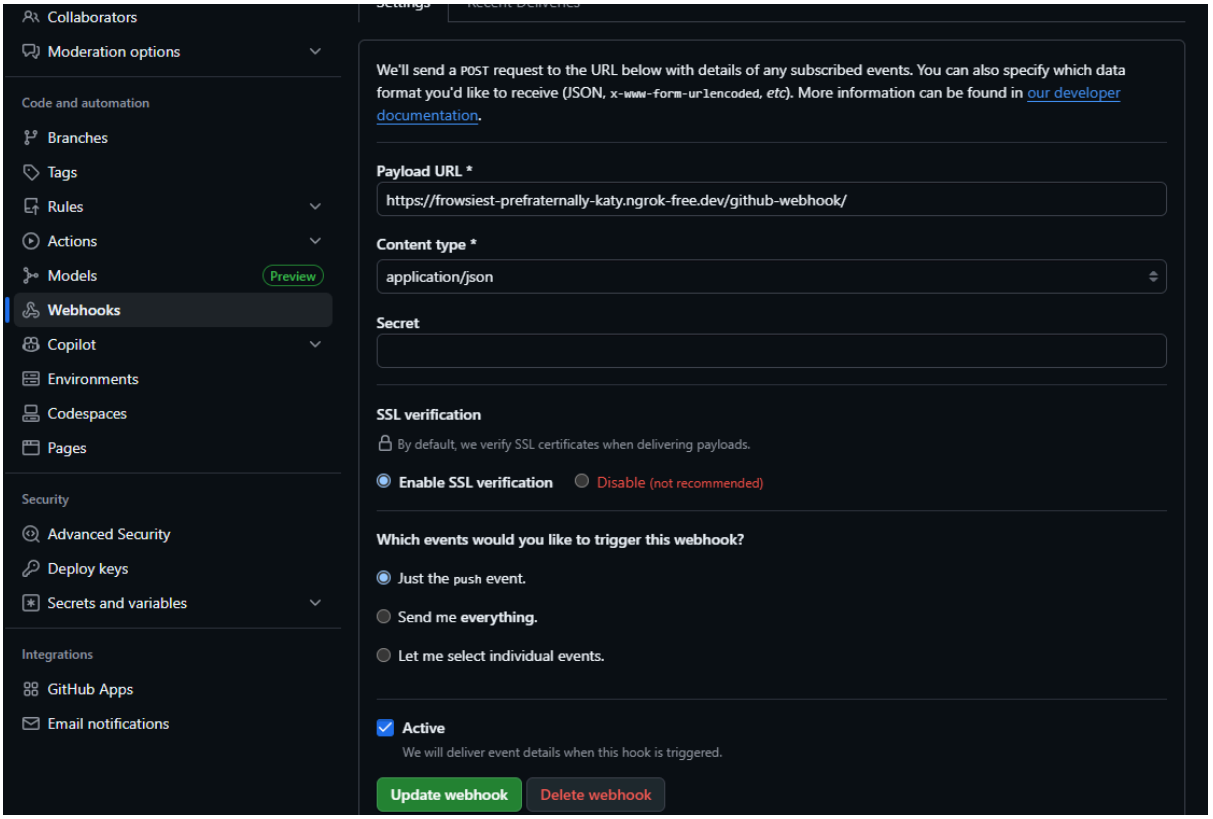


Phase 3 : Configuration du Webhook et Connexion Externe

1. Exposition de Jenkins local via ngrok :
   - ngrok a été téléchargé et installé.
   - Un tunnel a été créé pour exposer le port 8080 de Jenkins à une URL publique avec la commande ngrok http 8080.



2. Création du Webhook sur GitHub :
    - Dans les Settings > Webhooks du dépôt GitHub, un nouveau webhook a été ajouté.
    - Payload URL : L'URL publique fournie par ngrok a été utilisée, suivie de /github-webhook/ (ex: https://<id-ngrok>.ngrok-free.app/github-webhook/).
    - Content type : application/json.
    - Trigger : "Just the push event" a été sélectionné.



Phase 4 : Sécurité et Finalisation

1. Gestion des permissions :
    - Un premier build a échoué car il a été déclenché par un utilisateur "anonyme" sans les droits nécessaires.
    - Correction : Dans Administrer Jenkins > Configurer la sécurité globale > Autorisation, la permission Job > Build a été accordée au groupe Anonymous Users pour permettre au webhook de fonctionner sur une installation locale non sécurisée.

Résultat : Le système est maintenant pleinement opérationnel. Chaque git push vers la branche main du dépôt GitHub déclenche automatiquement le pipeline Jenkins, qui compile, teste (avec génération des rapports JaCoCo et JUnit) et archive les artéfacts du projet.