

Présentation du projet 724-Events

Bienvenue à cette présentation du projet 724-Events
Exercice de débogage d'une agence d'évènementiel.

 par thomas sifferle



The screenshot shows a website header with a purple gradient background. On the left, there's a navigation bar with '724 events' in white. To the right are links for 'Nos services', 'Nos réalisations', 'Notre équipe', and 'Contact'. Below the header is a large image of a conference stage at the World Economic Forum. Several people are seated in white armchairs, facing an audience. A blue banner across the image reads 'World economic forum' and 'Oeuvre à la coopération entre le secteur public et le privé.' At the bottom of the page, a white box contains the heading 'Nos services' and the subtext 'Nous organisons des événements sur mesure partout dans le monde.'

724 events

Contexte et contraintes techniques

- **Client :** Agence événementielle parisienne, 724events
- **Mission :** Débugger et finaliser le développement d'un site vitrine one-page
- **Contexte :**
 - Site présentant plusieurs bugs signalés par un ancien développeur.
 - Je suis développeur front-end freelance
 - Contacté par l'agence 724-events
 - Un premier développeur freelance a commencé l'intégration
- **Objectifs :**
 - Identifier et corriger les bugs du site React
 - Tester et valider le fonctionnement du site
 - Compléter un cahier de recette avec des scénarios de tests
 - Utiliser des outils comme Chrome DevTools, React Developer Tools, Yarn et Node.js
 - Gérer le projet via GitHub et Visual Studio Code
- **Interlocuteur :**
 - Jean-Baptiste, le directeur Marketing de l'agence

Dépot git et fichier readme.md

README

724 Events

Description

L'application est le site d'une agence evenementielle.

Pre-requis

- NodeJS >= v16.14.1

Installation

- `yarn install`

Lancement de l'application

- `yarn start`

Tests

- `yarn test`

OpenClassRooms_Projet-9_Debug-React-App Public

main 1 Branch 0 Tags

Go to file Add file Code

TomSif fix (Bug : Events + refacto : Slider): correction bug Events , il n'... 902b219 · 1 hour ago 13 Commits

public fix (footer): fix escaping ' with ' last week

src fix (Bug : Events + refacto : Slider): correction bug Events , il ... 1 hour ago

.eslintrc.js Create the application 3 years ago

.gitignore Initialize project using Create React App 3 years ago

README.md Update README.md 3 years ago

package-lock.json first commit: application se lance last month

package.json fix (Bug : Events + refacto : Slider): correction bug Events , il ... 1 hour ago

report.json fix (Bug : Events + refacto : Slider): correction bug Events , il ... 1 hour ago

yarn.lock first commit: application se lance last month

About

Exercice de débogage d'un application React.

Readme Activity 0 stars 0 watching 0 forks

Releases

No releases published [Create a new release](#)

Packages

No packages published [Publish your first package](#)



Le projet 724-Events consistait à déboguer et finaliser le développement d'un site vitrine one-page pour une agence événementielle parisienne. L'objectif principal était d'identifier et de corriger les bugs signalés par un ancien développeur, de tester et valider le fonctionnement du site, et de compléter un cahier de recette avec des scénarios de tests.

Pour ce faire j'ai mis l'accent sur ces trois axes que je vais développer en détail.

1 Liste des bugs et leurs corrections

- Présentation des bugs identifiés au départ
- Analyse des bugs
- Description des solutions apportées pour chaque bug
- Illustration des résultats avant et après correction

2 Cahier de recette

- Description du rôle du cahier de recette dans la validation du site
- Présentation de la structure des scénarios de test écrits
- Explication de la vérification des fonctionnalités clés via ces scénarios
- Exemples concrets de tests réalisés manuellement

3 Tests unitaires facultatifs

- Explication de l'intérêt des tests unitaires dans un projet React
- Présentation des tests ajoutés pour améliorer la qualité du code
- Illustration de la détection facilitée des régressions futures
- Précision des outils utilisés pour écrire et lancer les tests (ex : Jest)



Liste des Tests unitaires échoués

Voici les 5 tests qui échouaient initialement, confirmant les bugs présents dans l'application :

- **Fail** : *When Events is created > and we select a category > an filtered list is displayed*
 - **Emplacement** : .../src/containers/Events/index.test.js
 - **But du test** : Ce test simule un clic sur un filtre de catégorie et vérifie que les événements n'appartenant pas à cette catégorie disparaissent bien de la liste.
 - **Raison de l'échec** : Le filtre par catégorie ne fonctionnait pas correctement. Un événement qui aurait dû être masqué restait visible à l'écran.
- **Fail** : *When a event card is created > a title, a label and a month are displayed*
 - **Emplacement** : .../src/components/EventCard/index.test.js
 - **Raison de l'échec** : Le mois de l'événement n'était pas affiché correctement. Le test s'attendait à voir "avril" mais ne le trouvait pas, probablement à cause d'un problème dans la fonction d'affichage de la date.
- **Fail** : *When Events is created > and a click is triggered on the submit button > the success action is called*
 - **Emplacement** : .../src/containers/Form/index.test.js
 - **Raison de l'échec** : La fonction de succès (onSuccess) n'était pas appelée après la soumission du formulaire, indiquant un problème dans la logique de gestion du clic.
- **Fail** : *When slider is created > a list card is displayed*
 - **Emplacement** : .../src/containers/Slider/index.test.js
 - **Raison de l'échec** : Le slider n'affichait pas correctement les événements dans le bon ordre. Le test s'attendait à trouver l'événement de "janvier" en premier, mais il était absent, suggérant un problème de tri ou d'affichage de la première slide.
- **Fail** : *When Form is created > and a click is triggered on the submit button > the success message is displayed*
 - **Emplacement** : .../src/pages/Home/index.test.js
 - **Raison de l'échec** : Le message de confirmation "Message envoyé !" n'apparaissait pas après avoir cliqué sur "Envoyer", ce qui indique que le feedback utilisateur après soumission était cassé.

