# FModule 324 Portfolio Racine Thibaud

## Introduction

### Objectif du projet

Ce projet a pour objectif de documenter, à travers un portfolio détaillé, l’ensemble des étapes du cycle de vie de développement logiciel : planification (Plan), développement (Dev), génération (Build), tests (Tests) et déploiement (Deploy). Le portfolio illustre ces étapes à travers la création de deux applications indépendantes, mettant en œuvre des pratiques modernes de développement, d'intégration continue et de livraison continue (CI/CD). Ce travail met en avant les choix techniques, les défis rencontrés, et les solutions apportées pour mener à bien ces réalisations.

### Présentation des deux applications

Application 1: Add-on Firefox – Todo App

Application 2: Flask Login System

## Addon Firefox:

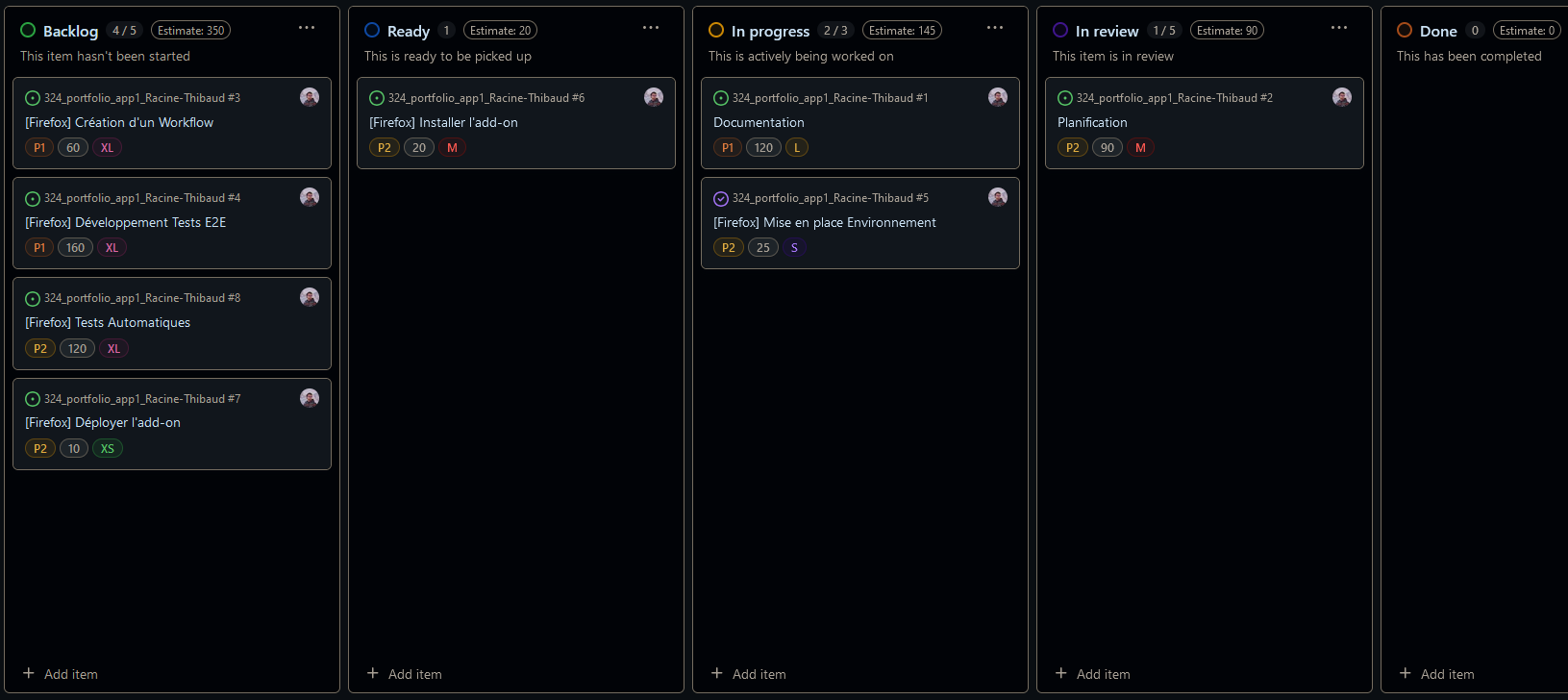
Addon avec une liste de TODO à tester.

