
Using GraphQL APIs
without a client library
LEARNING
VOYAGE

Table of Contents

Using a web UI library

- The most popular UI libraries today are Vue and React.
- I'll use React in this course because it's closer to GraphQL's ecosystem and design principles.
- Besides the fact that both React and GraphQL originated at Facebook, React offers a language to declaratively describe user interfaces (web or otherwise), and GraphQL offers a language to declaratively describe data requirements

Table of Contents

Running the web server

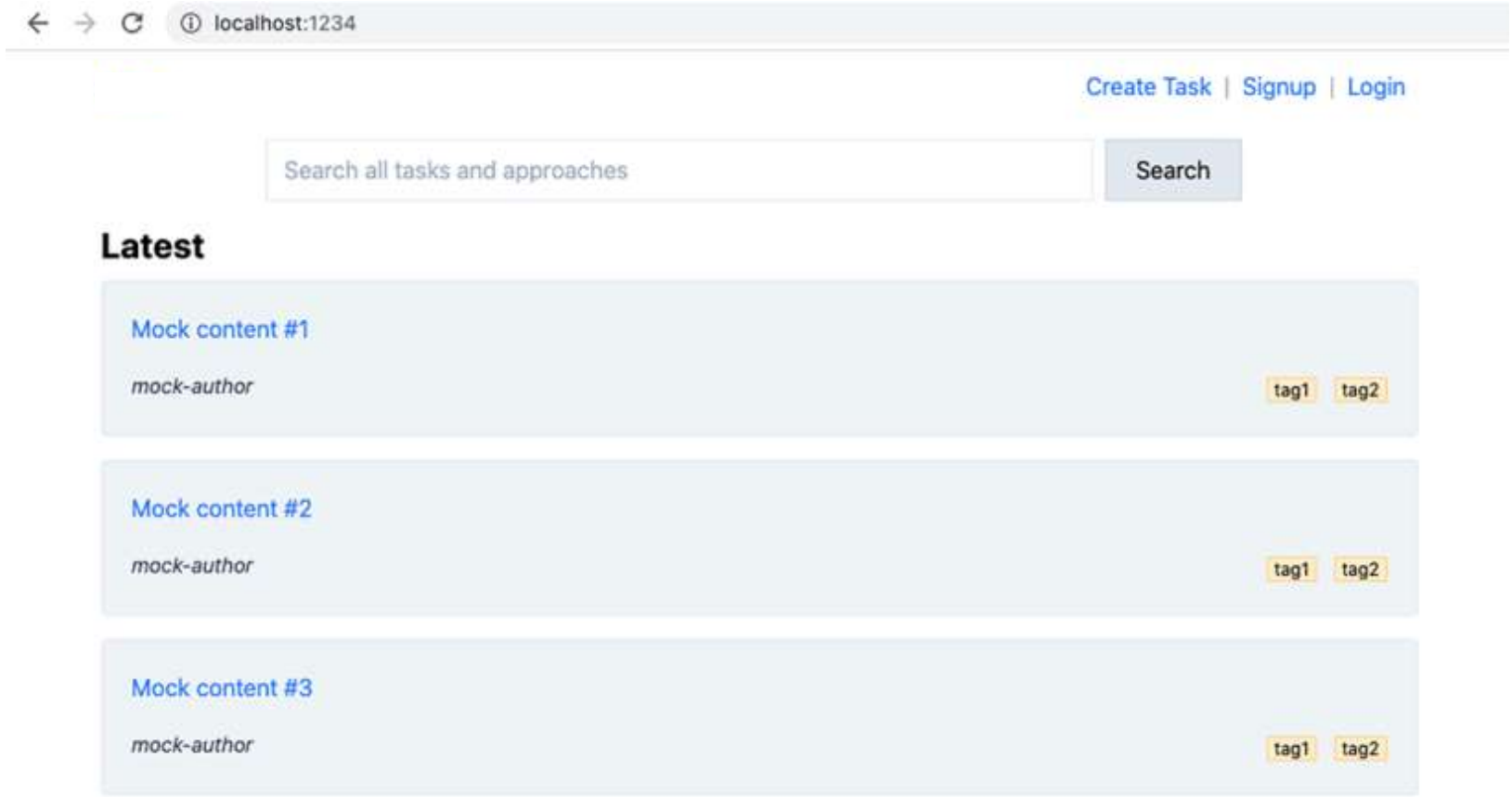
The api and web top-level directories

Running the web server

- With the database and API servers running, use the following command to run the web server.

```
$ npm run web-server
```

Running the web server



Running the web server

- Take a look around the web directory and get familiar with its structure

Running the web server

```
3   import { useStore } from '../store';
4   import Search from './Search';
5   import TaskSummary from './TaskSummary';
6
7   /** GIA NOTES
8    * Define GraphQL operations here...
9    */
10
11  > const mockTasks = [⋯
30  ];
31
32  export default function Home() {
33    const { request } = useStore();
34    const [taskList, setTaskList] = useState(null);
35
36    useEffect(() => {
37      /** GIA NOTES
38       *
39       *  1) Invoke the query to get list of latest Tasks
40       *     (You can't use `await` here but `promise.then
41       *
42       *  2) Change the setTaskList call below to use the
43       *
44       */
45
46      setTaskList(mockTasks); // TODO: Replace mockTasks w
47    }, [request]);
48
```

Table of Contents

```
export default function Home() {
  // ·-·-·

  useEffect(() => {
    request('{ currentTime }').then(({ data }) => {
      console.log(`Server time is:
${data.currentTime}`);
    });

    // ·-·-·
  }, [request]);

  return (
    // ·-·-·
  );
}
```