1.- Liste rapide résumée des corrections aux bugs

Bug 1 - Filtrage cassé

✗ Problème

```
(!type ? data?.events : data?.events) // Même chose partout !
```

✓ Solution

```
const allEvents = data?.events || [];
allEvents.filter(event => ltype || event.type === type)
// Si aucun type sélectionné, garde tous les événements
```

Impact : Les filtres par catégorie fonctionnent enfin

Bug 2 - Dates incorrectes

✗ Problème

```
import { getMonth } from "..." // Import faux
MONTHS[date.getMonth()] // Retourne 0-11, source fréquente d'erreurs
```

✓ Solution

```
import getMonth from "..." // Import correct
date.toLocaleDateString('fr-FR', {month: 'long'}) // API native
```

Impact : "avril" s'affiche au lieu d'undefined

Bug 3 - Formulaire muet

✗ Problème

```
try {
  await mockContactApi();
  setSending(false);
  // onSuccess() manquant !
}
```

✓ Solution

```
try {
  await mockContactApi();
  setSending(false);
  onSuccess(); // ← Ajout crucial
}
```

Impact : "Message envoyé !" s'affiche maintenant

Bug 4 - Slider buggé

✗ Problème

```
// Logique inversée complexe
index === byDateDesc.length - 1 - idx ? "display" : "hide"
// + Fuite mémoire (pas de cleanup)
```

✓ Solution

```
// Logique directe
index === idx ? "display" : "hide"
// + Modulo pour boucle infinie
setIndex(prev => (prev + 1) % length)
// + Cleanup function
```

Impact : Carrousel fluide sans slide blanche

Bug 5 - Page d'accueil instable

✗ Problème

```
// Crash si dernière prestation manquante
<EventCard date={new Date(last?.date)} /> // last peut être null
// + Import ModalEvent manquant
```

✓ Solution

```
// Gestion sécurisée
{last ? (
  <Modal><EventCard ... /></Modal>
) : (
  <p>Pas de prestation à afficher</p>
)}
// + Import ModalEvent ajouté
```

Impact : Page stable + interactivité complète

Résultats

Avant	Après
● 5/5 tests échoués	● 5/5 tests passent
● Filtres cassés	● Navigation fluide
● Dates undefined	● Affichage correct
● Pas de feedback	● UX complète

Points clés à retenir

1. **Test-driven debugging** = méthode efficace pour localiser les bugs

2. **Séparation des responsabilités** = code plus maintenable

3. **API natives** > mappings manuels = moins d'erreurs

4. **Cleanup functions** = pas de fuites mémoire

5. **Logique simple** > logique complexe = moins de bugs

2.- Cahier de Recette - Validation End-to-End

Rôle du cahier de recette

Objectif : Valider que le site fonctionne correctement du point de vue utilisateur final

Pourquoi nécessaire :

- Les **tests unitaires** vérifient le code ✓
- Le **cahier de recette** vérifie l'**expérience utilisateur** ✓
- Garantit que toutes les fonctionnalités marchent **ensemble**

Complémentarité : Code correct ≠ UX fonctionnelle

Structure des scénarios

Format BDD (Behavior Driven Development) :

Given (Contexte) → When (Action) → Then (Résultat attendu)

Exemple concret :

- **Given** : Je suis dans la section "Nos réalisations"
- **When** : Je clique sur une carte d'événement
- **Then** : Une modale s'ouvre avec les détails

Avantages : Format clair, reproductible, compréhensible par tous

Fonctionnalités clés couvertes

Navigation & Filtrage

- ✓ Filtrage par catégorie (Scénario 1)
- ✓ Pagination fonctionnelle (Scénario 4)
- ✓ Navigation par ancrés (Scénario 8)

Interactions Modales

- ✓ Ouverture détails événements (Scénarios 2, 6)
- ✓ Fermeture modales (Scénario 7)

Carrousel Automatique

- ✓ Navigation manuelle (Scénario 9)
- ✓ Défilement automatique (Scénario 11)

Formulaire Contact

- ✓ Soumission + feedback (Scénario 5)

Gestion d'erreurs

- ✓ Données manquantes (Scénario 12)

Exemples de tests manuels

Test Scénario 1 - Filtrage

1. **Action** : Clic sur "Conférence"
2. **Vérification** : Seuls les événements "Conférence" s'affichent
3. **Résultat** : ✓ PASS - Filtrage fonctionnel

Test Scénario 5 - Formulaire

1. **Action** : Remplir tous les champs + clic "Envoyer"
2. **Vérification** : Message "Message envoyé !" apparaît
3. **Résultat** : ✓ PASS - Feedback utilisateur OK

Test Scénario 11 - Carrousel Auto

1. **Action** : Attendre 5 secondes sans interaction
2. **Vérification** : Le carrousel passe à l'image suivante
3. **Résultat** : ✓ PASS - Auto-défilement OK

Bilan des tests

Fonctionnalité	Scénarios	Statut
Navigation	3 tests	✓ PASS
Modales	3 tests	✓ PASS
Formulaire	1 test	✓ PASS
Carrousel	2 tests	✓ PASS
Robustesse	1 test	✓ PASS

3.-Test Unitaires

Rôle des tests unitaires

Pourquoi ajouter des tests ?

