

Video : https://video.deakin.edu.au/media/t/1_gtq6a5ao

Git: <https://github.com/CameronHart1/SIT313-FullStack>

Screenshots:

The top screenshot shows a web application interface with a header bar containing the text "DEV@DEAKIN", a search input field, and links for "Post" and "Login". Below the header, there are filters for "search", "tags", and "Dates". The "Dates" filter is set to "None". A section titled "Questions" displays a list of three items, each with a title, tags, and a "Delete" button.

| Questions | Delete |
|--|--------|
| Third times the charm tags: RIPBOZO fix oops | Delete |
| I should mae this reset tags: annoying feature fix | Delete |
| Get help tags: boss fix help | Delete |

The bottom screenshot shows a single question detail page. It has a header bar with "Explore", "Support", and "Stay Connected" links. Below the header, there are filters for "tags", "Dates", and "Questions". The "tags" filter is set to "RIP". The "Dates" filter is set to "None". The "Questions" section displays a single question with a title, author, date, content, tags, and a "Delete" button.

| Questions | Delete |
|---|--------|
| Third times the charm Author: Shronkle Date: Aug 25th 2022 Yeah this is the last filler question, I can mess with the filters and stuff now RIP BOZO tags: RIPBOZO fix oops | Delete |

```

const QuestionView = (props) => {
  const [expanded, setExpanded] = useState(false);
  const { title, rate, author, content, tags, date } = props.question;

  const ToggleExpand = () => {
    setExpanded(!expanded);
  };

  return (
    <div className="QuestionBoxDiv">
      <div className="expandDiv" onClick={ToggleExpand}>
        <h1>{title}</h1>
        {expanded && (
          <div>
            <p>
              Author: {author} Date: {moment(date).format("MMM Do YYYY")}
            </p>
            <p>{content}</p>
          </div>
        )}
        <p className="tags">tags: {tags.join(" ")}</p>
      </div>
      <div className="deleteDiv" onClick={(e) => props.removeBox(props.id)}>
        <p>Delete</p>
      </div>
    </div>
  );
};

```

```

92 const keys = Object.keys(questionsList).filter((i) => {
93   let qi = questionsList[i];
94   const df = filter.dateFilter;
95   const dd = moment(qi.date);
96   const id = filter.date ? moment(filter.date) : null;
97   const dateTest = () => {
98     if (df !== "none") {
99       if (df === "After") return dd.isSameOrAfter(id, "day");
100       if (df === "Before") return dd.isSameOrBefore(id, "day");
101       if (df === "Same") return dd.isSame(id, "day");
102     }
103     return true;
104   };
105   const concatTags = qi.tags.sort().join(" ")
106
107   return `${qi.title} ${qi.content}`.includes(filter.text)
108     ? filter.tags.length > 0
109       ? filter.tags.every((e) => concatTags.includes(e))
110       : dateTest()
111     : false;
112 });
113
114 return (
115   <div>
116     {keys.map((q) => {
117       return (
118         <QuestionView
119           key={q}
120           question={questionsList[q]}
121           id={q}
122           removeBox={removeBox}
123         />
124       );
125     })}
126   </div>
127 );
128 };
129

```

```

return (
  <div>
    <p>search</p>
    <input
      type="text"
      name="text"
      placeholder="Search"
      onChange={handleChange}
    />
    <p>tags</p>
    <input
      type="text"
      name="tags"
      placeholder="tags"
      onChange={handleChange}
    />
    <p>Dates</p>
    <select id="dateDD" name="dateFilter" onChange={handleChange}>
      <option value="None">None</option>
      <option value="After">After</option>
      <option value="Before">Before</option>
      <option value="Same">Same day</option>
    </select>
    {filter.dateFilter !== "None" && (
      <input
        type="date"
        name="date"
        placeholder={new Date()}
        onChange={handleChange}
      />
    )}
    <h1>Questions</h1>
    <QuestionList
      questionsList={questionsList}
      filter={filter}
      removeBox={removeBox}
    />
  </div>
)

