Marvel - Version 1.4.0

Alexandre GIRARD - Conseiller en Nouvelles Technologies - alexandre.girard@maif.fr

Objectifs

Explication sur les différents types de tests :

- Tests unitaires
- Tests d'intégration
- Tests end-to-end

Mettre en place des tests end-to-end pour l'application Marvel.

Pyramide des tests

La pyramide des tests, qui se compose de trois niveaux :

1. Tests unitaires : tests qui vérifient le bon fonctionnement d'une unité de code (fonction, classe, etc.).
2. Tests d'intégration : tests qui vérifient que les différentes unités de code fonctionnent correctement ensemble.
3. Tests end-to-end : tests qui vérifient le bon fonctionnement de l'application dans son ensemble, en simulant le comportement de l'utilisateur.

Pyramide des tests - Recommandations

Il est recommandé de privilégier les tests unitaires aux tests d'intégration et aux tests end-to-end, car ils sont plus rapides à exécuter et plus simples à mettre en place.

Le terme de **pyramide des tests** fait référence à la forme de la pyramide, qui est plus large à la base (tests unitaires) et plus étroite au sommet (tests end-to-end).

Pyramide des tests - Points de vigilance

Cependant il est important de ne pas négliger les tests d'intégration et les tests end-to-end, car ils permettent de détecter des problèmes qui ne pourraient pas être détectés par des tests unitaires.

Mais il est également important de ne pas sur-investir dans les tests d'intégration et les tests end-to-end, car ils sont plus complexes et plus coûteux à mettre en place que les tests unitaires. Plus difficile à maintenir et à faire évoluer.

Tests d'intégration

Les tests d'intégration permettent de vérifier que les différents composants d'un système fonctionnent correctement ensemble (base de données, API, interface utilisateur, etc.). Contrairement aux tests unitaires, les tests d'intégration ne se concentrent pas sur une seule unité de code, mais sur l'ensemble du système.

Dans le cadre de l'application Marvel, les tests d'intégration permettraient de vérifier que l'application communique correctement avec l'API Marvel et que les données sont correctement restituées. Ayant choisi de simplifier l'application en "simulant" l'API Marvel, les tests d'intégration ne sont pas pertinents dans ce contexte.

Tests end-to-end

Les tests end-to-end permettent de vérifier le bon fonctionnement de l'application dans son ensemble, en simulant le comportement de l'utilisateur. Ils permettent de tester l'application dans des conditions proches de la réalité, en simulant les interactions de l'utilisateur avec l'application.

On test ici des scénarios complets, de l'ouverture de l'application à la consultation, modification et suppression de données.

Les frameworks permettant aussi généralement de tester l'application sur différents navigateurs, ce qui permet de vérifier la compatibilité de l'application avec les différents navigateurs.

Tests end-to-end - Mise en place avec Playwright

Pour mettre en place des tests end-to-end pour l'application Marvel, nous allons utiliser le framework Playwright.

Playwright est un framework de test moderne et performant, qui permet de tester des applications web et mobiles de manière fiable et efficace. Il est compatible avec les principaux navigateurs (Chrome, Firefox, Safari, etc.) et permet de tester des applications web et mobiles sur différentes plateformes (Windows, macOS, Linux, etc.).

Tests end-to-end - Mise en place avec Playwright

Playwright permet de simuler les interactions de l'utilisateur avec l'application, en cliquant sur des éléments, en remplissant des formulaires, en naviguant entre les pages, etc. Il permet également de vérifier que l'application affiche les bonnes données, en vérifiant le contenu des éléments de la page.

Alexandre GIRARD - Conseiller en Nouvelles Technologies - alexandre.girard@maif.fr

Tests end-to-end - Mise en place avec Playwright

Créer une nouvelle branche `feature/e2e-tests` et installer Playwright :

```
npm init playwright@latest
```

Renseigner les informations demandées (JavaScript, e2e-tests pour le nom du répertoire de tests, yes pour la github action et installation des browsers).

Renommer les fichiers `playwright.config.js` en `playwright.config.cjs` et `example.spec.js` en `example.spec.cjs`.

Tests end-to-end - Execution des tests

Pour exécuter les tests end-to-end, il suffit de lancer la commande suivante :

```
npx playwright test
```

Playwright va alors lancer les tests dans les différents navigateurs configurés (Chrome, Firefox, Safari, etc.) et afficher les résultats dans la console.

Il est possible de lancer une interface graphique pour visualiser les tests en cours d'exécution :

```
npx playwright test --ui
```

Créer un nouveau fichier `marvel.spec.js` dans le répertoire `e2e-tests` :

```
import { test, expect } from "@playwright/test";

test("navigation in marvel-app is correct", async ({ page }) => {
  await page.goto("http://localhost:5173");

  // Expect a title "to contain" a substring.
  await expect(page).toHaveTitle(/Marvel App/);

  // click on Beast element and expect to navigate to Beast page
  await page.click("text=Beast");
  await expect(page).toHaveTitle(/ Beast | Marvel App/);

  // click on Home element and expect to navigate to Home page
  await page.click("text=Home");
  await expect(page).toHaveTitle(/Marvel App/);

  // click on About element and expect to navigate to About page
  await page.click("text=About");
  await expect(page).toHaveTitle(/About | Marvel App/);

  // click on Contact element and expect to navigate to Contact page
  await page.click("text=Contact");
  await expect(page).toHaveTitle(/Contact | Marvel App/);
});
```

Tests end-to-end - Execution des tests

Relancer les tests end-to-end en mode graphique pour visualiser les tests en cours d'exécution :

```
npx playwright test --ui
```

Explications

Le test ouvre l'application Marvel, vérifie que le titre de la page contient "Marvel App", puis clique sur les différents éléments de la navigation (Home, Beast, About, Contact) et vérifie que la page correspondante s'affiche.

