# 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

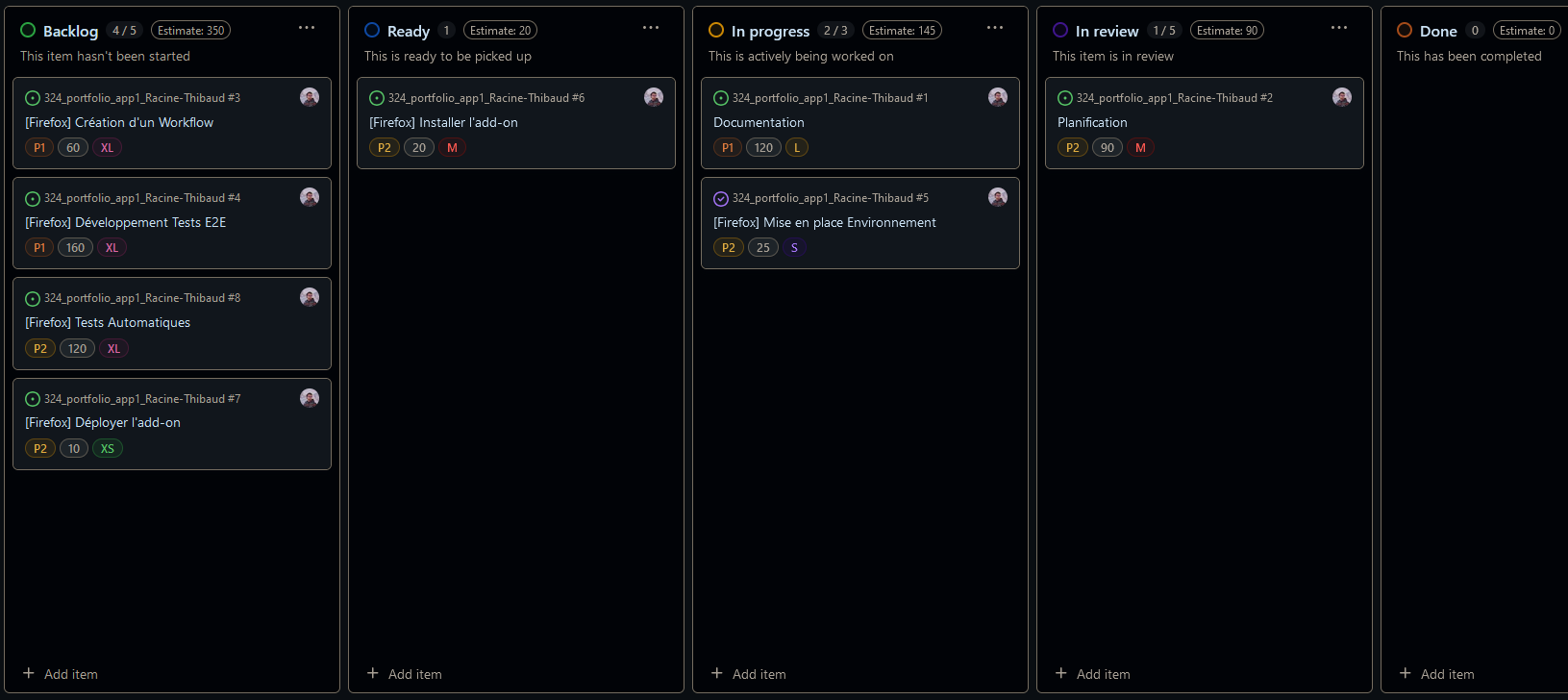
## Addon Firefox:

Addon avec une liste de TODO à tester.

## 1. Analyse et Planification Initiale

### Planification

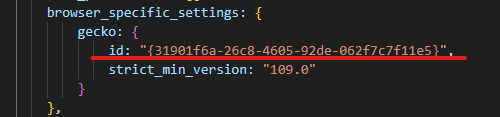
Utilisation de GitHub Project avec une planification de type Kanban.  
Les étapes sont :

- 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`.