## 1. Analyse et Planification Initiale

Journal des commits (datés)  
Développement et tests

### Planification

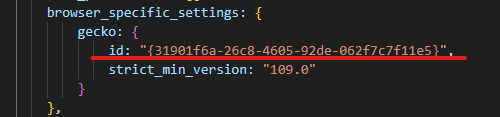
Utilisation de GitHub Project avec une planification de type Kanban.  
Les étapes sont :

- Backlog : Objet qui n'a pas été commencé.  
- Ready : Objet prêt à être commencé.  
- In Progress : Objet en cours de développement.  
- In Review : Objet en cours d'analyse.  
- Done : Objet terminé

Voici ma planification Initiale pour le projet addon Firefox

## 2. Configuration de l’Environnement de Développement

### Étapes installation de l’add-on Todo

1. Télécharger le zip du projet nommé « addon-todo-app.zip » qui se trouve sur teams.
2. Dans « wxt.config.ts » Modifier l’ID de gecko avec l’ID donné par le professeur.
3. Dans OneDrive, créer un dossier à la racine nommé “addon\_todo” et ajouter un fichier JSON nommé “taskList.json”.
4. Exécuter « npm install » dans le dossier local et « npm run dev:firefox ».
5. Une page Firefox va s’ouvrir. Appuyer sur le bouton des extensions, ouvrir l’app Todo et se connecter avec eduvaud.
6. Une fois connecté et la page actualisée, les tâches apparaissent dans la fenêtre Todo.
7. Pour accéder au debug de l’app, taper dans la barre de recherche `about:debugging`.

#### Commande

npm run zip:firefox

### Déployer manuellement l’add-on Firefox

1. Créer un compte Firefox.  
2. Générer des clés API en passant par le site Firefox add-ons > Atelier sur les extensions > Submit or manage extensions > Outils > Gérer les clés API.  
3. Enregistrer les clés générées et proposer un nouveau module.  
4. Accepter les accords, ajouter un nom et continuer.  
5. Ajouter le fichier zip de Firefox qui se trouve dans «.output » de l’app, choisir le fichier `firefox.zip`.  
6. Donner le code source en important le fichier `source.zip` qui se trouve également dans le dossier «.output ».

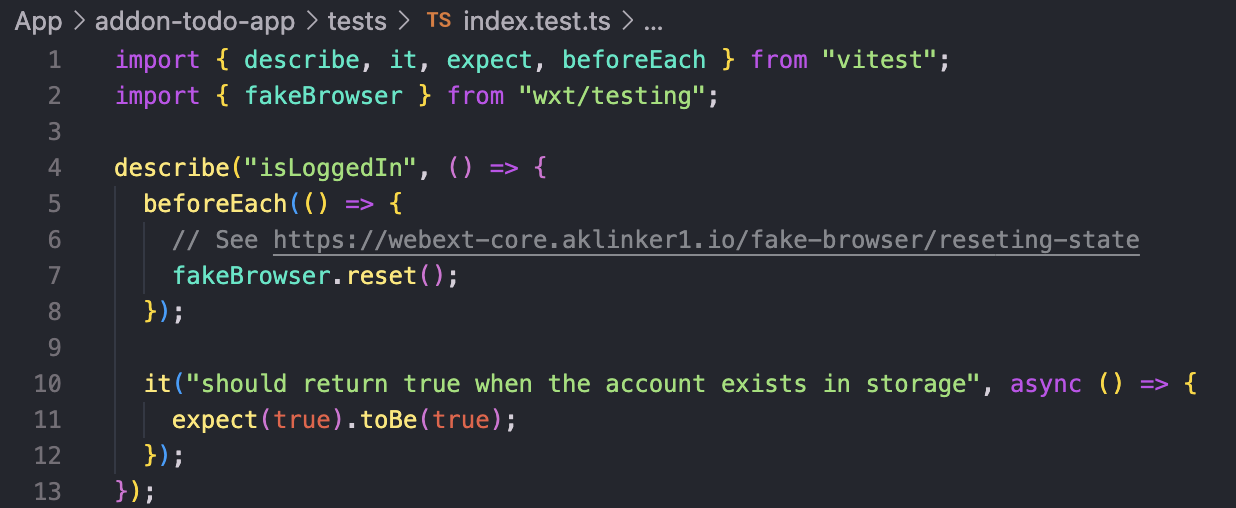
### Outils Utilisés

1. OneDrive  
 - Rôle : Stockage des données persistantes.  
 - Utilisation :  
 - Création d'un dossier nommé `addon\_todo` et ajout d’un fichier `taskList.json` pour la gestion des tâches.  
  
2. npm (Node Package Manager)  
 - Rôle : Gestionnaire de paquets JavaScript.  
 - Utilisation :  
 - Installation des dépendances nécessaires au projet (« npm install »).  
 - Commandes spécifiques pour le développement et la génération :  
 - « npm run dev:firefox » : Lance l'application en mode développement pour Firefox.  
 - « npm run zip:firefox » : Génère un fichier ZIP prêt pour le déploiement de l'addon.  
  
