Mise en place de tests d'intégration sur une app React

Performances, apprentissages et limites

Par
Joris Langlois

Date
25/03/2025

Conférence

Mise en place de tests d'intégration sur une app React _ Sommaire

Qui suis-je ?

Développeur.

- En activité depuis **2013**
- Chez KNP Labs depuis **2018**
- Petit passage par les pays baltes entre **2020 et 2022** (outl1ne)

Auditeur.

- Ecole d'ingénieur du Conservatoire National des Arts et Métiers (EiCNAM) depuis **2017**
- Ingéniérie des systèmes decisionnels
- Certificat de spécialisation : Union Européenne

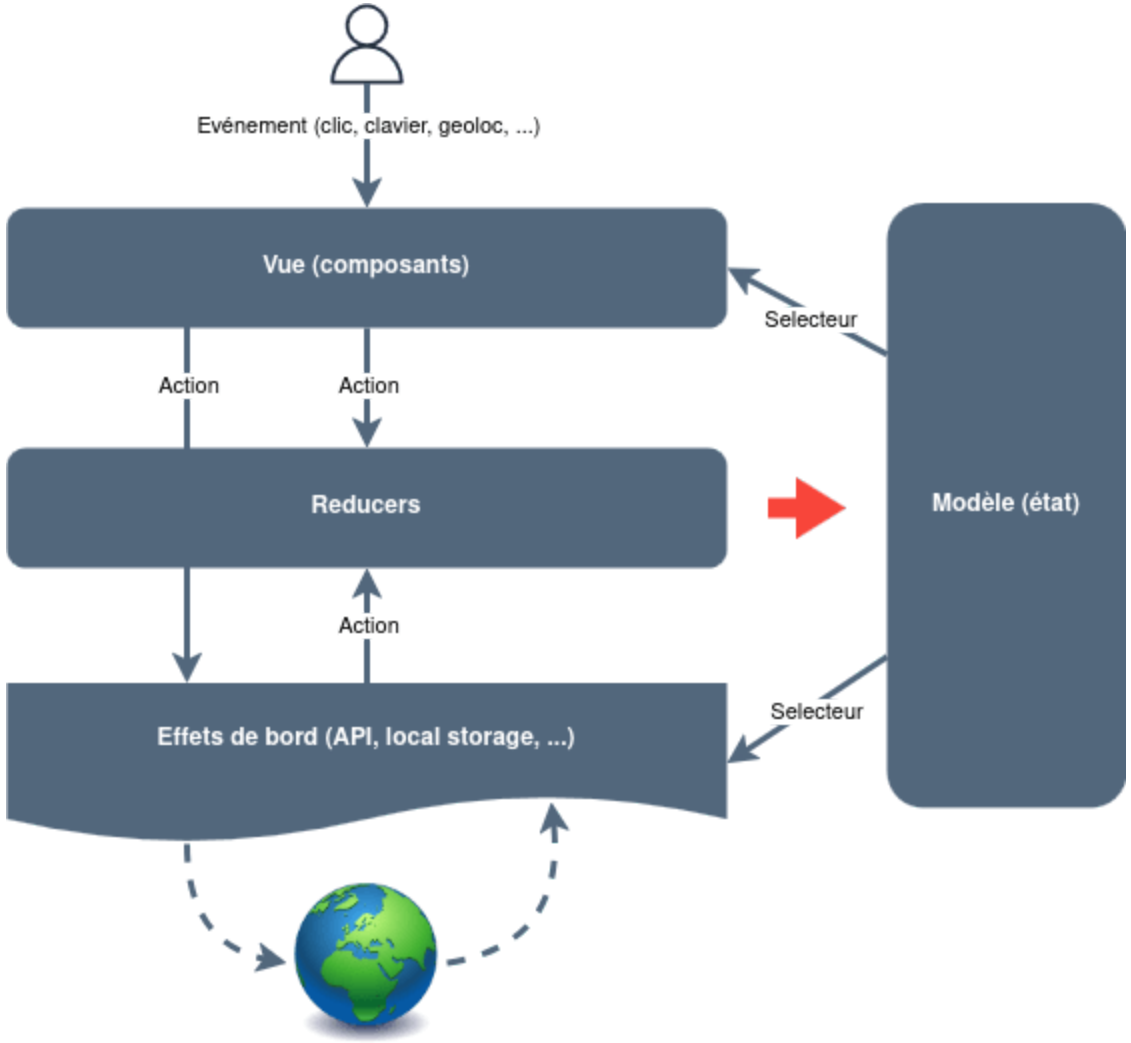
Introduction

Tout commença...