- **Prévention des régressions** : Éviter que les futurs changements cassent l'existant
- **Documentation vivante** : Les tests décrivent le comportement attendu
- **Confiance dans le code** : Déploiement serein avec une couverture complète

Complémentarité parfaite :

- **Tests unitaires** → Vérifient le code composant par composant ✓
- **Cahier de recette** → Valide l'expérience utilisateur globale ✓

Tests implémentés

Tests d'Intégration (Page Home)

• Test d'affichage des événements

```
it("a list of events is displayed", async () => {
  render(<DataProvider><Home /></DataProvider>);
  await screen.findByRole("heading", { name: "Nos réalisations", level: 2 });
  const eventElements = await screen.findAllByText(/#productCON|MixUsers/i);
  expect(eventElements.length).toBeGreaterThan(0);
});
```

- **Ce qu'il teste** : Chargement et affichage des événements via DataProvider
- **Protection** : Garantit l'intégration API → UI et détecte les problèmes de rendu
- **Test d'affichage de l'équipe**

```
it("a list of people is displayed", () => {
  render(<DataProvider><Home /></DataProvider>);
  expect(screen.getByText("Samira")).toBeInTheDocument();
  expect(screen.getByText("Luis")).toBeInTheDocument();
});
```

- **Ce qu'il teste** : Présence des membres d'équipe dans la section dédiée
- **Protection** : Évite la suppression accidentelle de données critiques
- **Test du footer**

```
it("a footer is displayed", () => {
  render(<DataProvider><Home /></DataProvider>);
  expect(screen.getByText("Notre dernière prestation")).toBeInTheDocument();
  expect(screen.getByText("contact@724events.com")).toBeInTheDocument();
});
```

- **Ce qu'il teste** : Intégrité des éléments essentiels du footer
- **Protection** : Assure que les informations de contact restent accessibles

Tests Unitaires (Composants)

• Tests d'intégrité des icônes

```
describe("When a icon is created with name facebook", () => {
  it("the icon contain this path hash value bbea4c9e40773b969fdb6e406059f853", () => {
    render(<Icon name="facebook" />)
    expect(md5(screen.getByTestId("icon").getAttribute('d'))).toEqual('bbea4c9e40773b969fdb6e406059f853')
  });
});
```

- **Ce qu'ils testent** : Hash MD5 des chemins SVG (Facebook, Twitter, YouTube)
- **Protection** : Détection immédiate de corruption/modification des assets visuels
- **Test de robustesse avec données manquantes**

```
it("an event card, with the last event, is displayed in the footer", async () => {
  const latestEvent = [...mockData.events]
  .sort((a, b) => new Date(b.date) - new Date(a.date))[0];
  const footerElement = await screen.findByRole("contentinfo");
  const lastEventTitle = await within(footerElement)
    .findByText(new RegExp(latestEvent.title, "i"));
  expect(lastEventTitle).toBeInTheDocument();
});
```

- **Ce qu'il teste** : Affichage correct de la dernière prestation dans le footer
- **Protection** : Évite les crashes du Bug #5 (gestion données null/undefined)

Syntaxe Jest & Concepts Clés

Structure de base

```
describe("When [context]", () => { // Suite de tests
  beforeEach(() => { // Setup avant chaque test
    api.loadData = jest.fn().mockResolvedValue(mockData);
  });
});
```

```
it("should [behavior]", async () => { // Test individuel
  // Arrange (préparation)
  render(<Component />);

  // Act (action)
  fireEvent.click(screen.getByText("Click me"));

  // Assert (vérification)
  expect(result).toBe(expected);
});
```

Concepts fondamentaux

- **describe()** : Groupe logique de tests liés à un contexte
- **it()** : Test unitaire individuel décrivant un comportement attendu
- **beforeEach()** : Fonction exécutée avant chaque test (setup)
- **render()** : Monte le composant React dans un DOM virtuel
- **screen** : Interface pour interagir avec les éléments rendus
- **expect()** : Assertion qui valide le résultat obtenu

Sélecteurs React Testing Library

```
// Par rôle sémantique (recommandé)
screen.getByRole("button", { name: "Submit" })
screen.findByRole("contentinfo") // <footer> élément
```

```
// Par texte visible
screen.getByText("Contact")
screen.findAllByText(/#productCON|MixUsers/i) // RegExp
```

```
// Par attribut de test
screen.getByTestId("icon")
```

Patterns asynchrones

```
// find* = attendre l'apparition (async)
await screen.findByText("Message envoyé !");
```

```
// get* = élément déjà présent (sync)
screen.getByText("Submit");
```

```
// within() = recherche dans un conteneur spécifique
const footer = screen.getByRole("contentinfo");
within(footer).findByText("Contact");
```