3. Firefox Developer Tools  
 - Rôle : Déboguer et tester les extensions dans Firefox.  
 - Utilisation :  
 - Accéder au débogueur via la commande `about:debugging` dans la barre de recherche de Firefox.  
 - Tester et visualiser les modifications en direct.

## 3. Journal des Commits

## 4. Développement et Tests

Mise en place de l’environnement de test :

1. Installer vitest avec « npm install --save-dev vitest ts-node. »
2. Aller modifier le package.json pour créer une commande dans les scripts pour lancer les tests : "test": "vitest"
3. Créer le fichier vitest.config.ts pour configurer l’outil. [source](https://wxt.dev/guide/essentials/unit-testing)
4. Créer un simple test unitaire :

Ce test est configuré pour vérifier la fonction isLoggedIn dans un environnement de navigateur simulé. Le test actuel vérifie simplement que true est égal à true.

**Automatisation :**

1. Il faut maintenant automatiser le test avec GitHub action. J’ai créé un workflow « **node.js.yml** » qui vérifie le code source, configure Node.js, met en cache les dépendances npm, installe les dépendances, construit le projet, exécute les tests, crée un fichier zip pour Firefox, et télécharge ce fichier zip comme un artifact.
2. J’ai rencontré des difficultés à mettre un chemin correct pour la sauvegarde du fichier .zip
3. Maintenant que le workflow est en place, il faut créer des vrais tests unitaires :

- Processus de développement :  
 - Description des étapes  
 - Difficultés rencontrées et solutions

## 5. Tests :

- Tests :  
 - Types de tests effectués (unitaires, fonctionnels, etc.)  
 - Résultats des tests

## 6. Résultats Finaux

- Présentation des applications finies :  
 - Screenshot ou captures d’écran  
 - Fonctionnalités finales implémentées

## Deuxième Addon : Flask Login System

Simple application Python Flask qui permet de créer un compte et se login avec.

## 1. Analyse et Planification Initiale

Cette simple application permet à un utilisateur de s’enregistrer avec un pseudo, un mot de passe et une adresse email. Il va falloir donc tester toutes ces fonctionnalités.

Ma stratégie de développement :

1. Installation de l’app
2. Configuration de l’environnement de développement
3. Mise en place de tests
4. Tests
5. Automatisation des tests avec GitHub Action

### Planification

Utilisation de GitHub Project avec une planification de type Kanban.  
Les étapes sont :

- Backlog : Objet qui n'a pas été commencé.   
- Ready : Objet prêt à être commencé.  
- In Progress : Objet en cours de développement.  
- In Review : Objet en cours d'analyse.  
- Done : Objet terminé. A screenshot of a chat

Description automatically generated

Journal des commits (datés)

## 2. Configuration de l’Environnement de Développement

Application 2 : Flask Login System

**Lancer l’application :**

Assurez-vous d’avoir python3 installé sur votre machine.

1. Cloner le repo :

**git clone** [**https://github.com/jpjacobpadilla/Flask-Login-System**](https://github.com/jpjacobpadilla/Flask-Login-System)

1. Rentrer dans la racine de l’app :

**cd Flask-Login-System**

1. Installer les Dépendances nécessaires :

**pip install -r requirements.txt**

1. Naviguer dans le dossier source :

**cd src**

1. Lancer l’application :

**python app.py**

1. Une fois l’app lancée elle est accessible à l’adresse :

**http://127.0.0.1:5000**

- Processus de développement :  
 - Description des étapes  
 - Difficultés rencontrées et solutions  
- Tests e2e :  
 - Types de tests effectués (unitaires, fonctionnels, etc.)  
 - Résultats des tests

## 3. Journal des Commits

## 4. Développement et Tests

Mise en place de l’environnement de test unitaire :

1. Choisir un Framework de tests, dans mon cas j’ai choisi Pytest.
2. Créer un dossier « tests » dans le projet
3. Créer un fichier « test\_app.py ». Ce fichier contiendra les tests.
4. Installer Pytest dans le projet avec « pip install pytest »
5. En utilisant la commande « python -m pytest » dans votre terminal, les tests devraient se lancer

Mise en place de l’environnement de test E2E :

1. J’ai choisi Selenium et Webdriver manager comme frameworks pour les tests e2e. Webdriver sert à faire des tests sur la plateforme Firefox.
2. Il faut donc ajouter les frameworks dans le requierments.txt
3. Maintenant il faut les installer avec « pip install -r requirements.txt »
4. J’ai créé un fichier nommé « test\_e2e.py » où je vais mettre mes tests E2E
5. Pour lancer les tests E2E il faut utiliser la commande « python -m pytest src/tests/test\_e2e.py »

- Processus de développement :  
 - Description des étapes  
 - Difficultés rencontrées et solutions

