Scénario 1 : Problème de défilement d'images dans le Slider

Solution expliquée :

Lors de ma recherche avec react dev tools j'ai pu voir que le state bouclait de 0 à 3 alors que la longueur de focus est de 3 éléments (donc de 0 à 2). Le problème venait du bout de code suivant que j'ai modifié :

```
const nextCard = () => {
    setTimeout(
      () => setIndex(index < byDateDesc.length-1 ? index + 1 : 0),
      5000
    );
```

Dans le code original il manquait le -1 à byDataDesc.length, ce qui le faisait du coup boucler sur 4 éléments au lieu de 3.

Scénario 2: Le problème d'affichage des mois

Solution expliquée :

En observant les mois affichés sur le site et en les comparant avec ceux du fichier json, j'ai remarqué que tous avait un mois de décalage. Pour rectifier ce bug, je suis allé dans le fichier helper/date où il suffisait de réduire les valeurs des mois de 1, de facon a ce que le tableau commence de zero afin que ce soit la première valeur de celui-ci.
Voici le code modifié :

```
export const MONTHS = {
  0: "janvier",
  1: "février",
  2: "mars",
```

```
  3: "avril",
  4: "mai",
  5: "juin",
  6: "juillet",
  7: "août",
  8: "septembre",
  9: "octobre",
  10: "novembre",
  11: "décembre",
};


              Scénario 3: défilement des bullets

Given : L'utilisateur consulte le slider
When : Le slider passe d'une image à l'autre toutes les 5 secondes
Then : Les bullets suivent la rotation des images

Solution expliquée :

Dans le code ci-dessous nous avions checked = {idx===radioIdx}. Celui
ci ne pouvait pas permettre de faire fonctionner les bullets
correctement car idx est utilisé comme l'index de map dans
{byDateDesc?.map((event, idx)
Pour corriger ce bug il fallait remplacer idx par index qui lui grace à
son state garde la trace de la carte actuellement affichée.

 <div className="SlideCard__paginationContainer">
            <div className="SlideCard__pagination">
              {byDateDesc.map((_, radioIdx) => (
                <input
                  key={`${event.id}`}
                  type="radio"
                  name="radio-button"
                  checked={index === radioIdx}
                />


     Scénario 4 : Filtrer les éléments par catégorie
```

Solution expliquée :

J’ai remplacé une partie du code de la page Events qui était censé faire fonctionner le filtre mais qui ne réagissait pas. Comme vous pourrez le voir ci-dessous je l’ai remplacé avec useEffect qui exécute sa fonction à chaque fois que le type est changée

```
useEffect (() => {
    if (!data) return;

    console.log("Data:", data);
     console.log("Type:", type);
      console.log("Current Page:", currentPage);

    const events = (
    (!type
      ? data?.events
      : data?.events.filter(event => event.type === type)) || []
  ).filter((event, index) => {
    if (
      (currentPage - 1) * PER_PAGE <= index &&
      PER_PAGE * currentPage > index
    ) {
      return true;
    }
    return false;
  });
  setFilteredEvents(events);
  console.log("Filtered Events:", events);
 }, [type, currentPage, data]);
```

J’ai aussi rajouté l’argument newValue dans la fonction onChange de notre composant Select afin que celui ci nous affiche bien les nouvelles valeurs lors du changement de catégorie.

```
 const changeValue = (newValue) => {
    onChange(newValue);
    setValue(newValue);
    setCollapsed(newValue);
  };
```

Scénario 5 : Soumission du formulaire avec confirmation par modale

Pour déboguer ce problème, je me suis rendu compte qu’il manquait dans le containers form, le rappel de la fonction onSuccess() apres la soumission réussie de la fonction sendContact

```
const sendContact = useCallback(
    async (evt) => {
      evt.preventDefault();
      setSending(true);
      // We try to call mockContactApi
      try {
        await mockContactApi();
        setSending(false);
        onSuccess();
      } catch (err) {
        setSending(false);
        onError(err);
      }
    },
    [onSuccess, onError]
  );

     Scénario 6 : Affichage du dernier élément dans le footer

Solution expliquée :

Dans le dataContext :

J'ai commencé par créer last et setLast avec un useState(null) pour
pouvoir intégrer la nouvelle valeur de last au dernier élément de la
liste events sans oublier d'ajouter last en tant que valeur à
dataContext.provider :

const [last, setLast] = useState(null);

  const getData = useCallback(async () => {
    try {
      const response = await api.loadData()
      setData(response);
     if (response && response.events && response.events.length) {
      setLast(response.events[response.events.length - 1]);
     }
     console.log({ last: response.events[response.events.length - 1] },
"toto", { data: response });
    } catch (err) {
      setError(err);
    }
```

```
  }, []);
  useEffect(() => {
    if (data) return;
    getData();

  }, [data, getData]);

  return (
    <DataContext.Provider
      // eslint-disable-next-line
react/jsx-no-constructed-context-values
      value={{
        data,
        error,
        last,
      }}
    >

Dans page :
```

J’ai commencé par récupérer le dataContext avec useData :
import { useData } from "../../contexts/DataContext";

Puis j’ai ajouté l’état last afin de pouvoir stocker les données du dernier élément :
const { last } = useData();

J’ai ensuite ajouté des vérifications pour m’assurer que last.cover et last.title soient bien définies avant de pouvoir les afficher grâce à eventCard