Type de test	Nombre	Statut
Tests d'Intégration	3 tests	✓ PASS
Tests Unitaires	4 tests	✓ PASS

Total : 7 tests ajoutés

Valeur ajoutée

- **Détection précoce** des bugs lors des modifications
- **Documentation automatique** du comportement attendu
- **Déploiement confiant** grâce à la validation automatisée
- **Maintenance facilitée** avec des tests de non-régression

Débrief Projet 724-Events

❓ Problèmes Rencontrés

- Logique complexe mal documentée
- Pause forcée du projet pendant un mois suite au décès de ma mère
- Reprise compliquée, j'ai du m'aider de mes commits et faire un git clone, checkout pour me rappeler des tests qui échouaient
- React toujours pas maitrisé complètement, encore besoin de travailler en profondeur sur d'autres projets

✅ Points Forts Développés

- Refactoring sans casser l'existant
- Patterns React modernes (hooks, cleanup)
- Testing avec Jest + React Testing Library
- Debugging avancé avec outils React
- Résilience

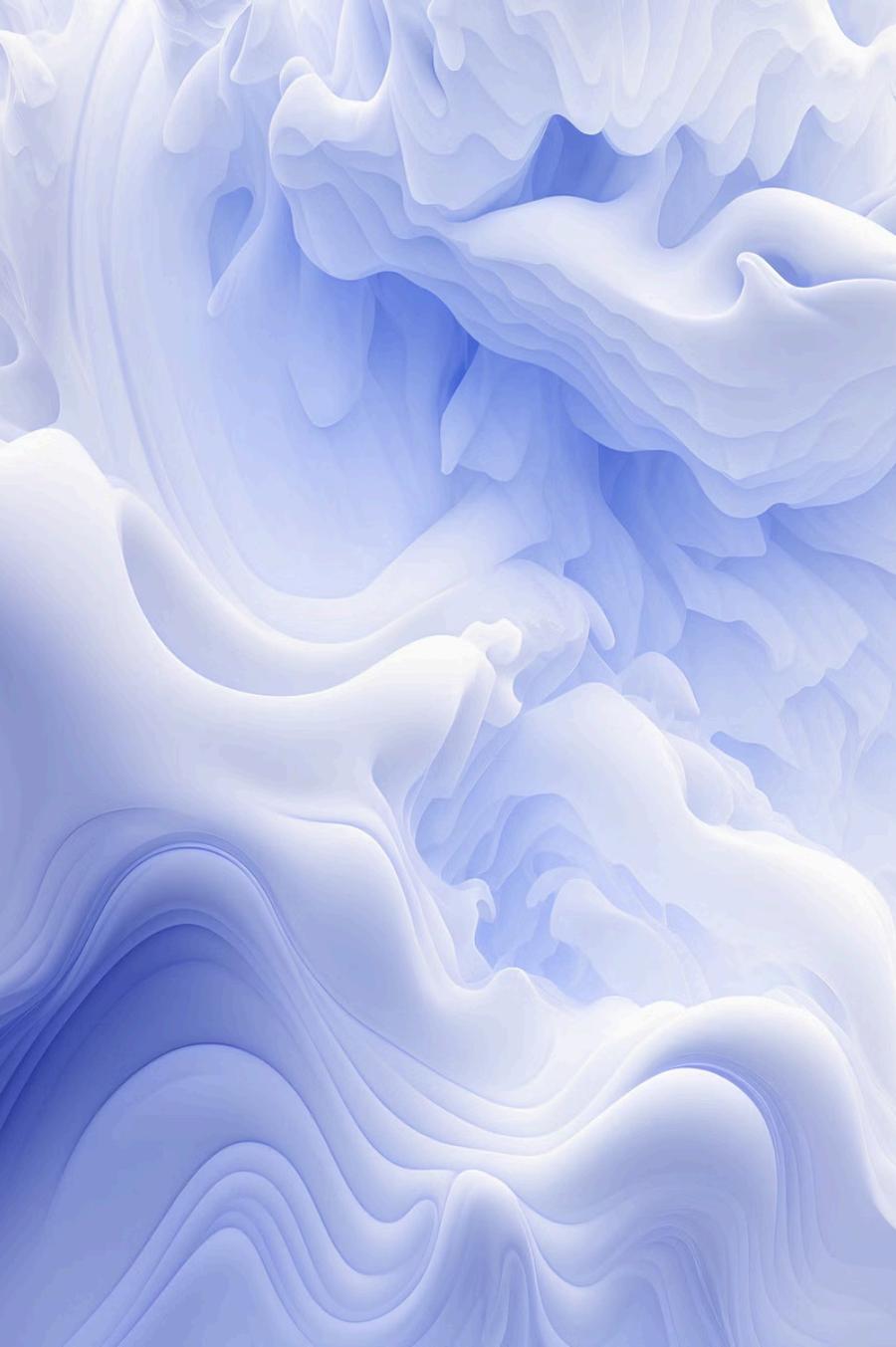
⚠️ Points d'Amélioration Identifiés

- Pas de fallback global en cas d'erreur API
- Pas de fonction qui vérifie les champs du formulaire
- events.json redondant
- Documentation JsDoc
- Séparation des logiques métiers

🏆 Bilan Global

Objectifs Atteints

- ✓ 5 bugs critiques corrigés et documentés
- ✓ Site 100% fonctionnel et validé
- ✓ 12 scénarios de recette passants
- ✓ 7 tests unitaires ajoutés avec succès
- ✓ Documentation technique complète



Conclusion

Ce projet a été une expérience formatrice et enrichissante, me permettant de développer des compétences techniques et méthodologiques essentielles au métier de développeur web. Je suis fier du travail accompli ainsi que des connaissances acquises, et je suis motivé à mettre en œuvre ces acquis dans mes futurs projets.

