Projet 5 :
Tests full stack

CHRISTOPHE PIERRÈS,
LE MERCREDI 28 MAI 2025, DURÉE : 15MN

JORDI CHARPENTIER

INTRODUCTION

- Avant propos

- Questions d’abord pour les tests front puis back :
  - Test réalisés ?
  - Comment ont-ils été implémentés et pourquoi ?
  - Comment ont-ils été choisis ?
  - Contraintes ou difficultés rencontrées, résolution ?
  - Ce que j'aurais fait différemment ?

TESTS FRONTEND

- Tests Jest + TestBed pour fixture Angular
  - Rapides
  - Tests unitaires
  - Tests d’intégration
  - Séparés par des blocs `describe` spécifiques

TESTS TU ET TI FRONTEND RÉALISÉS (EXEMPLES)

- Composants :
  - login.component.spec.ts : logique et validation du formulaire de connexion
  - app.component.spec.ts : logique principale de l'application et rendu initial
  - form.component.spec.ts, list.component.spec.ts : formulaire, vue liste
  - me.component.spec.ts : comportement des infos de connexion de l'utilisateur
- Services et Intercepteurs :
  - auth.service.spec.ts : vérifie Register, authentification et appels HTTP sécurisés.
  - jwt.interceptor.spec.ts : valide le bon comportement de l’intercepteur JWT.
  - session.service.spec.ts : teste la session utilisateur (logIn, LogOut, Info session)
  - session-api.service.spec.ts : fonctionnalités métier pour sessions de yoga
- Gardes et navigation :
  - auth.guard.spec.ts, unauth.guard.spec.ts : Vérifient la logique de protection des routes pour les utilisateurs connectés ou non

TU ET TI : COMMENT ONT-ILS ÉTÉ IMPLÉMENTÉS ET POURQUOI ?

- Tests orientés comportement (BDD) :
  - Utilisation de blocs `describe` et `it` pour structurer les tests et expliquer les scénarios
  - Séparation des TU et des Tests d'intégration
- Mise en oeuvre avec stubs/mocks/spy :
  - Les dépendances critiques (comme des services HTTP) sont imitées pour éviter les appels externes (`jest.fn()`)
- Simulations d'événements utilisateur :
  - Par exemple, la saisie dans un formulaire ou un clic sur un bouton peut être mocké dans les composants.

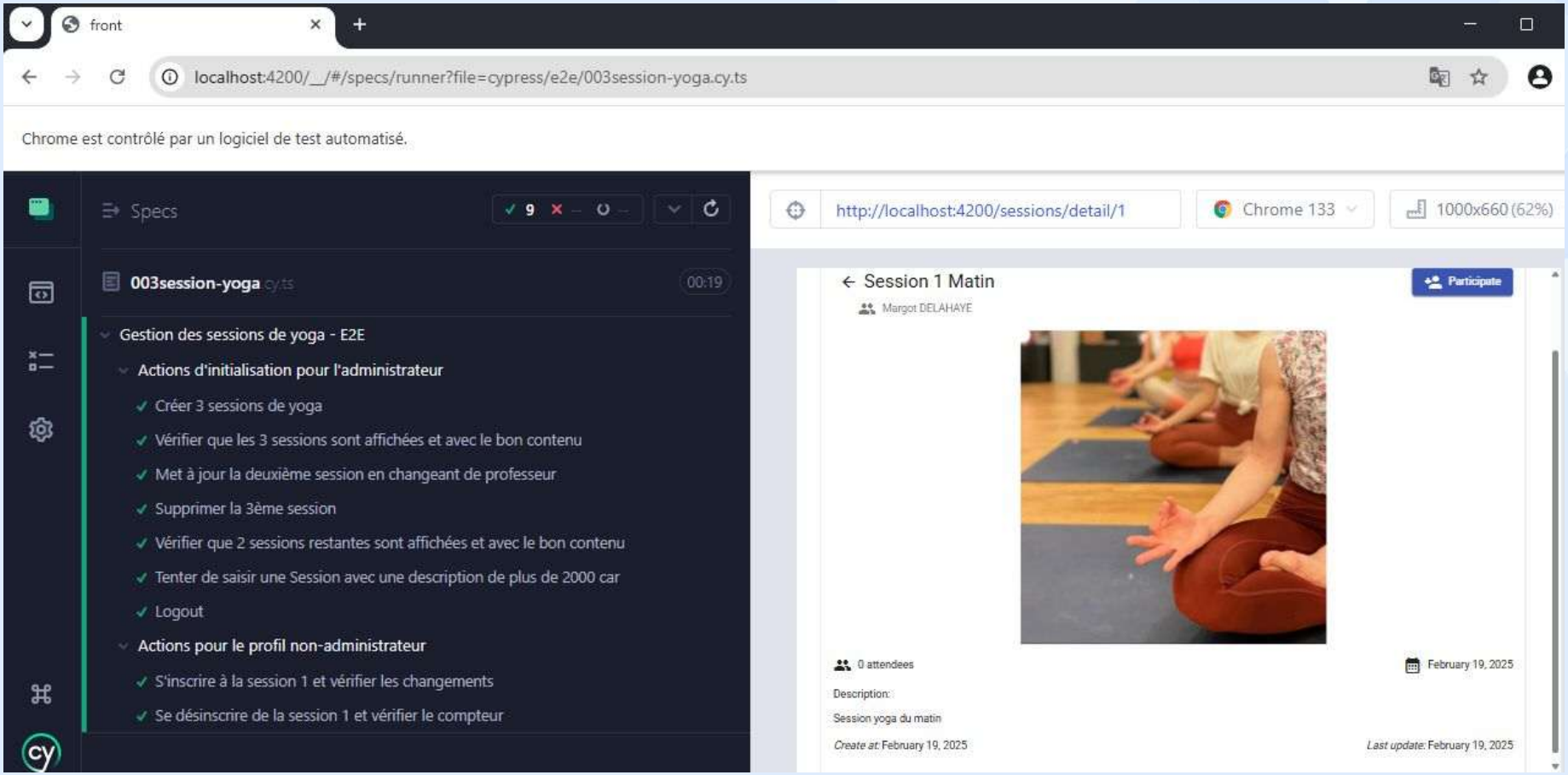
TESTS UNITAIRES - COMMENT ONT-ILS ÉTÉ IMPLÉMENTÉS ET POURQUOI ?