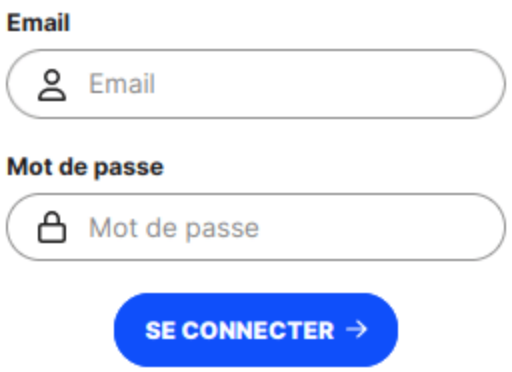
... par un nouveau projet

- **un peu trop de certitudes**
- **et un vague sentiment de malaise**

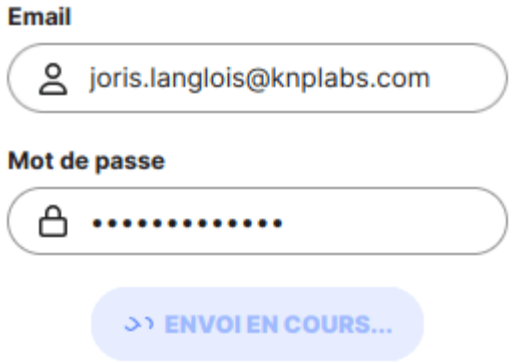
Introduction _ Une architecture réactive, qu'est ce que c'est ?



Introduction _ Cas d'étude



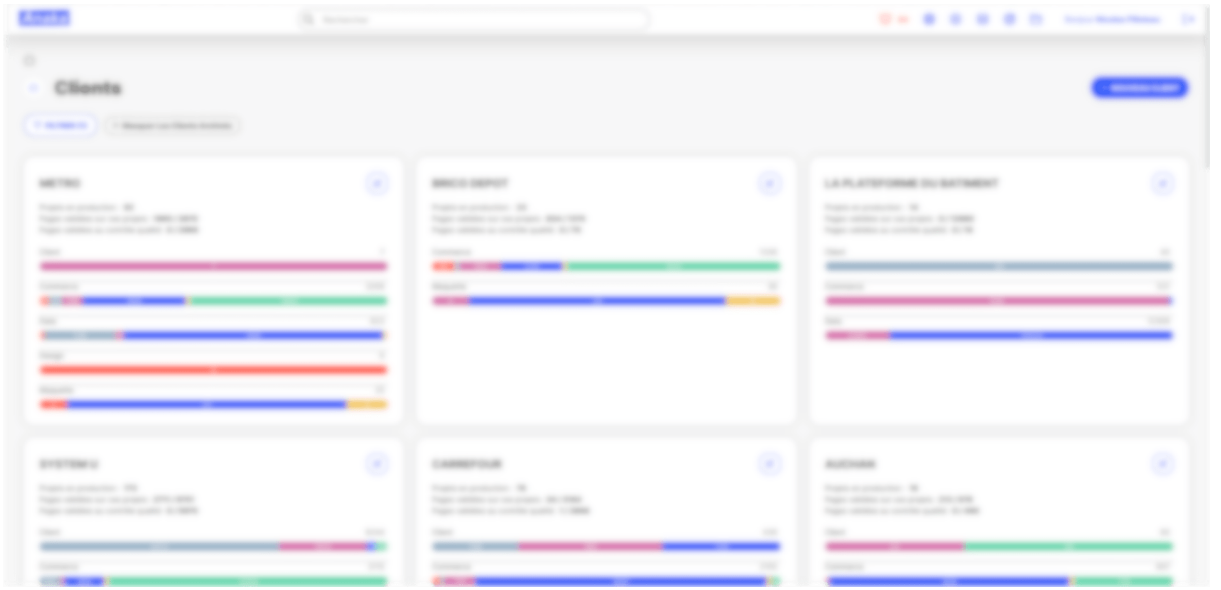
- **Cas 1 : Non authentifié**



- **Cas 2 : En cours d'authentification**

Introduction _ Cas d'étude

- **Cas 3 : Erreur**



- **Cas 4 : Authentification OK**

Introduction _ Cas d'étude : vue

```tsx
// features/Authentication/index.tsx

const Firewall: React.FC<{ children: React.ReactNode[] }> = ({ children }) => {
  const dispatch = useDispatch()
  const isAuthenticating = useSelector(selectIsAuthenticating)
  const isAuthenticated = useSelector(selectIsAuthenticated)
  const hasError = useSelector(selectHasError)

  if (isAuthenticated) return children

  const onSubmit: React.FormEventHandler = e => {
    e.preventDefault()

    dispatch(slice.actions.authenticateWithCredentials({
      username: e.currentTarget.username?.value || '',
      password: e.currentTarget.password?.value || '',
    }))
  }

  return <Form onSubmit={ onSubmit }>
    {/* ... */}
    <SubmitButton loading={ isAuthenticating }>
        Se connecter
        <Icon name="arrow-right" />
    </SubmitButton>

    { hasError && <Message type={ MessageType.ERROR } content="Une erreur s'est produite..." /> }
  </Form>
}
```