Making Ajax requests

```
export const useStoreObject = () => {
  // ·-·-·

  const request = async (requestText, { variables } = {}) => {
    const gsResp = await fetch(config.GRAPHQL_SERVER_URL, {
      method: 'post',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ query: requestText, variables }),
    }).then((response) => response.json());

    return gsResp;
  };

  // ·-·-·
};
```

Making Ajax requests

Making Ajax requests

- You should now see the server time log message in your browser console

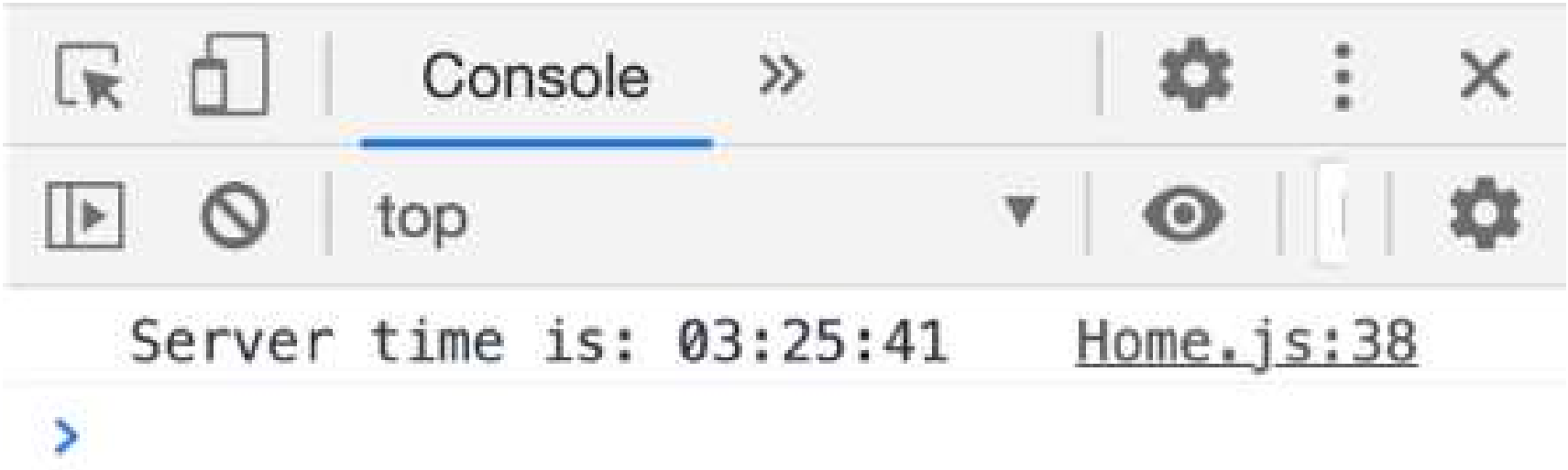


Table of Contents

Performing GraphQL query requests

```
{
  id: 1,
  content: 'Mock content #1',
  author: { username: 'mock-author' },
  tags: ['tag1', 'tag2'],
}
```

Performing GraphQL query requests

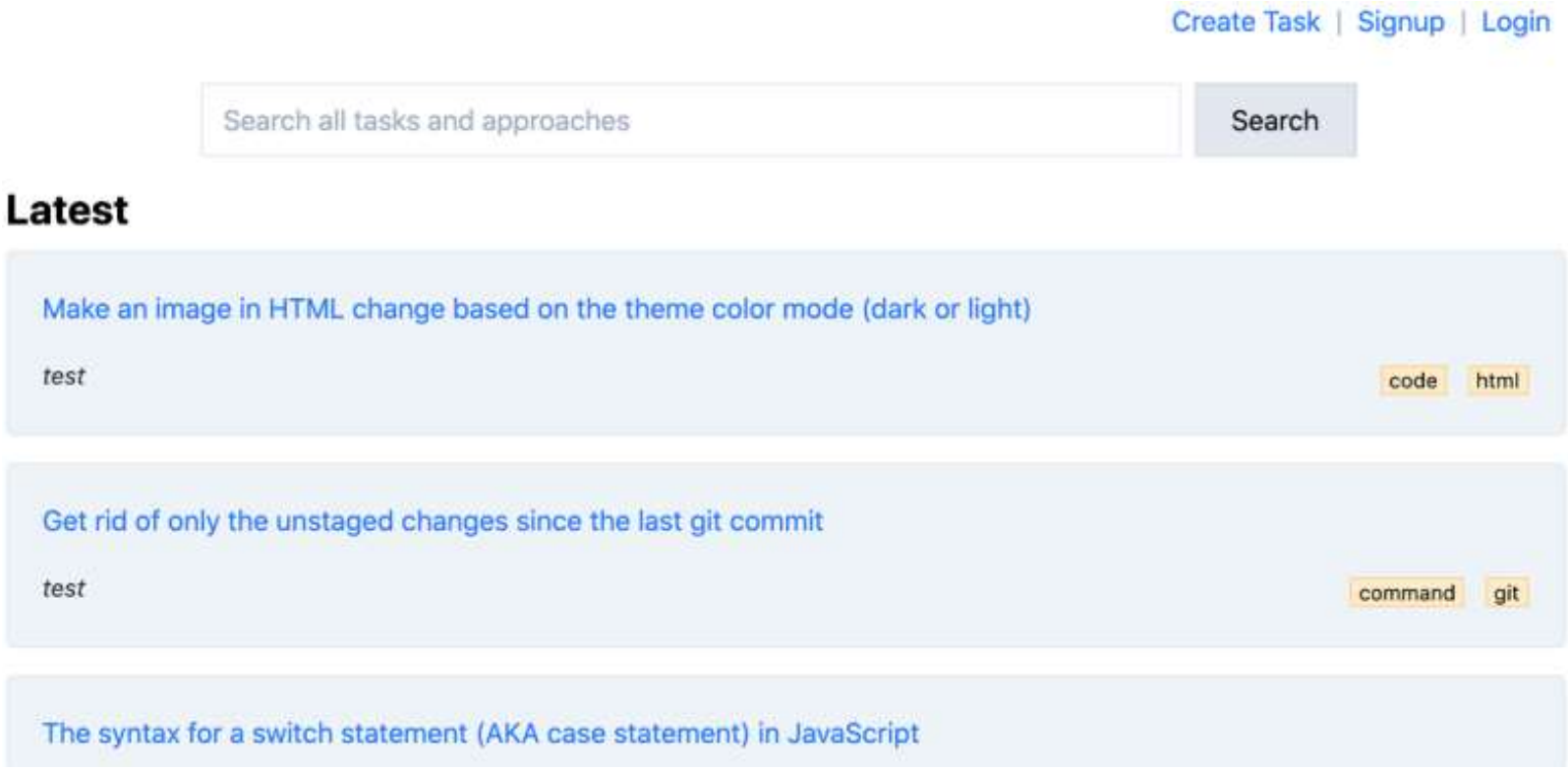
- Relying on the structure of that object, here's the GraphQL query required by the HOME component.

```
query taskMainList {
  taskMainList {
    id
    content
    author {
      username
    }
    tags
  }
}
```

```
const TASK_MAIN_LIST = `
  query taskMainList {
    taskMainList {
      id
      content
      author {
        username
      }
      tags
    }
  }
`;
// delete the mockTasks object...

export default function Home() {
  const { request } = useStore();
  const [ taskList, setTaskList ] = useState(null);

  useEffect(() => {
    request(TASK_MAIN_LIST).then(({ data }) => {
      setTaskList(data.taskMainList);
    });
  }, [request]);

  // ----
}
```