Cela permet de vérifier que la navigation de l'application fonctionne correctement et que les différentes pages s'affichent correctement.

Pour aller plus loin

Modifier le fichier de configuration `playwright.config.cjs` pour ajouter un nouveau navigateur simulant un mobile :

```
    /* Test against mobile viewports. */
    {
      name: 'Mobile Chrome',
      use: {
        ...devices['Desktop Chrome'],
        // It is important to define the `viewport` property after destructuring `devices`,
        // since devices also define the `viewport` for that device.
        viewport: { width: 599, height: 900 },
       },
    },
```

Créer un test end-to-end pour vérifier que le footer est bien caché sur les petits écrans :

```
import { test, expect } from "@playwright/test";

test("test footer display", async ({ page }) => {
    await page.goto("/");

    // get the size of the viewport
    const viewport = page.viewportSize();

    if (viewport.width < 600) {
        // expect footer to be hidden
        await expect(page.locator("footer")).not.toBeVisible();
    } else {
        // expect footer to be visible
        await expect(page.locator("footer")).toBeVisible();
    }
})
```

Relancer les tests end-to-end pour vérifier que l'application est bien responsive et que les pages s'affichent correctement sur mobile. Nous avons ici mis en oeuvre un test pour vérifier que le footer est bien caché sur les petits écrans.

Le nouveau test est ko car nous n'avons pas encore adapté le footer pour les petits écrans.

Il est possible de modifier le comportement de l'application en fonction de la taille de l'écran, en utilisant les media queries CSS.

Par exemple, on peut cacher le footer de l'application sur les petits écrans de moins de 600px de large :

```
/* hide footer on small screens */
@media (max-width: 600px) {
  footer {
    display: none;
  }
}
```

Il est possible de vérifier le comportement de l'application grâce aux outils de développement des navigateurs, en simulant un mobile ou une tablette ou en modifiant la taille de la fenêtre du navigateur.

Intégration continue

Il est possible d'intégrer les tests end-to-end dans un pipeline d'intégration continue, pour automatiser les tests et s'assurer que l'application fonctionne correctement à chaque modification du code.

Playwright a généré un fichier de configuration pour GitHub Actions, qui permet de lancer les tests end-to-end à chaque push sur la branche `main`. Les tests end-to-end étant plus longs à exécuter que les tests unitaires, ils ne sont généralement pas exécutés sur chaque branche.

Intégration continue (suite)

Pour nos tests, nous allons commenter la partie `on` du fichier de configuration pour lancer les tests quelque soit la branche, mais sur un "vrai" projet, il est recommandé de ne lancer les tests end-to-end que sur la branche `main` ou sur une branche spécifique dédiée aux tests end-to-end.

```
name: Playwright Tests
on:
  push:
  #   branches: [ main, master ]
  # pull_request:
  #   branches: [ main, master ]
```

Intégration continue (suite)

Décommenter et modifier la partie `webserver` du fichier de configuration pour lancer le serveur web local avant les tests :

```
  /* Run your local dev server before starting the tests */
  webServer: {
    command: 'npm run dev',
    url: 'http://localhost:5173',
    reuseExistingServer: !process.env.CI,
  },
```

Celà permet de lancer le serveur web local avant les tests, pour s'assurer que l'application est bien accessible avant de lancer les tests.

Publier les modifications sur la branche `feature/e2e-tests` et créer une pull request pour l'intégrer dans la branche `develop`.

Conclusion

Les tests end-to-end permettent de vérifier le bon fonctionnement de l'application dans son ensemble, en simulant le comportement de l'utilisateur. Ils permettent de tester des scénarios complets et de vérifier que l'application fonctionne correctement dans des conditions proches de la réalité.

Attention à ne pas sur-investir dans les tests end-to-end, car ils sont plus complexes et plus coûteux à mettre en place que les tests unitaires. Il est recommandé de privilégier les tests unitaires aux tests d'intégration et aux tests end-to-end.

Il n'est pas rare de voir des tests end-to-end qui ne sont pas maintenus et qui deviennent obsolètes, car ils sont plus difficiles à maintenir et à faire évoluer que les tests unitaires.

Conclusion (suite)

Attention à ne pas mettre en oeuvre une pyramide des tests inversée, avec plus de tests end-to-end que de tests unitaires. Il est important de trouver le bon équilibre entre les différents types de tests, en fonction des besoins et contraintes du projet.

Les tests end-to-end sont un outil précieux pour s'assurer que l'application fonctionne correctement dans des conditions proches de la réalité, mais il est important de ne pas négliger les tests unitaires et les tests d'intégration, qui permettent de détecter des problèmes plus rapidement et plus facilement.

Release

Faire le nécessaire pour créer une version 1.4.0 de l'application.