Introduction _ Cas d'étude : vue

```tsx
// features/Authentication/index.tsx

const Firewall: React.FC<{ children: React.ReactNode[] }> = ({ children }) => {
  const dispatch = useDispatch()
  const isAuthenticating = useSelector(selectIsAuthenticating)
  const isAuthenticated = useSelector(selectIsAuthenticated)
  const hasError = useSelector(selectHasError)

  if (isAuthenticated) return children

  const onSubmit: React.FormEventHandler = e => {
    e.preventDefault()

    dispatch(slice.actions.authenticateWithCredentials({
      username: e.currentTarget.username?.value || '',
      password: e.currentTarget.password?.value || '',
    }))
  }

  return <Form onSubmit={ onSubmit }>
    {/* ... */}
    <SubmitButton loading={ isAuthenticating }>
        Se connecter
        <Icon name="arrow-right" />
    </SubmitButton>

    { hasError && <Message type={ MessageType.ERROR } content="Une erreur s'est produite..." /> }
  </Form>
}
```

- **Couplage fort avec le modèle => selecteurs**

Introduction _ Cas d'étude : vue

```tsx
// features/Authentication/index.tsx

const Firewall: React.FC<{ children: React.ReactNode[] }> = ({ children }) => {
  const dispatch = useDispatch()
  const isAuthenticating = useSelector(selectIsAuthenticating)
  const isAuthenticated = useSelector(selectIsAuthenticated)
  const hasError = useSelector(selectHasError)

  if (isAuthenticated) return children

  const onSubmit: React.FormEventHandler = e => {
    e.preventDefault()

    dispatch(slice.actions.authenticateWithCredentials({
      username: e.currentTarget.username?.value || '',
      password: e.currentTarget.password?.value || '',
    }))
  }

  return <Form onSubmit={ onSubmit }>
    {/* ... */}
    <SubmitButton loading={ isAuthenticating }>
        Se connecter
        <Icon name="arrow-right" />
    </SubmitButton>

    { hasError && <Message type={ MessageType.ERROR } content="Une erreur s'est produite..." /> }
  </Form>
}
```

- **Couplage fort avec le modèle => selecteurs**
- **Couplage fort avec le modèle => actions**

Introduction _ Cas d'étude : modèle

```ts
// features/Authentication/slice.ts

export enum Status {
  NeedsAuthentication = 'NeedsAuthentication',
  Authenticating = 'Authenticating',
  Authenticated = 'Authenticated',
  UnknownError = 'UnknownError',
}

type State = {
  status: Status
}

export const initialState: State = ({
  status: Status.NeedsAuthentication,
})

export type AuthenticatePayload = {
  username: string,
  password: string,
}

export const selectIsAuthenticating = (state: State) =>
    state.auth.status === Status.Authenticating

export const selectIsAuthenticated = (state: State) =>
    state.auth.status === Status.Authenticated
```

```ts
// features/Authentication/slice.ts

const slice = createSlice({
  name: 'auth',
  initialState,
  reducers: {
    authenticateWithCredentials: (
      state,
      _action: PayloadAction<AuthenticatePayload>,
    ) => {
      state.status = AuthStatus.Authenticating,
    },

    successfullyRetrievedToken: state => state,

    authenticated: (
      state,
      _action: PayloadAction<User>,
    ) => {
      state.status: AuthStatus.Authenticated,
    },

    error: state => {
      state.status: AuthStatus.UnknownError,
    },
  },
})
```

Introduction _ Cas d'étude : modèle

```ts
// features/Authentication/slice.ts

export enum Status {
  NeedsAuthentication = 'NeedsAuthentication',
  Authenticating = 'Authenticating',
  Authenticated = 'Authenticated',
  UnknownError = 'UnknownError',
}

type State = {
  status: Status
}

export const initialState: State = ({
  status: Status.NeedsAuthentication,
})

export type AuthenticatePayload = {
  username: string,
  password: string,
}

export const selectIsAuthenticating = (state: State) =>
    state.auth.status === Status.Authenticating

export const selectIsAuthenticated = (state: State) =>
    state.auth.status === Status.Authenticated
```

```ts
// features/Authentication/slice.ts

const slice = createSlice({
  name: 'auth',
  initialState,
  reducers: {
    authenticateWithCredentials: (
      state,
      _action: PayloadAction<AuthenticatePayload>,
    ) => {
      state.status = AuthStatus.Authenticating,
    },

    successfullyRetrievedToken: state => state,

    authenticated: (
      state,
      _action: PayloadAction<User>,
    ) => {
      state.status: AuthStatus.Authenticated,
    },

    error: state => {
      state.status: AuthStatus.UnknownError,
    },
  },
})
```

Introduction _ Cas d'étude : modèle

```ts
// features/Authentication/slice.ts

export enum Status {
  NeedsAuthentication = 'NeedsAuthentication',
  Authenticating = 'Authenticating',
  Authenticated = 'Authenticated',
  UnknownError = 'UnknownError',
}

type State = {
  status: Status
}

export const initialState: State = ({
  status: Status.NeedsAuthentication,
})

export type AuthenticatePayload = {
  username: string,
  password: string,
}

export const selectIsAuthenticating = (state: State) =>
    state.auth.status === Status.Authenticating

export const selectIsAuthenticated = (state: State) =>
    state.auth.status === Status.Authenticated
```

```ts
// features/Authentication/slice.ts

const slice = createSlice({
  name: 'auth',
  initialState,
  reducers: {
    authenticateWithCredentials: (
      state,
      _action: PayloadAction<AuthenticatePayload>,
    ) => {
      state.status = AuthStatus.Authenticating,
    },

    successfullyRetrievedToken: state => state,

    authenticated: (
      state,
      _action: PayloadAction<User>,
    ) => {
      state.status: AuthStatus.Authenticated,
    },

    error: state => {
      state.status: AuthStatus.UnknownError,
    },
  },
})
```