Performing GraphQL query requests

Performing GraphQL query requests



```
export default function TaskSummary({ task, link = false }) {
  const { AppLink } = useStore();

  return (
    <div className="box box-primary">
      {link ? (
        <AppLink to="TaskPage" taskId={task.id}>
          {task.content}
        </AppLink>
      ) : (
        task.content
      )}
      <div className="box-footer">
        <div className="text-secondary">{task.author.username}</div>
        <div className="flex-end">
          {task.tags.map((tag) => (
            <span key={tag} className="box-label">
              {tag}
            </span>
          ))}
        </div>
      </div>
    </div>
  );
}
```

Using GraphQL fragments in UI components

Using GraphQL fragments in UI components

```
// ...
export const TASK_SUMMARY_FRAGMENT = `
  fragment TaskSummary on Task {
    content
    author {
      username
    }
    tags
  }
`;

export default function TaskSummary({ task, link = false }) {
  // ...
}
```

Using GraphQL fragments in UI components

```
// ·-·-·

import TaskSummary, { TASK_SUMMARY_FRAGMENT } from
'./TaskSummary';

const TASK_MAIN_LIST = `
  query taskMainList {
    taskMainList {
      id
      ...TaskSummary
    }
  }
  ${TASK_SUMMARY_FRAGMENT}
`;
// ·-·-·
```

Including variables in requests

```
export default function Approach({ approach, isHighlighted }) {
  // ...

  const [ voteCount, setVoteCount ] = useState(approach.voteCount);

  // ...

  return (
    <div className={`box highlighted-${isHighlighted}`}>
      <div className="approach">
        <div className="vote">
          {renderVoteButton('UP')}
          {voteCount}
          {renderVoteButton('DOWN')}
        </div>
        <div className="main">
          <pre className="code">{approach.content}</pre>
          <div className="author">{approach.author.username}</div>
        </div>
      </div>
      <Errors errors={uiErrors} />
      {approach.detailList.map((detail, index) => (
        <div key={index} className="approach-detail">
          <div className="header">{detail.category}</div>
          <div>{detail.content}</div>
        </div>
      ))}
    </div>
  );
}
```

```
// .-.-.

export const APPROACH_FRAGMENT = `
  fragment ApproachFragment on Approach {
    content
    voteCount
    author {
      username
    }
    detailList {
      content
      category
    }
  }
`;
// .-.-.
```

Using GraphQL fragments in UI components

Using GraphQL fragments in UI components

```
// ·-·-·

import Approach, { APPROACH_FRAGMENT } from './Approach';
import TaskSummary, { TASK_SUMMARY_FRAGMENT } from './TaskSummary';

const TASK_INFO = `
  query taskInfo($taskId: ID!) {
    taskInfo(id: $taskId) {
      id
      ...TaskSummary
      approachList {
        id
        ...ApproachFragment
      }
    }
  }
  ${TASK_SUMMARY_FRAGMENT}
  ${APPROACH_FRAGMENT}
`;
// ·-·-·
```

Using GraphQL fragments in UI components

```
// delete the mockTaskInfo object...

export default function TaskPage({ taskId }) {
  // ----

  useEffect(() => {
    if (!taskInfo) {
      request(TASK_INFO, { variables: { taskId } }).then(
        ({ data }) => {
          setTaskInfo(data.taskInfo);
        },
      );
    }
  }, [taskId, taskInfo, request]);

  // ----
}
```

< Home

Create a secure one-way hash for a text value (like a password) in Node

test

code node

+ Add New Approach

Approaches (1)

0

```
const bcrypt = require('bcrypt');
const hashedPass = bcrypt.hashSync('testPass123', 10);
```

test

EXPLANATION

The second argument to hashSync (or hash) is for the "salt" to be used to hash the text. When specified as a number then a salt will be generated with the specified number of rounds and used.