Bug n°1 - Le filtrage des événements par catégorie est défectueux

• Diagnostic du Bug

- **Test associé :** When Events is created > and we select a category > an filtered list is displayed
- **But du test :** Ce test simule un clic sur un filtre de catégorie et vérifie que les événements n'appartenant pas à cette catégorie disparaissent bien de la liste.
- **Emplacement :** .../src/containers/Events/index.test.js

Analyse des corrections apportées au composant EventList

1.-Problème : Filtrage dysfonctionnel

Problème : (!type ? data?.events : data?.events) - même condition dans les deux cas

Consequence : Le filtrage par type ne fonctionne pas, tous les événements s'affichent

Solution : Séparation claire : filtrage par type puis pagination

Amélioration : Logique de filtrage complètement refaite pour séparer le filtrage par type de la pagination

Explication : La condition ternaire retournait data?.events dans les deux cas, rendant le filtrage par type inefficace



AVANT - Même condition partout

```
const filteredEvents = (  
  !type ? data?.events : data?.events) || []  
).filter(...)
```



APRÈS - Filtrage réel

```
const typeFilteredEvents = allEvents.filter((event) => !type || event.type === type);
```

Si aucun type n'est précisé, garde tous les événements. Sinon, garde seulement ceux dont le type correspond.

2.-Problème : Pagination complexe et incorrecte

Problème : Filter sur l'index au lieu d'utiliser slice, logique complexe et peu performante

Consequence : Pagination peu claire et potentiellement buggée

Solution : slice() avec calculs directs pour une pagination claire et efficace



AVANT - Logique tortueuse

```
.filter((event, index) => {  
  if (  
    (currentPage - 1) * PER_PAGE <= index &&  
    PER_PAGE * currentPage > index  
  ) {  
    return true;  
  }  
  return false;  
});
```



APRÈS - Simple et clair

```
const paginatedEvents = typeFilteredEvents.slice(  
  (currentPage - 1) * PER_PAGE,  
  currentPage * PER_PAGE  
)
```

3.-Problème : Calcul de pages incorrect

Problème : Math.floor((filteredEvents?.length || 0) / PER_PAGE) + 1 - logique de calcul erronée

Consequence : Nombre de pages incorrect, pagination cassée

Amélioration : Calcul mathématiquement incorrect corrigé

Solution : Utilisation de Math.ceil pour arrondir au supérieur

Explication : Math.floor + 1 donnait un nombre de pages insuffisant (ex: 10 éléments = 1 page au lieu de 2)



AVANT - Dernière page manquante

```
const pageNumber = Math.floor(filteredEvents.length / PER_PAGE) + 1;
```



APRÈS - Calcul correct avec Math.ceil

```
const totalPages = Math.ceil(typeFilteredEvents.length / PER_PAGE);  
const pageNumber = totalPages > 0 ? totalPages : 1;
```

4.-Problème : Navigation inaccessible

Problème : Liens <a> au lieu de boutons pour la pagination

Consequence : Mauvaise accessibilité, pas d'indication de la page active

Solution : Boutons sémantiques avec état disabled pour la page courante



AVANT - Liens sans indication d'état

```
{[...Array(pageNumber)].map(_, n) => (  
  <a href="#events" onClick={() => setCurrentPage(n + 1)}>  
    {n + 1}  
  </a>  
)}
```



APRÈS - Boutons avec état visuel

```
{Array.from({ length: pageNumber }, (_, index) => (  
  <button  
    onClick={() => setCurrentPage(index + 1)}  
    disabled={currentPage === index + 1} // Page active  
  >  
    {index + 1}  
  </button>  
)})
```

5.-Problème : Gestion défaillante du changement de type

Problème : setType(evtType) sans traitement de la valeur "Toutes"

Consequence : Le filtre "Toutes" ne remet pas à zéro le filtrage

Solution : Gestion explicite du cas "Toutes"

Amélioration : Aucune logique pour remettre le filtre à zéro

Explication : Si evtType est "Toutes" ou falsy, on met null pour afficher tous les événements

AVANT - Pas de gestion du cas "Toutes"

```
const changeType = (evtType) => {  
  setCurrentPage(1);  
  setType(evtType);  
};
```


APRÈS - Gestion du cas "Toutes"

```
const changeType = (evtType) => {  
  setCurrentPage(1);  
  setType(evtType === "Toutes" || !evtType ? null : evtType);  
};
```

Bug n°2 - La date ne s'affiche pas correctement sur les cartes d'événement

• Diagnostic du Bug

- **Test associé :** When a event card is created > a title, a label and a month are displayed
- **But du test :** Ce test s'assure qu'une carte d'événement affiche toutes ses informations essentielles, y compris le bon nom du mois (ici "avril").
- **Emplacement :** .../src/components/EventCard/index.test.js