Introduction _ Cas d'étude : effets de bord

```ts
// features/Authentication/effects.ts

export default function* rootSaga(): Generator {
  yield takeLeading(slice.actions.authenticateWithCredentials, authenticateWithCredentials)
  yield takeEvery(slice.actions.successfullyRetrievedToken, getAuthenticatedUser)
}

function* authenticateWithCredentials({ payload }: { payload: AuthenticatePayload }): Generator {
  try {
    const post = yield getContext(Context.Post)
    const container = (yield call(post, '/auth-tokens', payload)) as LdapToken
    const storage = yield getContext(Context.Storage)
    yield call([ storage, 'setItem' ], 'token', container.token)
    yield put(slice.actions.successfullyRetrievedToken())
  } catch (e) {
    yield put(slice.actions.error())
  }
}

function* getAuthenticatedUser(): Generator {
  try {
    const get = yield getContext(Context.Get)
    const user = (yield call(get, '/me')) as User
    yield put(slice.actions.authenticated(user))
  } catch (e) {
    yield put(slice.actions.error())
  }
}
```

Introduction _ Cas d'étude : effets de bord

1. Action in

```ts
// features/Authentication/effects.ts

export default function* rootSaga(): Generator {
  yield takeLeading(slice.actions.authenticateWithCredentials, authenticateWithCredentials)
  yield takeEvery(slice.actions.successfullyRetrievedToken, getAuthenticatedUser)
}

function* authenticateWithCredentials({ payload }: { payload: AuthenticatePayload }): Generator {
  try {
    const post = yield getContext(Context.Post)
    const container = (yield call(post, '/auth-tokens', payload)) as LdapToken
    const storage = yield getContext(Context.Storage)
    yield call([ storage, 'setItem' ], 'token', container.token)
    yield put(slice.actions.successfullyRetrievedToken())
  } catch (e) {
    yield put(slice.actions.error())
  }
}

function* getAuthenticatedUser(): Generator {
  try {
    const get = yield getContext(Context.Get)
    const user = (yield call(get, '/me')) as User
    yield put(slice.actions.authenticated(user))
  } catch (e) {
    yield put(slice.actions.error())
  }
}
```

Introduction _ Cas d'étude : effets de bord

1. Action in

```ts
// features/Authentication/effects.ts

export default function* rootSaga(): Generator {
  yield takeLeading(slice.actions.authenticateWithCredentials, authenticateWithCredentials)
  yield takeEvery(slice.actions.successfullyRetrievedToken, getAuthenticatedUser)
}

function* authenticateWithCredentials({ payload }: { payload: AuthenticatePayload }): Generator {
  try {
    const post = yield getContext(Context.Post)
    const container = (yield call(post, '/auth-tokens', payload)) as LdapToken
    const storage = yield getContext(Context.Storage)
    yield call([ storage, 'setItem' ], 'token', container.token)
    yield put(slice.actions.successfullyRetrievedToken())
  } catch (e) {
    yield put(slice.actions.error())
  }
}

function* getAuthenticatedUser(): Generator {
  try {
    const get = yield getContext(Context.Get)
    const user = (yield call(get, '/me')) as User
    yield put(slice.actions.authenticated(user))
  } catch (e) {
    yield put(slice.actions.error())
  }
}
```

2. Effet

Introduction _ Cas d'étude : effets de bord

1. Action in

```ts
// features/Authentication/effects.ts

export default function* rootSaga(): Generator {
  yield takeLeading(slice.actions.authenticateWithCredentials, authenticateWithCredentials)
  yield takeEvery(slice.actions.successfullyRetrievedToken, getAuthenticatedUser)
}

function* authenticateWithCredentials({ payload }: { payload: AuthenticatePayload }): Generator {
  try {
    const post = yield getContext(Context.Post)
    const container = (yield call(post, '/auth-tokens', payload)) as LdapToken
    const storage = yield getContext(Context.Storage)
    yield call([ storage, 'setItem' ], 'token', container.token)
    yield put(slice.actions.successfullyRetrievedToken())
  } catch (e) {
    yield put(slice.actions.error())
  }
}

function* getAuthenticatedUser(): Generator {
  try {
    const get = yield getContext(Context.Get)
    const user = (yield call(get, '/me')) as User
    yield put(slice.actions.authenticated(user))
  } catch (e) {
    yield put(slice.actions.error())
  }
}
```