NOTE

To do the hashing asynchronously, use the `bycrypt.hash` method. It returns a promise.

Using GraphQL fragments in UI components

Table of Contents

Performing GraphQL mutation requests

```
// ·-·-·
const USER_LOGIN = `
  mutation userLogin($input: AuthInput!) {
    userLogin(input: $input) {
      errors {
        message
      }
      user {
        id
        username
      }
      authToken
    }
  }
`;
// ·-·-·
```

The login/signup forms

```
// .-.-.
const USER_CREATE = `
  mutation userCreate($input: UserInput!) {
    userCreate(input: $input) {
      errors {
        message
      }
      user {
        id
        username
      }
      authToken
    }
  }
`;
// .-.-.
```

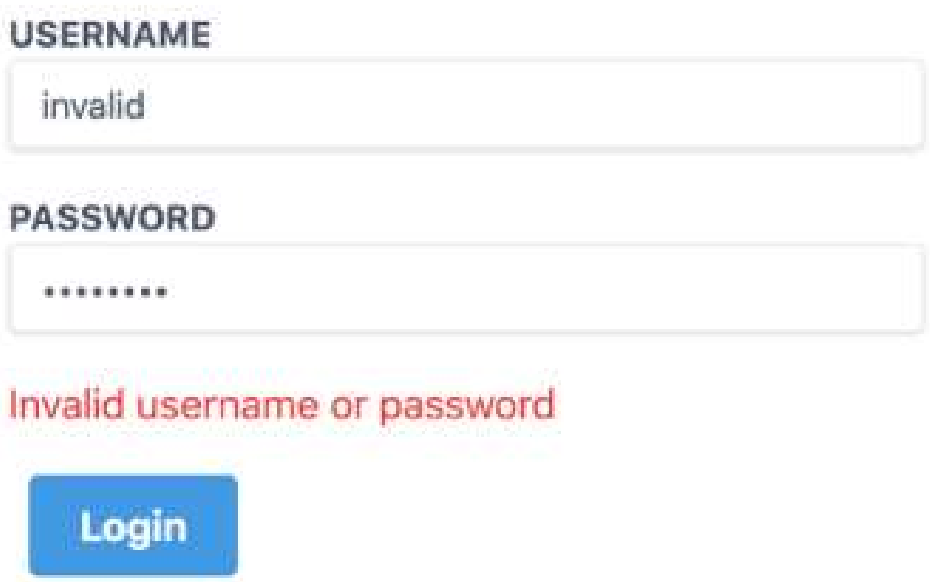
The login/signup forms

The login/signup forms

```
// ·-·-·

export default function Login() {
  // ·-·-·

  const handleLogin = async (event) => {
    event.preventDefault();
    const input = event.target.elements;
    const { data } = await request(USER_LOGIN, {
      variables: {
        input: {
          username: input.username.value,                    1
          password: input.password.value,
        },
      },
    });
    const { errors, user, authToken } = data.userLogin;
    if (errors.length > 0) {                                 2
      return setUIErrors(errors);
    }
    user.authToken = authToken;
    window.localStorage.setItem('azdev:user', JSON.stringify(user))
    setLocalAppState({ user, component: { name: 'Home' } });
  };

  // ·-·-·
}
```

The login/signup forms

- Test the Login form with invalid credentials.
- You should see the “Invalid username or password” message



The login/signup forms

- To test the Login form with valid credentials, use the test account in the sample data script (test/123456).
- You should be redirected to the home page, and the navigation bar should now display the user's username