- commencer par :
  - tests simples comme la création du composant,
  - puis validation des méthodes et comportements sans trop dépendre de services ou d'éléments complexes
- workflow d'implémentation (partie TestBed)
  - Configuration d'un module avec TestBed.configureTestingModule
  - Injection des dépendances nécessaires
  - Création du composant/service à tester grâce à TestBed.createComponent ou TestBed.inject
  - Test des comportements ou des outputs en mockant les dépendances si nécessaire

TESTS D’INTÉGRATION - COMMENT ONT-ILS ÉTÉ IMPLÉMENTÉS ET POURQUOI ?

- Utilité de TestBed en complément de Jest
- Exemple d’un test (depuis login.component.spec.ts) :
  - Vérifier l'intégration entre le formulaire de connexion et le SessionService associé
- Utilisation de Ajv pour valider structures (dans session.service.spec.ts)

TESTS FRONTEND- TESTS E2E AVEC CYPRESS



- 2 modes (variable d'environnement=>Cypress.env('useRealBackend')) :
  - En simulation (mocks et fixture de données)
  - Optionnellement avec le vrai backend

COMMENT ONT-ILS ÉTÉ CHOISIS ?

- Priorisés selon la criticité et la couverture des fonctionnalités essentielles :
  - Composants interactifs
  - Authentification et sécurisation :
  - Points sensibles : La structure globale (app.component) et les erreurs communes (comme les pages 404) sont couvertes.

FRONTEND : CONTRAINTES OU DIFFICULTÉS RENCONTRÉES, RÉSOLUTION

- Configurer les tests pour communiquer avec dépendances tierces (API ou services)
  - Utilisation de HttpTestingController dans Angular ou des mocks full Jest
  - Se défaire des réflexes « synchrones » :
    - httpMock.expectOne('api/auth/register') doit être après le service qui appelle l'API
- Tester des composants avec directives complexes (comme FormBuilder ou ngIf)
  - Modules de test Angular configurés avec les dépendances exactes pour le composant (TestBed.configureTestingModule)

TESTS BACKEND

- Tests rapides (phase ‘test’ dans maven) :
  - Tests unitaires
  - Tests d’intégration mockés

- Tests d’intégration système (SIT) plus lents avec base H2/dialecte mysql (phase ‘validate’ de maven) :
  - Correction du jacoco pour lui faire prendre en compte les tests d’intégration lors de la phase validate de maven
  - Configuration du plugin fail-safe

COMMENT ONT-ILS IMPLÉMENTÉS ?

- Chaque test est totalement autonome
  - Crée ses données utiles également (ne se repose pas sur un jeu de données pré-supposé dans la base H2 par exemple)
    - Création d'une classe utilitaire PersistentTestDataCreator

DIFFICULTÉS RENCONTRÉES SUR TESTS BACKEND

- Petites difficultés ...
  - Voir éventuellement ensemble dans le code (mot clé « soutenance »)
  - Pb de comparaison de date (json = UTC vs Fuseau horaire pour java)

CE QUE J'AURAIS FAIT DIFFÉREMMENT

- Pour tests backend
  - Utilisation de TestContainers plutôt que H2
  - Pour des écrans avec une logique métier complexe, j'aurais sans doute prévu des jeux de tests spécifiques dans ces TestContainers
- Tests à automatiser dans Intégration Continue

- Merci !