2. Effet

3. Action out !

Introduction _ Cas d'étude : effets de bord

4. Action in

```
1  // features/Authentication/effects.ts
2
3  export default function* rootSaga(): Generator {
4    yield takeLeading(slice.actions.authenticateWithCredentials, authenticateWithCredentials)
5    yield takeEvery(slice.actions.successfullyRetrievedToken, getAuthenticatedUser)
6  }
7
8  function* authenticateWithCredentials({ payload }: { payload: AuthenticatePayload }): Generator {
9    try {
10     const post = yield getContext(Context.Post)
11     const container = (yield call(post, '/auth-tokens', payload)) as LdapToken
12     const storage = yield getContext(Context.Storage)
13     yield call([ storage, 'setItem' ], 'token', container.token)
14     yield put(slice.actions.successfullyRetrievedToken())
15   } catch (e) {
16     yield put(slice.actions.error())
17   }
18 }
19
20 function* getAuthenticatedUser(): Generator {
21   try {
22     const get = yield getContext(Context.Get)
23     const user = (yield call(get, '/me')) as User
24     yield put(slice.actions.authenticated(user))
25   } catch (e) {
26     yield put(slice.actions.error())
27   }
28 }
```

Introduction _ Cas d'étude : effets de bord

4. Action in

```
1  // features/Authentication/effects.ts
2
3  export default function* rootSaga(): Generator {
4    yield takeLeading(slice.actions.authenticateWithCredentials, authenticateWithCredentials)
5    yield takeEvery(slice.actions.successfullyRetrievedToken, getAuthenticatedUser)
6  }
7
8  function* authenticateWithCredentials({ payload }: { payload: AuthenticatePayload }): Generator {
9    try {
10     const post = yield getContext(Context.Post)
11     const container = (yield call(post, '/auth-tokens', payload)) as LdapToken
12     const storage = yield getContext(Context.Storage)
13     yield call([ storage, 'setItem' ], 'token', container.token)
14     yield put(slice.actions.successfullyRetrievedToken())
15   } catch (e) {
16     yield put(slice.actions.error())
17   }
18 }
19
20 function* getAuthenticatedUser(): Generator {
21   try {
22     const get = yield getContext(Context.Get)
23     const user = (yield call(get, '/me')) as User
24     yield put(slice.actions.authenticated(user))
25   } catch (e) {
26     yield put(slice.actions.error())
27   }
28 }
```

5. Effet

Introduction _ Cas d'étude : effets de bord

4. Action in

```ts
// features/Authentication/effects.ts

export default function* rootSaga(): Generator {
  yield takeLeading(slice.actions.authenticateWithCredentials, authenticateWithCredentials)
  yield takeEvery(slice.actions.successfullyRetrievedToken, getAuthenticatedUser)
}

function* authenticateWithCredentials({ payload }: { payload: AuthenticatePayload }): Generator {
  try {
    const post = yield getContext(Context.Post)
    const container = (yield call(post, '/auth-tokens', payload)) as LdapToken
    const storage = yield getContext(Context.Storage)
    yield call([ storage, 'setItem' ], 'token', container.token)
    yield put(slice.actions.successfullyRetrievedToken())
  } catch (e) {
    yield put(slice.actions.error())
  }
}

function* getAuthenticatedUser(): Generator {
  try {
    const get = yield getContext(Context.Get)
    const user = (yield call(get, '/me')) as User
    yield put(slice.actions.authenticated(user))
  } catch (e) {
    yield put(slice.actions.error())
  }
}
```

5. Effet

6. Action out !

Introduction _ Cas d'étude : modèle

```ts
// features/Authentication/slice.ts

export enum Status {
  NeedsAuthentication = 'NeedsAuthentication',
  Authenticating = 'Authenticating',
  Authenticated = 'Authenticated',
  UnknownError = 'UnknownError',
}

type State = {
  status: Status
}

export const initialState: State = ({
  status: Status.NeedsAuthentication,
})

export type AuthenticatePayload = {
  username: string,
  password: string,
}

export const selectIsAuthenticating = (state: State) =>
    state.auth.status === Status.Authenticating

export const selectIsAuthenticated = (state: State) =>
    state.auth.status === Status.Authenticated
```

```ts
// features/Authentication/slice.ts

const slice = createSlice({
  name: 'auth',
  initialState,
  reducers: {
    authenticateWithCredentials: (
      state,
      _action: PayloadAction<AuthenticatePayload>,
    ) => {
      state.status = AuthStatus.Authenticating,
    },

    successfullyRetrievedToken: state => state,

    authenticated: (
      state,
      _action: PayloadAction<User>,
    ) => {
      state.status: AuthStatus.Authenticated,
    },

    error: state => {
      state.status: AuthStatus.UnknownError,
    },
  },
})
```

Problématique

Problématique _ Fonctions

- **Qu'il s'agisse de la vue :**
  - $Firewall : Props \longrightarrow JSX.Element$
- **Du modèle :**
  - $authenticateWithCredentials : (State, Action) \longrightarrow State$
- **Ou des effets de bord :**
  - $getAuthenticatedUser* : Action \longrightarrow Action$
- **Des fonctions, des fonctions, toujours des fonctions !**

Problématique _ Fonctions

- **Quelques propriétés élémentaires que l'on peut exploiter à notre avantage :**
  - Pureté
  - Composabilité

- **Dont découle logiquement une approche fonctionnelle des tests :**
  - La robustesse du système est garantie par la composition de fonctions pures unitairement testées

- **Et nos tests unitaires, dans tout ça ?**
  - Modèle : application directe et triviale 
  - Effets de bord : possible, à un certain coût 
  - Vues : de quels composants parle-t-on ? 