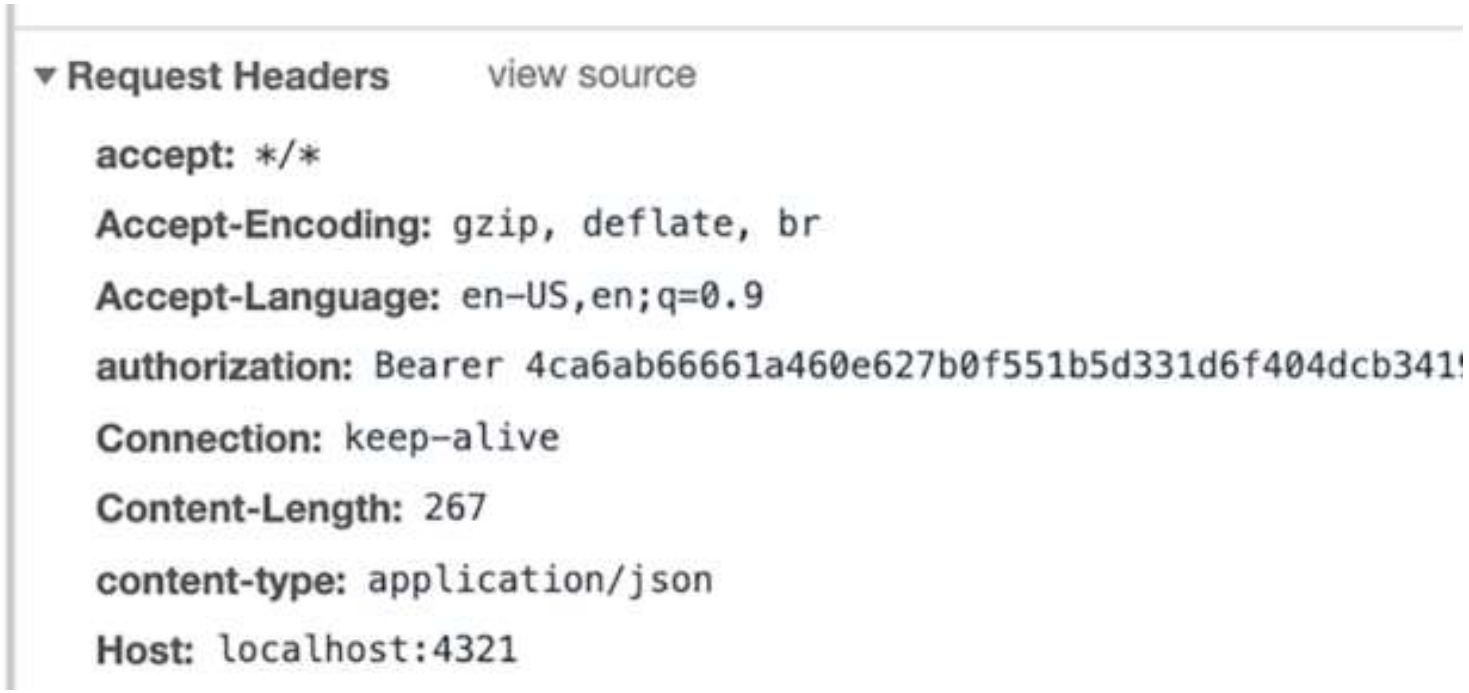
Implemented Of mutation call.

```
// ·-·-·

export default function Signup() {
  // ·-·-·

  const handleSignup = async (event) => {
    event.preventDefault();
    const input = event.target.elements;
    if (input.password.value !== input.confirmPassword.value) {
      return setUIErrors([{ message: 'Password mismatch' }]);
    }
    const { data } =
      await request(USER_CREATE, {
        variables: {
          input: {
            firstName: input.firstName.value,
            lastName: input.lastName.value,
            username: input.username.value,
            password: input.password.value,
          },
        },
      });
    const { errors, user, authToken } = data.userCreate;
    if (errors.length > 0) {
      return setUIErrors(errors);
    }
    user.authToken = authToken;
    window.localStorage.setItem('azdev:user', JSON.stringify(user))
    setLocalAppState({ user, component: { name: 'Home' } });
  };
  // ·-·-·
}
```

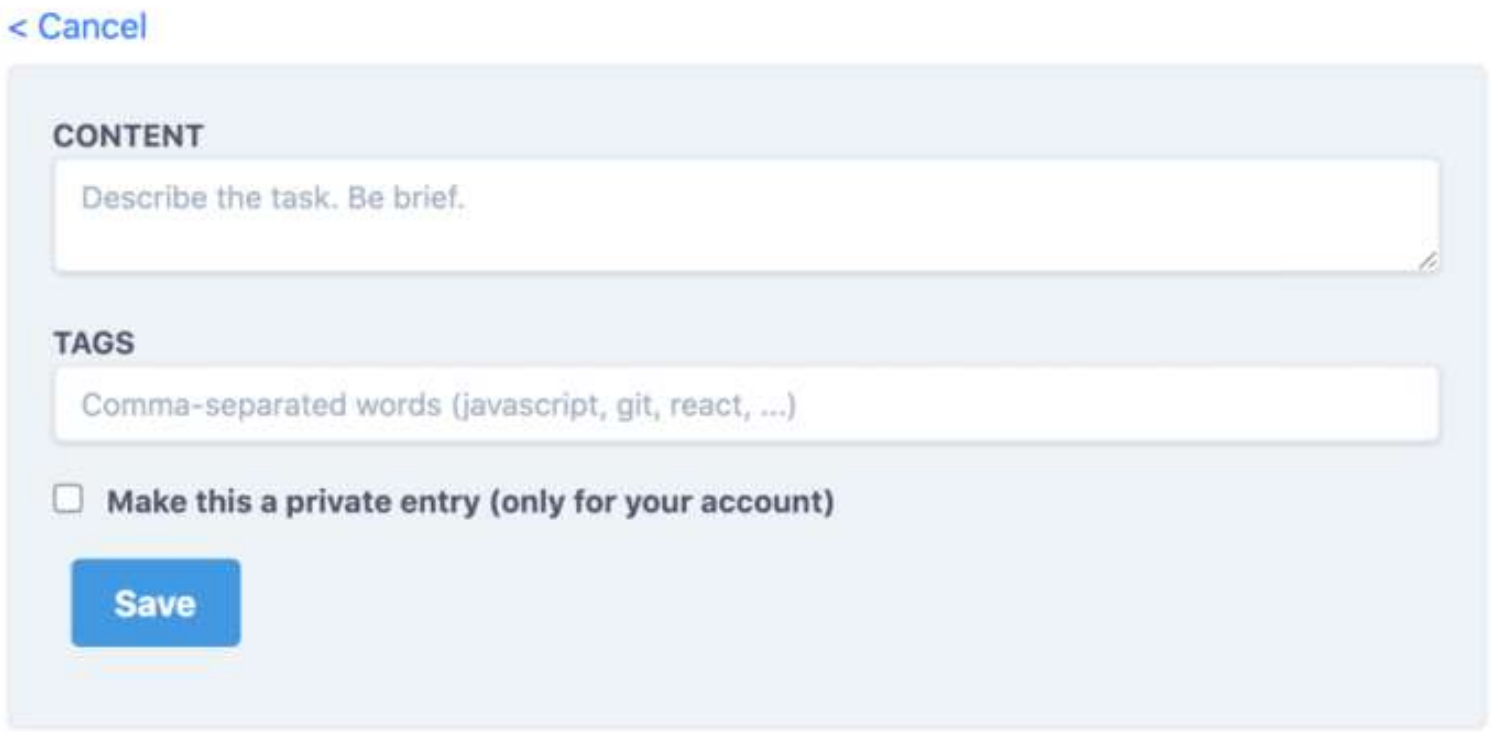
Authenticating GraphQL requests

```
const request = async (requestText, { variables } = {}) => {
  const headers = state.user
    ? { Authorization: 'Bearer ' + state.user.authToken }
    : {};
  const gsResp = await fetch(config.GRAPHQL_SERVER_URL, {
    method: 'post',
    headers: { ...headers, 'Content-Type': 'application/json'
},
    body: JSON.stringify({ query: requestText, variables }),
  }).then((response) => response.json());

  return gsResp;
};
```