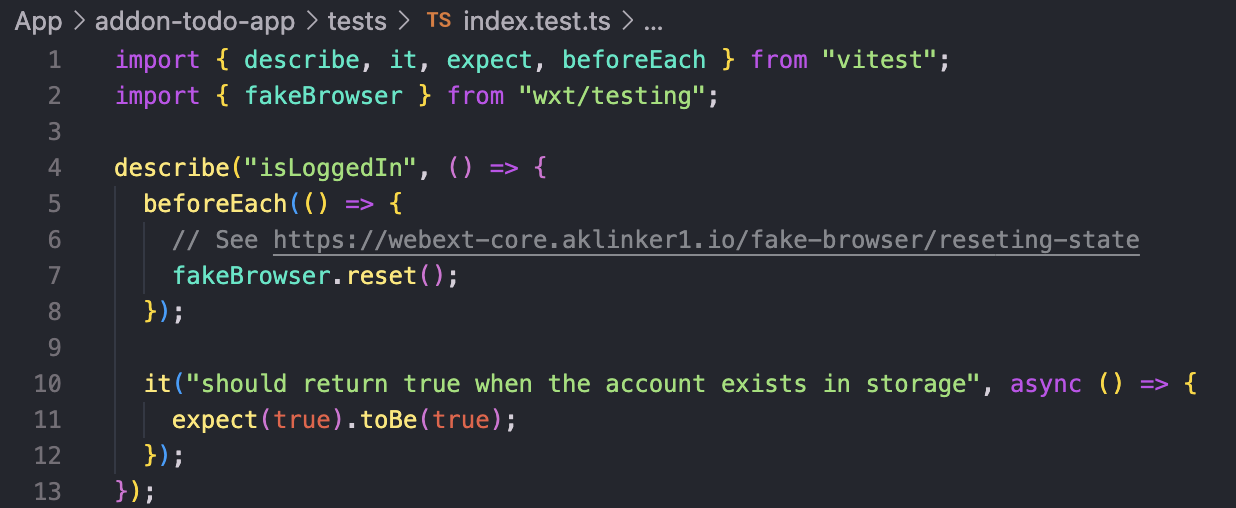
### Outils Utilisés

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'add-on.  
  
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 artefact.
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 :

|  |  |  |
| --- | --- | --- |
| Nom Test | But du Test | Résultat |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

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

## 6. Résultats Finaux

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

## Deuxième Addon : Flask Login System

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

## 1. Analyse et Planification Initiale

### 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

## 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 :

**Python3 app.py**

Une fois l’app lancée elle est accessible à l’adresse : [**http://127.0.0.1:5000**](http://127.0.0.1:5000)

1. executer les tests: python3 -m pytest tests/test\_app.py

## 3. Journal des Commits

Les commits sont visibles sur Github et le journal de travail dans le repo.

## 4. Développement et Tests

Test Unitaires :

|  |  |  |
| --- | --- | --- |
| Nom Test | But du Test | Résultat |
| test\_login\_page | Vérifier que la page de connexion s’affiche correctement. | La page retourne un statut HTTP 200 et contient le mot "Login". |
| test\_login\_incorrect\_username | Tester une connexion avec un nom d’utilisateur inexistant. | La réponse affiche "Username does not exist" et retourne un statut HTTP 200. |
| test\_login\_success | Tester une connexion avec des identifiants valides. | Redirection (statut HTTP 302) vers la page principale (/). |
| test\_logout | Vérifier que la déconnexion fonctionne correctement. | Redirection (statut HTTP 302) vers la page de connexion (/login). |
| test\_register\_page | Vérifier que la page d'inscription s’affiche correctement. | La page retourne un statut HTTP 200 et contient le mot "Register". |
| test\_register\_success | Tester une inscription avec des informations valides. | Redirection (statut HTTP 302) vers la page principale (/). |
| test\_login\_incorrect\_password | Tester une connexion avec un mot de passe incorrect. | La réponse affiche "Incorrect password" et retourne un statut HTTP 200. |

J’ai développé mes tests unitaires dans un ordre précis car chacun des tests dépens de l’autre. Car afin de faire fonctionner correctement le test logout, il faut d’abord se connecter avec un identifiant valide grâce au test

Tests E2E :

|  |  |  |
| --- | --- | --- |
| Nom Test | But du Test | Résultat |
| test\_register | Vérifier que l'inscription d'un nouvel utilisateur fonctionne correctement. | Après avoir soumis le formulaire d'inscription avec des informations valides, l'utilisateur est redirigé vers la page d'accueil (http://localhost:5000/). |
| test\_login | Vérifier que la connexion fonctionne avec un nom d'utilisateur et un mot de passe valides. | Après avoir rempli et soumis le formulaire de connexion, l'utilisateur est redirigé vers la page d'accueil (http://localhost:5000/). |
| test\_logout | Vérifier que la déconnexion fonctionne correctement. | Après avoir cliqué sur le lien "Logout", l'utilisateur est redirigé vers la page de connexion (http://localhost:5000/login). |

J’ai utilisé Selenium pour mes tests end 2 end. L’action GitHub crée un utilisateur, le connecte et ensuite le déconnecte.

- Processus de développement & étapes

- Difficultés rencontrées et solutions

J’ai eu des soucis avec les tests end 2 end car pytest va chercher l’exécutable FireFox.exe pour lancer les tests. J’ai voulu refaire les tests sur mon Mac et cela n’a pas fonctionné car MacOS ne reconnait pas les .exe J’ai donc dû implémenter dans mon code une fonction qui permets de détecter l’OS pour les tests ce qui n’a pas très bien fonctionné.

## 6. Résultats Finaux

- Présentation des applications finies :  
 - Screenshots 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>

Journal de travail :

* Le journal de travail se trouve dans le repo.