## 5. Tests

Les tests unitaires :

|  |
| --- |
| **Test 1** : Test Login Page |
|  |
| **Description :** Vérifie que la page de connexion est accessible et contient le mot "Login". |
| **Résultat :** Code de statut : 200 (OK).  Contenu de la réponse : Le mot "Login" doit être présent dans la page. |

|  |
| --- |
| **Test 2** : Login Incorrect Username |
|  |
| **Description :** Vérifie que l'application retourne une erreur si un utilisateur inexistant tente de se connecter. |
| **Résultat :** Code de statut : 200 (page affichée sans redirection).  Contenu de la réponse : Message "Username does not exist". |

|  |
| --- |
| **Test 3** : Login Incorrect password |
|  |
| **Description :** Vérifie qu'une tentative de connexion avec un mot de passe incorrect affiche un message d'erreur. |
| **Résultat :** Code de statut : 200.  Contenu de la réponse : Message "Incorrect password". |

|  |
| --- |
| **Test 4** : Test Login Success |
|  |
| **Description : Vérifie que l'utilisateur est redirigé vers la page d'accueil après une connexion réussie.** |
| **Résultat :** Code de statut : 302 (redirection).  En-tête de la réponse : Location doit être /. |

|  |
| --- |
| **Test 5** : Test Logout |
|  |
| **Description :** Vérifie que la déconnexion redirige l'utilisateur vers la page de connexion. |
| **Résultat :** Code de statut : 302.  En-tête de la réponse : Location doit être /login. |

|  |
| --- |
| **Test 6** : Test Register Page |
|  |
| **Description :** Vérifie que la page d'inscription est accessible et contient le mot "Register". |
| **Résultat :** Code de statut : 200.  Contenu de la réponse : Le mot "Register" doit être présent. |

|  |
| --- |
| **Test 7** : Test Register Success |
|  |
| **Description :** Vérifie qu'un utilisateur peut s'inscrire avec succès et est redirigé vers la page d'accueil. |
| **Résultat :** Code de statut : 200.  Contenu de la réponse : Message "Username already exists". |

|  |
| --- |
| **Test 8** : Test Successful Registration |
|  |
| **Description :** Vérifie qu'un utilisateur peut s'inscrire avec succès et est redirigé vers la page d'accueil. |
| **Résultat attendu :**  Code de statut : 302 (redirection).  En-tête de la réponse : Location doit être /. |

Les tests End 2 End :

|  |
| --- |
|  |
|  |
|  |
|  |

|  |
| --- |
|  |
|  |
|  |
|  |

|  |
| --- |
|  |
|  |
|  |
|  |

- Tests :  
 - Types de tests effectués (unitaires, fonctionnels, etc.)  
 - Résultats des tests

## 6. Résultats Finaux

Problèmes Rencontrés :

Dans la création des tests E2E J’ai eu des problèmes avec le register, car à chaque fois que je lançais le test, comme le register avait déjà créé un user avec un certain pseudo, quand je relançais le test pour la deuxième fois, comme l’username était déjà pris, le test ne passait pas. J’ai fait une fonction qui à chaque test créé un nouveau pseudo comme ça le problème du user already existing n’est plus. J’ai ensuite rencontré un deuxième problème avec register, comme je créais des username à chaque fois sans avec des caractères random, j’ai dû ajouter une règle pour que les pseudos ne soient que des lettres ou des chiffres, comme l’application le demande.

J’ai rencontré aussi beaucoup de problèmes sur la manière dont il faut déclarer les chemins aux dépendances comme l’accès au fichier requierments.txt

- Présentation des applications finies :  
 - Screenshot ou captures d’écran  
 - Fonctionnalités finales implémentées

## Documentation Projet :

## 5. Planification / Jobs / Étapes CI/CD

- Introduction à la CI/CD  
- Plan de CI/CD pour chaque application :  
 - Jobs configurés  
 - Étapes automatisées (build, tests, déploiement)

## 6. Conclusion - Retours d’expérience :

## J’ai trouvé difficile de se lancer dans le projet car je ne connaissais pas du tout GitHub action et j’avais de la peine à comprendre comment cela fonctionnait. Mais grâce à ce portfolio j’ai mieux compris le concept de test et GitHub action.

## - Ce qui a bien fonctionné - Améliorations possibles - Conclusion générale

## 7. Annexes

- Captures d’écran ou diagrammes supplémentaires  
Documentation complémentaire :  
GitHub Actions  
Plateforme CI/CD qui permet d’automatiser des tâches dans un dépôt GitHub, d’exécuter des workflows, etc.

Sources :

* <https://wxt.dev/guide/essentials/unit-testing>
* <https://docs.github.com/en/actions/use-cases-and-examples/building-and-testing/building-and-testing-python>

Journal de travail :

* Le journal de travail se trouve dans le repo.