Authenticating GraphQL requests



The Create Task form



The Create Task form

```
// ·-·-·
const TASK_CREATE = `
  mutation taskCreate($input: TaskInput!) {
    taskCreate(input: $input) {
      errors {
        message
      }
      task {
        id
      }
    }
  }
`;

export default function NewTask() {
  // ·-·-·

  const handleNewTaskSubmit = async (event) => {
    event.preventDefault();
    const input = event.target.elements;
    const { data, errors: rootErrors } = await request(TASK_CREATE,
      variables: {
        input: {
          content: input.content.value,
          tags: input.tags.value.split(','),
          isPrivate: input.private.checked,
        },
      },
```

Continued code

```
    if (rootErrors) {
      return setUIErrors(rootErrors);
    }
    const { errors, task } = data.taskCreate;
    if (errors.length > 0) {
      return setUIErrors(errors);
    }
    setLocalAppState({
      component: { name: 'TaskPage', props: { taskId: task.id } },
    });
  };

  // ----
}
```

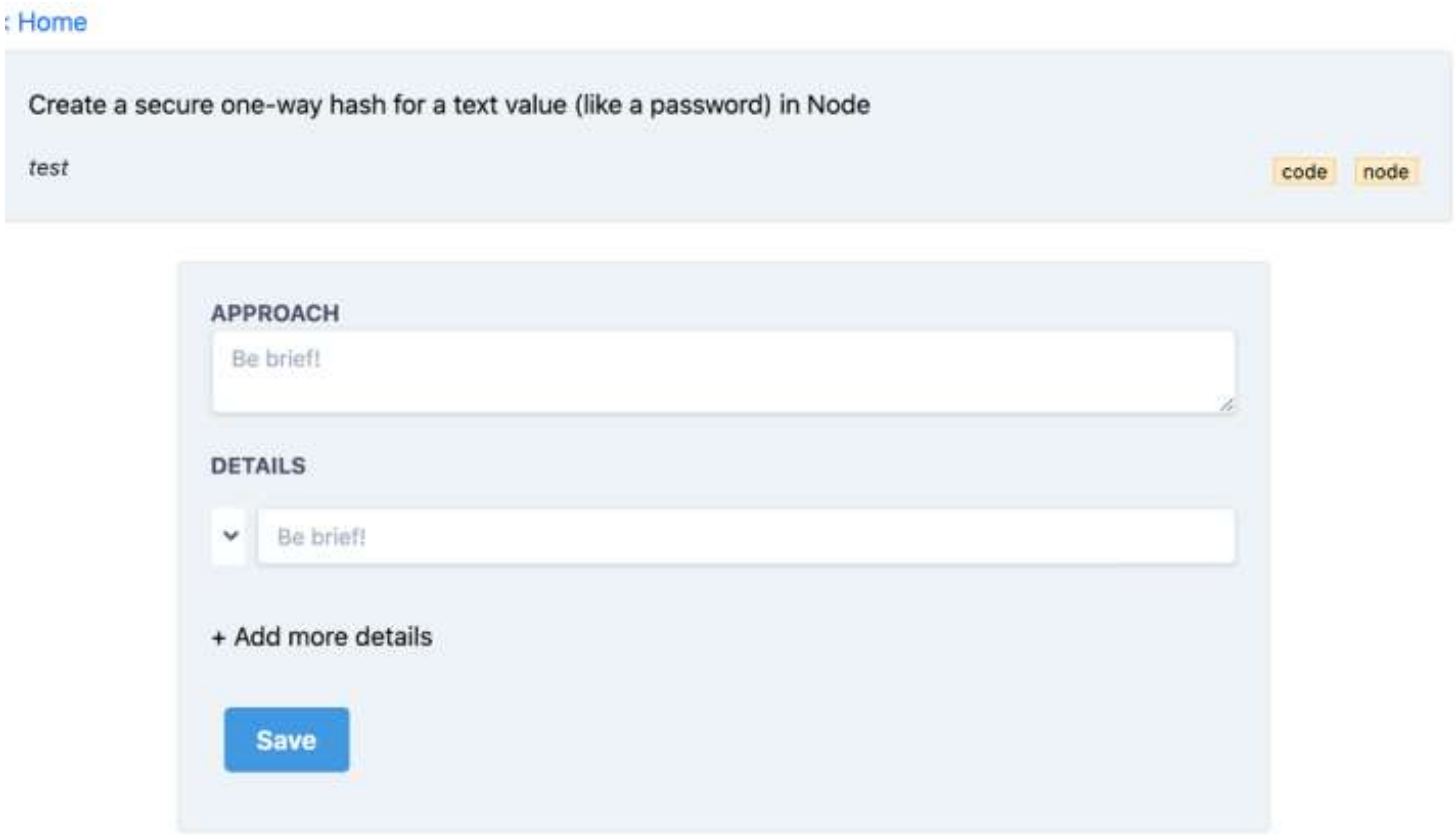
The Create Task form

```
// ----
export const FULL_TASK_FRAGMENT = `
  fragment FullTaskData on Task {
    id
    ...TaskSummary
    approachList {
      id
      ...ApproachFragment
    }
  }
  ${TASK_SUMMARY_FRAGMENT}
  ${APPROACH_FRAGMENT}
`;

const TASK_INFO = `
  query taskInfo($taskId: ID!) {
    taskInfo(id: $taskId) {
      ...FullTaskData
    }
  }
  ${FULL_TASK_FRAGMENT}
`;
// ----
```

```
// .-.-.
import { FULL_TASK_FRAGMENT } from './TaskPage';

const TASK_CREATE = `
  mutation taskCreate($input: TaskInput!) {
    taskCreate(input: $input) {
      errors {
        message
      }
      task {
        id
        ...FullTaskData
      }
    }
  }
  ${FULL_TASK_FRAGMENT}
`;
// .-.-.
```