Analyse des corrections apportées au composant EventCard

1.-Problème : Import incorrect de getMonth

Problème : import { getMonth } alors que c'est un export default

Conséquence : getMonth est undefined, le mois ne s'affiche pas

Solution : Import direct sans destructuring

Amélioration : Import cohérent avec l'export default du module

Explication : Import direct qui correspond à l'export par défaut de la fonction



AVANT - Import avec destructuring

```
import { getMonth } from "../../helpers/Date";
```



APRÈS - Import direct

```
import getMonth from "../../helpers/Date";
```

Analyse des corrections apportées à Helpers/Date

1.- Problème : Fonction getMonth défaillante

Problème : MONTHS[date.getMonth()] avec index décalé (getMonth() retourne 0-11)

Conséquence : Mois incorrect affiché (décalage d'un mois)

Solution : Utilisation de l'API native toLocaleDateString

Amélioration : Utilisation de l'API native fiable et internationalisée

Explication : toLocaleDateString avec options gère automatiquement la localisation française sans risque d'erreur d'index



AVANT - Mapping manuel avec décalage

```
export const MONTHS = {
  1: "janvier", 2: "février", /* ... */
};

export const getMonth = (date) => MONTHS[date.getMonth()];
```



APRÈS - API native sans décalage

```
const getMonth = (date) => {
  const options = { month: 'long' };
  return new Date(date).toLocaleDateString('fr-FR', options);
};

export default getMonth;
```

Bug n°3 - La logique de soumission du formulaire est rompue

- **Diagnostic du Bug**

- **Test associé :** When Events is created > and a click is triggered on the submit button > the success action is called
- **But du test :** Ce test s'assure que lorsqu'un utilisateur clique sur le bouton "Envoyer", la fonction onSuccess est bien appelée, confirmant que l'action principale est bien déclenchée.
- **Emplacement :** .../src/containers/Form/index.test.js

Analyse des corrections apportées au composant Form

1.- Problème : Callback onSuccess jamais appelé

Problème : onSuccess() absent dans le bloc try/catch

Conséquence : Pas de feedback utilisateur après envoi réussi

Solution : Ajout de l'appel à onSuccess() après succès

Amélioration : Notification complète du succès au composant parent

Explication : onSuccess() déclenche l'affichage de la modal "Message envoyé !" comme attendu par le test



AVANT - onSuccess manquant

```
try {  
    await mockContactApi();  
    setSending(false);  
} catch (err) {  
    setSending(false);  
    onError(err);  
}
```



APRÈS - onSuccess appelé

```
const try {  
    await mockContactApi();  
    setSending(false);  
    onSuccess();  
} catch (err) {  
    setSending(false);  
    onError(err);  
}
```

Bug n°4 - Correction du Slider

• Diagnostic du Bug

- **Test associé :** When slider is created > a list card is displayed
- **But du test :** Ce test vérifie que le slider, au chargement, affiche bien le premier événement attendu (le plus récent, celui de "janvier"), ce qui valide la logique de tri.
- **Emplacement :** .../src/containers/Slider/index.test.js

Analyse des corrections apportées au container Slider

1.- Problème : Slide blanche en fin de boucle et désynchronisation

- **Problème :** La logique pour déterminer la slide active et le bouton radio coché était inutilement complexe et basée sur un calcul inversé, ce qui menait à des erreurs d'index ("off-by-one error").
- **Conséquence :** Le carrousel affichait une slide vide (blanche) lorsqu'il arrivait à la dernière image. De plus, le bouton radio actif n'était pas synchronisé avec l'image affichée.
- **Solution :** Remplacer la logique complexe par une gestion d'index simple et directe. L'état index correspond maintenant directement à la position de la slide dans le tableau.
- **Amélioration :** Le code est plus simple, plus lisible et, surtout, correct. Le carrousel boucle parfaitement et l'interface reste cohérente à tout moment.



AVANT - (Logique incorrecte)

```
// La slide active était calculée avec une logique inversée et fragile
{byDateDesc.map((event, idx) =>
  <div>
    // La comparaison était souvent source de bugs
    className={'SlideCard SlideCard--${index === byDateDesc.length - 1 - idx ? "display" : "hide"}'}
  >
    {/* ... */}
  </div>
)}
 {/* ... */}
/* Le bouton radio coché utilisait la même logique erronée */
<input
  type="radio"
  name="radio-button"
  // Cette formule était la cause principale du bug
  checked={index === byDateDesc.length - 1 - radioidx}
/>
```



APRÈS - (Logique corrigée et simplifiée)

```
// La boucle du minuteur est maintenant robuste grâce au modulo (%)
useEffect(() => {
  const interval = setInterval(() => {
    // Fait cycler l'index de 0 à la fin, puis revient à 0
    setIndex((prev) => (prev + 1) % byDateDesc.length);
  }, 5000);
  // ...
}, [byDateDesc.length]);

// La comparaison est devenue simple et directe
{byDateDesc.map((event, idx) =>
  <div>
    className={'SlideCard SlideCard--${index === idx ? "display" : "hide"}'}
  >
    {/* ... */}
  </div>
)}
 {/* ... */}
/* La synchronisation du bouton radio est maintenant triviale et correcte */
<input
  key={eventRadio.id}
  type="radio"
  name={`radio-button-${eventRadio.id}`}
  checked={index === radioidx}
  onChange={() => setIndex(radioidx)}
/>
```