Problématique _ Limites de l'approche unitaire

- **Tests de reducers (modèle)** ✅
  - Incontournables
- **Tests de selecteurs (modèle)** ✅
  - Certains sélecteurs ne sont que des accesseurs
  - Chronophages, verbeux et répétitifs
- **Tests d'effets de bord (appels d'API, accès au presse papier, au local storage, ...)** 
  - Nécessitent de mocker le comportement des dépendances injectées
  - 90% des effets de bord ne gèrent "que" des appels d'API (succès / erreur)
  - Verbeux et répétitifs
  - Peu lisibles / maintenables (marble testing)

Problématique _ Limites de l'approche unitaire

- **Tests de composants d'interface** ✅
  - **Réutilisables** par définition
  - Gèrent leur propre état interne
  - Peu couplés aux autres couches
  - Composables

- **Tests de composants montés par une route (features)** ❌
  - Spécifiques, peu réutilisables et composables
  - Fortement couplés aux autres couches
  - Impliquent un certain degré de simulation pour pouvoir mettre ces composants dans l'état souhaité pour pouvoir les tester

Problématique _ Limites de l'approche unitaire : exemple avec React Testing Library

```tsx
// features/Authentication/Firewall.test.tsx

describe('features/Me/Authentication', () => {
  test('authenticates', async () => {
    const { userEvent } = setup(<Firewall>app</Firewall>)
    const submit = screen.getByRole('button', { name: 'Se connecter' })
    expect(submit).toBeEnabled()

    await userEvent.type(screen.getByLabelText('Email'), 'joris.langlois@knplabs.com')
    await userEvent.type(screen.getByLabelText('Mot de passe'), 'notanychance')

    await userEvent.click(submit)
    expect(submit).toBeDisabled()

    act(() => {
      store.dispatch(slice.actions.authenticated({
        firstname: 'Joris',
        lastname: 'Langlois',
        company: 'KNP Labs',
        // ...
      }))
    })
    expect(await screen.findByText(/app/)).toBeInTheDocument()
  })
})
```

Problématique _ Limites de l'approche unitaire : exemple avec React Testing Library

```tsx
// features/Authentication/Firewall.test.tsx

describe('features/Me/Authentication', () => {
  test('authenticates', async () => {
    const { userEvent } = setup(<Firewall>app</Firewall>)
    const submit = screen.getByRole('button', { name: 'Se connecter' })
    expect(submit).toBeEnabled()

    await userEvent.type(screen.getByLabelText('Email'), 'joris.langlois@knplabs.com')
    await userEvent.type(screen.getByLabelText('Mot de passe'), 'notanychance')

    await userEvent.click(submit)
    expect(submit).toBeDisabled()

    act(() => {
      store.dispatch(slice.actions.authenticated({
        firstname: 'Joris',
        lastname: 'Langlois',
        company: 'KNP Labs',
        // ...
      }))
    })
    expect(await screen.findByText(/app/)).toBeInTheDocument()
  })
})
```

Problématique _ Bilan

- **Risques de faux positifs**
  - Si on supprime / modifie **l'effet** responsable de diffuser les actions (authenticated ou error) dans l'application, les tests passent toujours, mais l'application est KO

- **Risques de faux négatifs**
  - Si on arrête de diffuser ces actions **dans les tests**, ils sont KO alors que l'application va très bien...

- **Un bilan mi-figue mi-raisin**
  - Les composants fonctionnent tous en isolation mais...
  - Les conditions d'exécution sont éloignées de la réalité
  - On teste l'implémentation plutôt que le comportement
  - Tests chronophages, répétitifs et verbeux
  - Pour un gain de confiance faible

Nouvelle approche

Nouvelle approche _ Quelle stratégie ?

*It's all about getting a good return on your investment where "return" is **confidence** and "investment" is **time**.*

Kent C. Dodds - 2021

Nouvelle approche _ Quelle stratégie ?

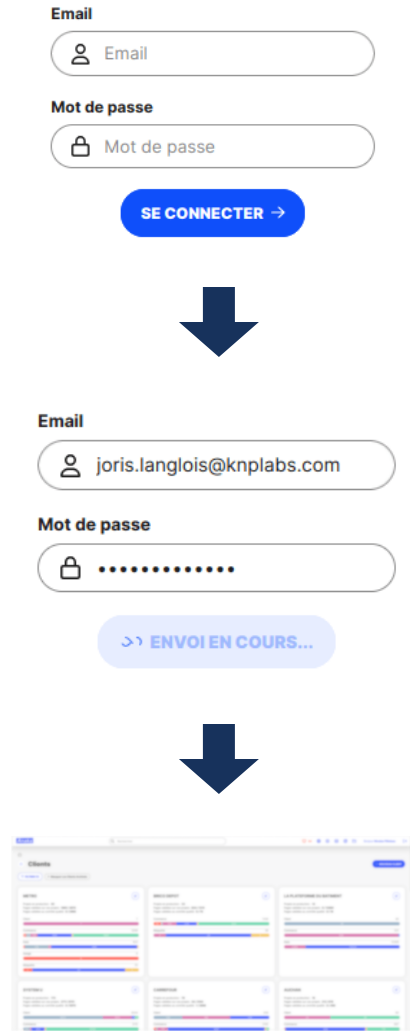
- **Kent C. Dodds. Write tests. Not too many. Mostly integration. 2019.**
  - https://kentcdodds.com/blog/write-tests