```

```

const QuestionPage = (props) => {
  const context = useContext(PostContext);

  const [questionsList, setQuestionsList] = useState(
    context.currentPosts.questions
  );
  const [filter, setFilter] = useState({
    tags: [],
    text: "",
    dateFilter: "none",
    date: new Date(),
  });

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFilter((preValue) => {
      if (name === "tags")
        return {
          ...preValue,
          [name]: value.split(/(?:,| |#)+/).filter((s) => s),
        };
      return {
        ...preValue,
        [name]: value,
      };
    });
  };

  // it's wierd but mutating the array with delete doesnt live update so instead we filter out the deleted ID
  const removeBox = (id) => {
    console.log("removed " + id);
    const tmp = Object.keys(questionsList).
      filter((k) => k !== id).
      reduce((cur, k) => { return Object.assign(cur, { [k]: questionsList[k] }); }, {});
    setQuestionsList(tmp);
  };
}

```

```

import { useReducer } from "react";
import { useState, useEffect } from "react";
import { createContext } from "react";
import { fetchQuestionsAndTutorials } from "../utils/firebase";
// import { addCollectionAndDocument } from "../utils/firebase";

// Basically making it stored in session, so updating page doesnt remove everything
let reducer = (info, newInfo) => {
  if (newInfo === null) {
    sessionStorage.removeItem("Posts");
    return null;
  }
  return { ...info, ...newInfo };
};

const localState = JSON.parse(sessionStorage.getItem("Posts"));

export const PostContext = createContext({
  currentPosts: null,
  setCurrentPosts: () => null,
});

export const PostProvider = ({ children }) => {
  // useEffect(()=>{
  //   addCollectionAndDocument('posts',)
  // },[])
  const [currentPosts, setCurrentPosts] = useReducer(
    reducer,
    localState || null
  );
  const value = { currentPosts, setCurrentPosts };

  useEffect(() => {
    sessionStorage.setItem("Posts", JSON.stringify(currentPosts));
    if (currentPosts === null) {
      fetchQuestionsAndTutorials().then((data) => setCurrentPosts(data));
    }
  }, [currentPosts]);

  export const addCollectionAndDocument = async (collectionKey, objectsToAdd) => {
    const collectionRef = collection(db, collectionKey);
    const batch = writeBatch(db);
    objectsToAdd.forEach((obj) => {
      const docRef = doc(collectionRef, obj.id.toLowerCase());
      batch.set(docRef, obj);
    });
    await batch.commit();
    console.log("succesful batch commit");
  };

  // -----
  export const fetchQuestionsAndTutorials = async () => {
    const articleCollectionRef = collection(db, "Articles");
    const aq = query(articleCollectionRef);
    const articleSnaphsot = await getDocs(aq);
    const ArticleMap = articleSnaphsot.docs.reduce((acc, docSnapshot) => {
      const { ...items } = docSnapshot.data();
      console.log(docSnapshot);
      acc[docSnapshot.id] = items;
      acc[docSnapshot.id].date = new Date(acc[docSnapshot.id].date.seconds * 1000);
      return acc;
    }, {});

    const questionCollectionRef = collection(db, "Questions");
    const q = query(questionCollectionRef);
    const questionSnaphsot = await getDocs(q);
    console.log(questionSnaphsot);
    const QuestionMap = questionSnaphsot.docs.reduce((acc, docSnapshot) => {
      const { ...items } = docSnapshot.data();
      acc[docSnapshot.id] = items;
      acc[docSnapshot.id].date = new Date(acc[docSnapshot.id].date.seconds * 1000);
      return acc;
    }, {});
    return { questions: QuestionMap, articles: ArticleMap };
  };

  export const getUserData = async (ref) => {
    const snap = await getDoc(ref);
  }

```