The Create Task form

The Create Approach form

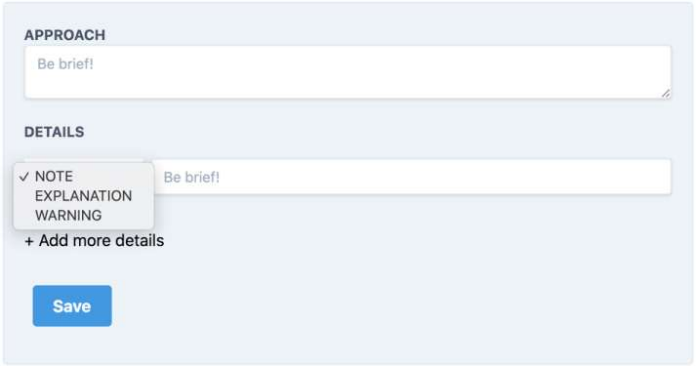


The Create Approach form

```
// .-.-.
const DETAIL_CATEGORIES = `
  query getDetailCategories {
    detailCategories: __type(name:
"ApproachDetailCategory") {
      enumValues {
        name
      }
    }
  }
`;
// .-.-.
```

The Create Approach form

```
// ·-·-·

export default function NewApproach({ taskId, onSuccess }) {
  // ·-·-·

  useEffect(() => {
    if (detailCategories.length === 0) {
      request(DETAIL_CATEGORIES).then(({ data }) => {
        setDetailCategories(data.detailCategories.enumValues);
      });
    }
  }, [detailCategories, request]);

  // ·-·-·
}
```



```
// ·-·-·
import { APPROACH_FRAGMENT } from './Approach';
// ·-·-·
const APPROACH_CREATE = `
  mutation approachCreate($taskId: ID!, $input:
ApproachInput!) {
    approachCreate(taskId: $taskId, input: $input) {
      errors {
        message
      }
      approach {
        id
        ...ApproachFragment
      }
    }
  }
  ${APPROACH_FRAGMENT}
`;
// ·-·-·
```

The Create Approach form

The Create Approach form

```
export default function NewApproach({ taskId, onSuccess }) {
  // ----

  const handleNewApproachSubmit = async (event) => {
    event.preventDefault();
    setUIErrors([]);
    const input = event.target.elements;
    const detailList = detailRows.map((detailId) => ({
      category: input[`detail-category-${detailId}`].value,
      content: input[`detail-content-${detailId}`].value,
    }));
    const { data, errors: rootErrors } = await request(
      APPROACH_CREATE,
      {
        variables: {
          taskId,
          input: {
            content: input.content.value,
            detailList,
          },
        },
      },
    );
    if (rootErrors) {
      return setUIErrors(rootErrors);
    }
    const { errors, approach } = data.approachCreate;
    if (errors.length > 0) {
      return setUIErrors(errors);
    }
    onSuccess(approach);
  };
```