- **Principe**
  - Plutôt que de tout tester de la même manière et de façon peu efficace, on déplace l'effort de test pour **le centrer sur le comportement de l'utilisateur**

Nouvelle approche _ Exemple avec MSWjs

Scénario nominal

```
1  // features/Authentication/Firewall.test.tsx
2
3  describe('features/Me/Authentication', () => {
4    test('authenticates', async () => {
5      const { userEvent } = setup(<Firewall>app</Firewall>)
6      const submit = screen.getByRole('button', { name: 'Se connecter' })
7      expect(submit).toBeEnabled()
8
9      await userEvent.type(screen.getByLabelText('Email'), 'joris.langlois@knplabs.com')
10     await userEvent.type(screen.getByLabelText('Mot de passe'), 'notanychance')
11     await userEvent.click(submit)
12
13     expect(submit).toBeDisabled()
14     expect(await screen.findByText(/app/)).toBeInTheDocument()
15   })
16 })
```



Nouvelle approche _ Exemple avec MSWjs

- **MSW (Mock Service Worker)**, par Artem Zakharchenko et 150+ contributeurs
  - Interception bas niveau des appels réseaux
  - Agnostique du client
  - Tous les composants, de l'envoi de la requête à sa résolution, sont exécutés

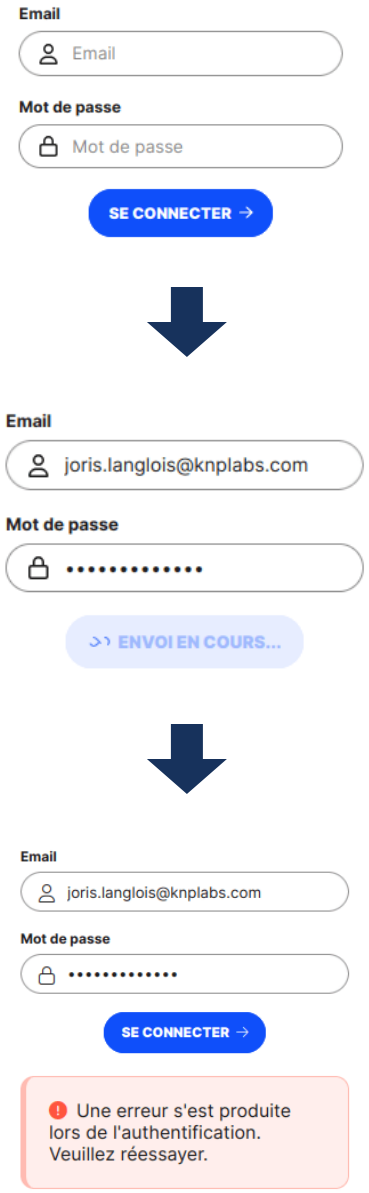
```ts
// setupTests.ts

import { setupServer } from 'msw/node'
import { handlers } from 'test/mocks/handlers'

const server = setupServer(...handlers)

beforeAll(() => {
  server.listen()
})

afterEach(() => server.resetHandlers())

afterAll(() => server.close())
```

```ts
// test/mocks/handlers.ts

import { HttpResponse, http } from 'msw'
import { users } from 'test/fixtures'

const ok = (body:any = null) => new HttpResponse(
  JSON.stringify(body),
  {
    status: 200,
    headers: { 'Content-Type': ContentType.Json },
  }
)

const handlers = [
  http.post('/auth-tokens', () => ok({ token: 'my-token' })),
  http.get('/me', () => ok(users[0])),
]
```

Nouvelle approche _ Exemple avec MSWjs

Scénario d'échec

```
1  // features/Authentication/Firewall.test.tsx
2
3  describe('features/Me/Authentication', () => {
4    test('cannot authenticate', async () => {
5      const { userEvent } = setup(<Firewall />)
6      const submit = screen.getByRole('button', { name: 'Se connecter' })
7
8      server.use(
9        http.get('/me', async () => HttpResponse.error())
10     )
11
12     await userEvent.type(screen.getByLabelText('Email'), 'joris.langlois@knplabs.com')
13     await userEvent.type(screen.getByLabelText('Mot de passe'), 'notanychance')
14
15     await userEvent.click(submit)
16     await waitFor(() => expect(submit).toBeDisabled())
17     await waitFor(() => expect(submit).toBeEnabled())
18
19     expect(screen.getByText(/Une erreur s'est produite/)).toBeInTheDocument()
20   })
21 })
```