Bug Corrigé n°2 : Avertissement React sur la prop "key"

- **Problème :** L'index du tableau (idx) était utilisé comme key pour les éléments générés dans la boucle .map().
- **Conséquence :** React affichait un avertissement dans la console. Plus gravement, cela peut causer des bugs de rendu et des performances dégradées, car React ne peut pas identifier de manière fiable les éléments si la liste est modifiée (triée, filtrée, etc.).
- **Solution :** Utiliser un identifiant unique et stable qui est lié à la donnée elle-même, ici event.id.
- **Amélioration :** L'application respecte les bonnes pratiques de React. Elle est plus performante et plus robuste, car le rendu de la liste est optimisé et prévisible.



AVANT - (Mauvaise pratique avec key={index})

```
// La slide active était calculée avec une logique inversée et fragile
{byDateDesc.map((event, idx) =>
  <div>
    // La comparaison était souvent source de bugs
    className={'SlideCard SlideCard--${index === byDateDesc.length - 1 - idx ? "display" : "hide"}'}
  >
    {/* ... */}
  </div>
)}
 {/* ... */}
/* Le bouton radio coché utilisait la même logique erronée */
<input
  type="radio"
  name="radio-button"
  // Cette formule était la cause principale du bug
  checked={index === byDateDesc.length - 1 - radioidx}
/>
```



APRÈS - (Bonne pratique avec key={id})

```
// Chaque slide a une key unique et stable issue des données
{byDateDesc.map((event, idx) =>
  <div
    key={event.id} // ID unique de l'événement
    className={'SlideCard...'}
  >
    {/* ... */}
  </div>
)}

// De même pour les boutons radio
{byDateDesc.map((eventRadio, radioidx) => (
  <input
    key={eventRadio.id} // ID unique, la meilleure solution
    type="radio"
    checked={index === radioidx}
  />
))
}
```

Bug Corrigé n°3 : Fuite de mémoire avec setInterval

- **Problème :** Le useEffect qui lançait le minuteur setInterval n'avait pas de fonction de nettoyage pour l'arrêter.
- **Conséquence :** Fuite de mémoire. Si l'utilisateur quittait la page, le composant Slider était "démonté", mais le minuteur continuait de s'exécuter en arrière-plan pour toujours, consommant des ressources inutilement et pouvant causer des erreurs si l'utilisateur revenait sur la page.
- **Solution :** Retourner une fonction de nettoyage (cleanup function) depuis le useEffect. Cette fonction exécute clearInterval pour stopper le minuteur lorsque le composant est sur le point d'être démonté.
- **Amélioration :** L'application est plus stable et gère correctement ses ressources. Le cycle de vie du composant est respecté, prévenant les fuites de mémoire et les comportements inattendus.



AVANT - (Fuite de mémoire)

```
useEffect(() => {
  // Le minuteur est créé, mais jamais détruit.
  setInterval(() => {
    setIndex(prevIndex => prevIndex + 1);
  }, 5000);

  // Pas de fonction de nettoyage retornée !
}, []); // Une dépendance vide pouvait aussi causer d'autres bugs
```



APRÈS - (Gestion correcte du cycle de vie)avec key={id})

```
// CauseEffect(() => {
  let cleanup;

  if(byDateDesc.length > 0) {
    const interval = setInterval(() => {
      setIndex((prev) => (prev + 1) % byDateDesc.length);
    }, 5000);

    // La fonction de nettoyage est définie
    cleanup = () => clearInterval(interval);
  }

  // React appellera cette fonction quand le composant sera démonté
  return cleanup;
}, [byDateDesc.length]); // Dépendance correcte pour relancer si besoin
```

Bug n°5 - Correction du Bug du formulaire ?

- **Diagnostic du Bug**
- **Test associé :** When Form is created > and a click is triggered on the submit button > the success message is displayed
- **But du test :** Ce test vérifie que le formulaire de contact fonctionne correctement, affiche "En cours" puis "Message envoyé !" après soumission.
- **Emplacement :** .../src/pages/Home/index.test.js

Analyse des corrections apportées à la home page

1.- Gestion manquante de l'état last undefined/null

- **Problème :** Le code original tentait d'accéder aux propriétés de last sans vérifier si cet objet existait, et utilisait l'optional chaining ?. de manière incomplète.
- **Conséquence :** Erreur JavaScript Cannot read properties of null/undefined lors du rendu si aucune dernière prestation n'existe, causant un crash de l'application et empêchant l'exécution des tests.
- **Solution :** Ajout d'une vérification conditionnelle complète avec {last ? ... : ...} pour rendre soit le composant EventCard avec Modal, soit un message d'information.
- **Amélioration :** L'application est robuste face aux données manquantes et offre une expérience utilisateur gracieuse même quand aucune prestation n'est disponible.