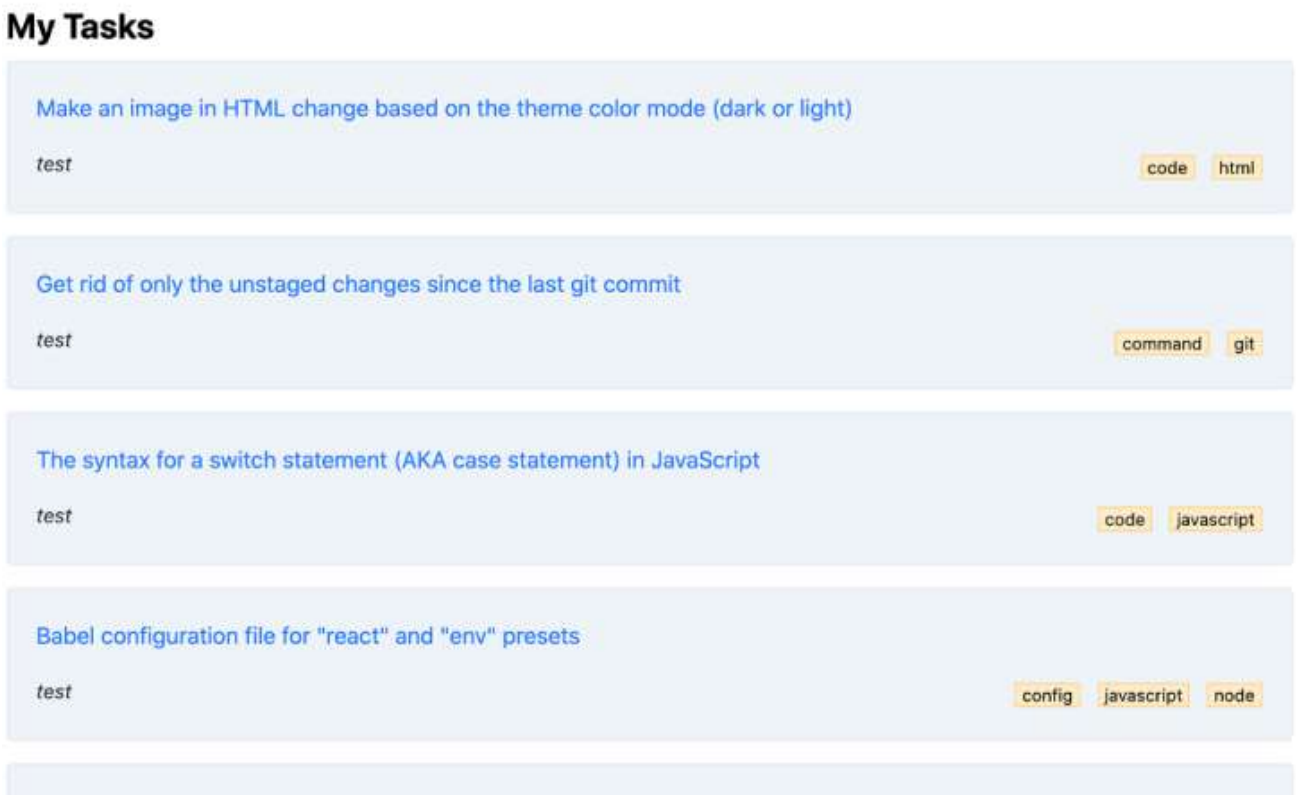
The Create Approach form

Table of Contents

```
// ·-·-·

import TaskSummary, { TASK_SUMMARY_FRAGMENT } from './TaskSummary';

const MY_TASK_LIST = `
  query myTaskList {
    me {
      taskList {
        id
        ...TaskSummary
      }
    }
  }
  ${TASK_SUMMARY_FRAGMENT}
`;

export default function MyTasks() {
  // ·-·-·

  useEffect(() => {
    request(MY_TASK_LIST).then(({ data }) => {
      setMyTaskList(data.me.taskList);
    });
  }, [request]);

  // ·-·-·
```

PERFORMING QUERY REQUESTS SCOPED FOR A USER

PERFORMING QUERY REQUESTS SCOPED FOR A USER



The Search form

```
<h2>Search Results</h2>
<div className="y-spaced">
  {searchResults.length === 0 && (
    <div className="box box-primary">No results</div>
  )}
  {searchResults.map((item, index) => (
    <div key={index} className="box box-primary">
      <AppLink
        to="TaskPage"
        taskId={
          item.type === 'Approach' ? item.task.id : item.id
        }
      >
        <span className="search-label">{item.type}</span>{' '}
        {item.content.substr(0, 250)}
      </AppLink>
      <div className="search-sub-line">
        {item.type === 'Task'
          ? `Approaches: ${item.approachCount}`
          : `Task: ${item.task.content.substr(0, 250)}`}
      </div>
    </div>
  ))}
</div>
```

```
// ----
const SEARCH_RESULTS = `
  query searchResults($searchTerm: String!) {
    searchResults: search(term: $searchTerm) {
      type: __typename
      id
      content
      ... on Task {
        approachCount
      }
      ... on Approach {
        task {
          id
          content
        }
      }
    }
  }
`;
// ----
```

The Search form

```
// ·-·-·

export default function Search({ searchTerm = null }) {
  // ·-·-·

  useEffect(() => {
    if (searchTerm) {
      request(SEARCH_RESULTS, { variables: { searchTerm } }).then(
        ({ data }) => {
          setSearchResults(data.searchResults);
        },
      );
    }
  }, [searchTerm, request]);

  // ·-·-·
}
```

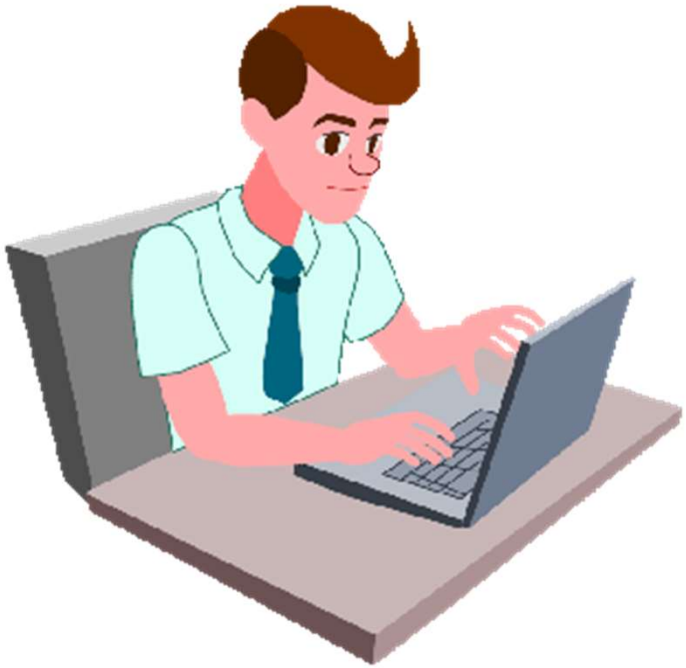
How I invoked the searchResults query.


babel
Search
Search Results
No results
< Home
babel
Search
Search Results
Task
Babel configuration file for "react" and "env" presets
Approaches: 1
< Home

SEARCH FORM

Summary

- Components can define operations and fragments.
- An operation can be a query, a mutation, or a subscription.
- A component query operation is generally used to display that component.
- A mutation operation is usually used within a DOM event handler (like onSubmit or onClick)



"Complete Lab"