Bénéfices et limites

- **Les sélecteurs de type accesseurs sont testés indirectement**
  - Plus besoin de les tester unitairement
- **La plupart des effets de bord sont aussi testés indirectement à partir du moment où l'interface change en réaction à une action système**
  - Plus besoin de les tester unitairement ni même de les exporter (= simplification)
- **Plus besoin de diffuser des actions "à la main" dans les tests de composant pour simuler un comportement**
  - Tests plus lisibles et plus simples à maintenir
- **Baisse du risque de faux positifs / négatifs**
  - Davantage de confiance dans ce qu'on livre
  - Diminution du risque de régression

Bénéfices et limites _ Limites

- **Plus lents que des tests unitaires**
  - (100 suites / 500 tests ~ 50s)
- **Prise en main : tout est asynchrone !**
  - Résolution des réponses de MSW potentiellement instantanées
  - Besoin d'attendre que les éléments de l'interface soient tous là pour les assertions et passer au test suivant
- **Risque de tests un peu instables lors de certains enchaînements**

Bénéfices et limites _ Réflexions complémentaires

Si les jeux de test (fixtures) sont :

- testés unitairement en utilisant les JSON schémas du backend...
- ...tout comme les données que l'on envoie à l'API

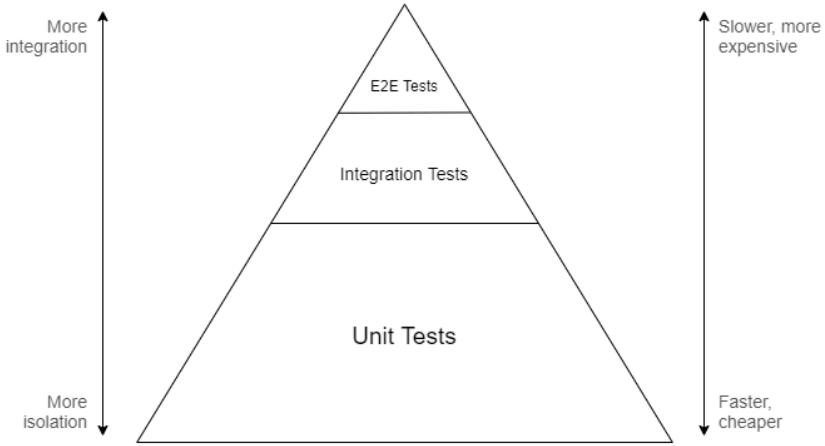
Les tests E2E sont-ils utiles ?

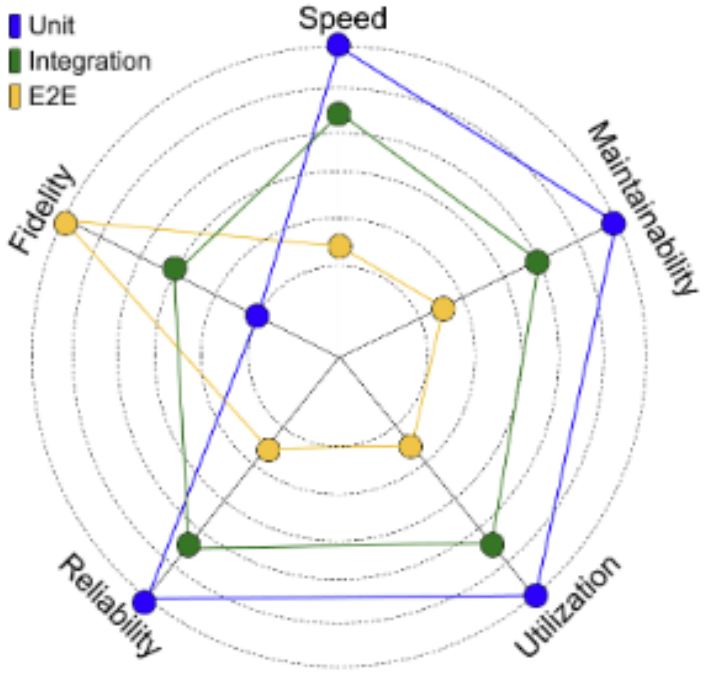
```ts
// apps/frontend/test/fixtures/index.ts

export const users = [
  {
    id: 1,
    firstname: 'John',
    lastname: 'Doe',
    profilePicturePath: 'cat.png',
    locale: 'fr',
    role: 'admin',
  },
]
```

```json
// apps/backend/doc/schemas/user.json

{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "id": "http://cacom.fr/user.json",
    "type": "object",
    "required": [ "id", "firstname", "lastname", "locale" ],
    "additionalProperties": false,
    "properties": {
        "id": {
            "type": "integer"
        },
        "firstname": {
            "type": "string"
        },
        "lastname": {
            "type": "string"
        },
        "profilePicturePath": {
            "type": "string"
        },
        "locale": {
            "type": "string"
        },
        "role": {
            "$ref": "defs/user-role.json"
        }
    }
}
```

Bénéfices et limites _ Réflexions complémentaires

- **Quel coverage ?**
- **Adam Bender. SMURF: Beyond the Test Pyramid. 2024.**
  - https://testing.googleblog.com/2024/10/smurf-beyond-test-pyramid.html





(c) Google 2024

Merci !