AVANT - (Crash potentiel avec données manquantes)

```
<div className="col presta">
  <h3>Notre dernière prestation</h3>
  <EventCard
    imageSrc={last?.cover}
    title={last?.title}
    date={new Date(last?.date)} // Crash si last est null !
    small
    label="boom"
  />
</div>
```



APRÈS - (Gestion sécurisée des données manquantes)

```
<div className="col presta">
  <h3>Notre dernière prestation</h3>
  {last ? (
    <Modal Content={<ModalEvent event={last} />}>
      {{ setIsOpened }} => (
        <EventCard
          onClick={() => setIsOpened(true)}
          imageSrc={last.cover}
          title={last.title}
          date={new Date(last.date)}
          small
          label={last.type}
        />
      )
    </Modal>
  ) : (
    <p>Pas de dernière prestation à afficher pour le moment.</p>
  )}
</div>
```

Bug Corrigé n°2 : Import manquant du composant ModalEvent

- **Problème :** Le composant ModalEvent était référencé dans le code mais son import était absent des déclarations en haut du fichier.
- **Conséquence :** Erreur de référence ModalEvent is not defined au moment du rendu, empêchant complètement l'exécution de l'application et des tests.
- **Solution :** Ajout de l'import manquant import ModalEvent from ".../containers/ModalEvent"; dans la liste des imports.
- **Amélioration :** Résolution complète des dépendances du composant, permettant un fonctionnement normal et l'exécution des tests.



APRÈS - (Toutes les dépendances importées)

```
import Modal from ".../containers/Modal";
import ModalEvent from ".../containers/ModalEvent"; // Import ajouté
```

Bug Corrigé n°3 : Absence d'identifiants pour la navigation et les tests

- **Problème :** Les sections principales de la page n'avaient pas d'attributs id, rendant impossible la navigation interne et compliquant la localisation des éléments lors des tests.
- **Conséquence :** Tests fragiles car ils ne peuvent pas cibler précisément les sections, et navigation utilisateur impossible vers des sections spécifiques de la page.
- **Solution :** Ajout d'attributs id descriptifs sur toutes les sections principales (id="nos-services", id="nos-realisations", id="notre-equipe").
- **Amélioration :** Meilleure accessibilité, navigation interne possible, et tests plus robustes grâce à des sélecteurs fiables.



AVANT - (Sections sans identifiants)

```
<section className="ServicesContainer">
  <h2 className="Title">Nos services</h2>
```

```
<section className="EventsContainer">
  <h2 className="Title">Nos réalisations</h2>
```

```
<section className="PeoplesContainer">
  <h2 className="Title">Notre équipe</h2>
```



APRÈS - (Sections identifiées pour navigation et tests)

```
<section className="ServicesContainer" id="nos-services">
  /* Added id for potential navigation */
  <h2 className="Title">Nos services</h2>
```

```
<section className="EventsContainer" id="nos-realisations">
  <h2 className="Title">Nos réalisations</h2>
```

```
<section className="PeoplesContainer" id="notre-equipe">
  <h2 className="Title">Notre équipe</h2>
```

Bug Corrigé n°4 : Label statique incorrect dans EventCard

- **Problème :** Le label de l'EventCard était codé en dur avec la valeur "boom" au lieu d'utiliser la propriété dynamique type de l'objet last.
- **Conséquence :** Affichage incohérent et incorrect des informations, ne reflétant pas le vrai type d'événement, créant une confusion pour l'utilisateur et des tests basés sur de fausses données.
- **Solution :** Remplacement de la chaîne statique "boom" par la propriété dynamique last.type pour afficher le vrai type d'événement.
- **Amélioration :** Affichage cohérent et correct des données réelles, interface utilisateur fidèle aux informations stockées.



AVANT - (Label statique incorrect)

```
<EventCard
  imageSrc={last?.cover}
  title={last?.title}
  date={new Date(last?.date)}
  small
  label="boom" // Valeur statique incorrecte !
/>
```


APRÈS - (Label dynamique correct)

```
<EventCard
  onClick={() => setIsOpened(true)}
  imageSrc={last.cover}
  title={last.title}
  date={new Date(last.date)}
  small
  label={last.type} // Valeur dynamique correcte
/>
```

Bug Corrigé n°5 : Manque d'interactivité sur EventCard

- **Problème :** L'EventCard dans le footer n'était pas interactive et ne permettait pas d'ouvrir une modal pour voir les détails de l'événement, contrairement aux autres EventCard de la page.
- **Conséquence :** Expérience utilisateur incohérente où certains EventCard sont cliquables et d'autres non, limitant la fonctionnalité et rendant impossible les tests d'interaction.
- **Solution :** Encapsulation de l'EventCard dans un composant Modal avec un handler onClick pour ouvrir les détails de l'événement.
- **Amélioration :** Interface utilisateur cohérente et pleinement interactive, permettant aux utilisateurs d'accéder aux détails de tous les événements affichés.

AVANT - (EventCard non-interactive)

```
<EventCard
  imageSrc={last?.cover}
  title={last?.title}
  date={new Date(last?.date)}
  small
  label="boom"
/>
```

```
// Pas de Modal, pas d'interaction possible
```


APRÈS - (EventCard interactive avec Modal)

```
<Modal Content={<ModalEvent event={last} />}>
  {{ setIsOpened }} => (
    <EventCard
      onClick={() => setIsOpened(true)} // Handler d'interaction ajouté
      imageSrc={last.cover}
      title={last.title}
      date={new Date(last.date)}
      small
      label={last.type}
    />
  )
</Modal>
```