Using GraphQL APIs without a client library

Using GraphQL APIs without a client library

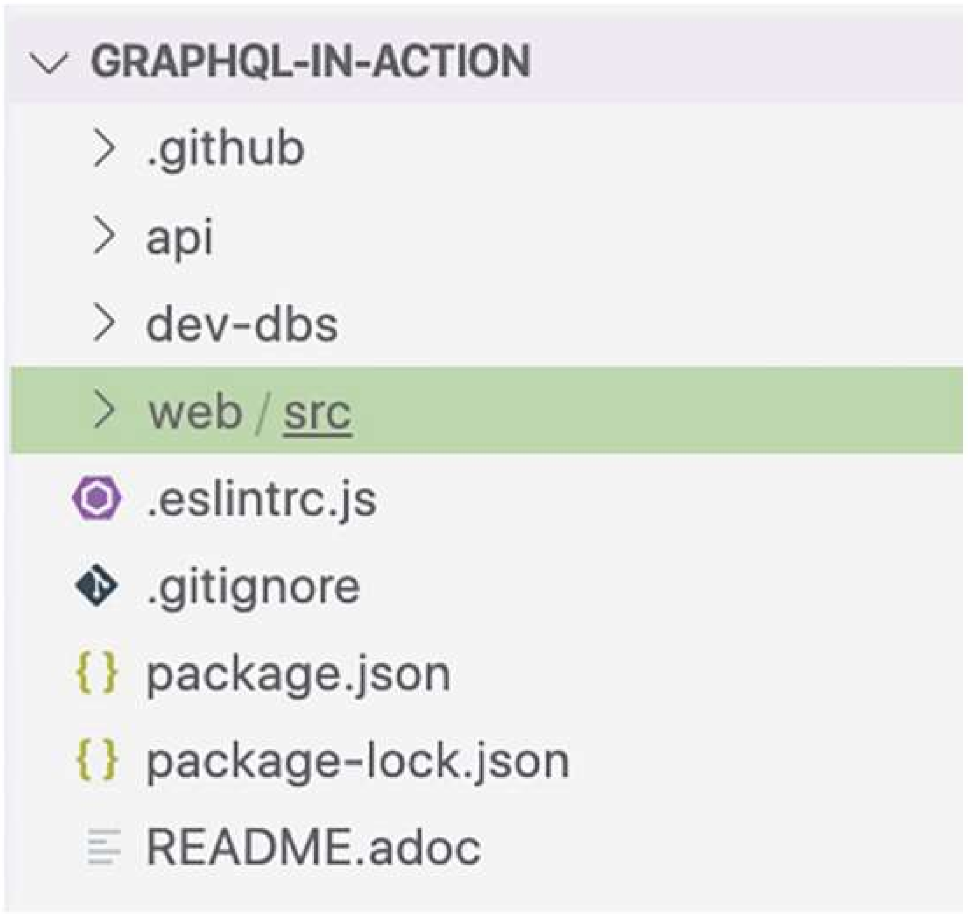
This lesson covers

- Analyzing UI components to determine their GraphQL data requirements
- Performing Ajax POST requests for GraphQL operations
- Using tokens in request headers to identify the API consumer
- Using fragments to make every UI component responsible for its own data requirements
- Using GraphQL features to simplify and generalize the UI code

Using a web UI library

- I contemplated demonstrating how to use GraphQL without using a UI library (like React or Angular).
- While that would be a good exercise to learn the DOM API, it’s really not practical.
- For many reasons (beyond the scope of this course), no one builds frontend applications today without a UI library, It’s simply messy.
- UI libraries decrease the complexities of dealing with the DOM API and offer a declarative alternative to it.

Running the web server

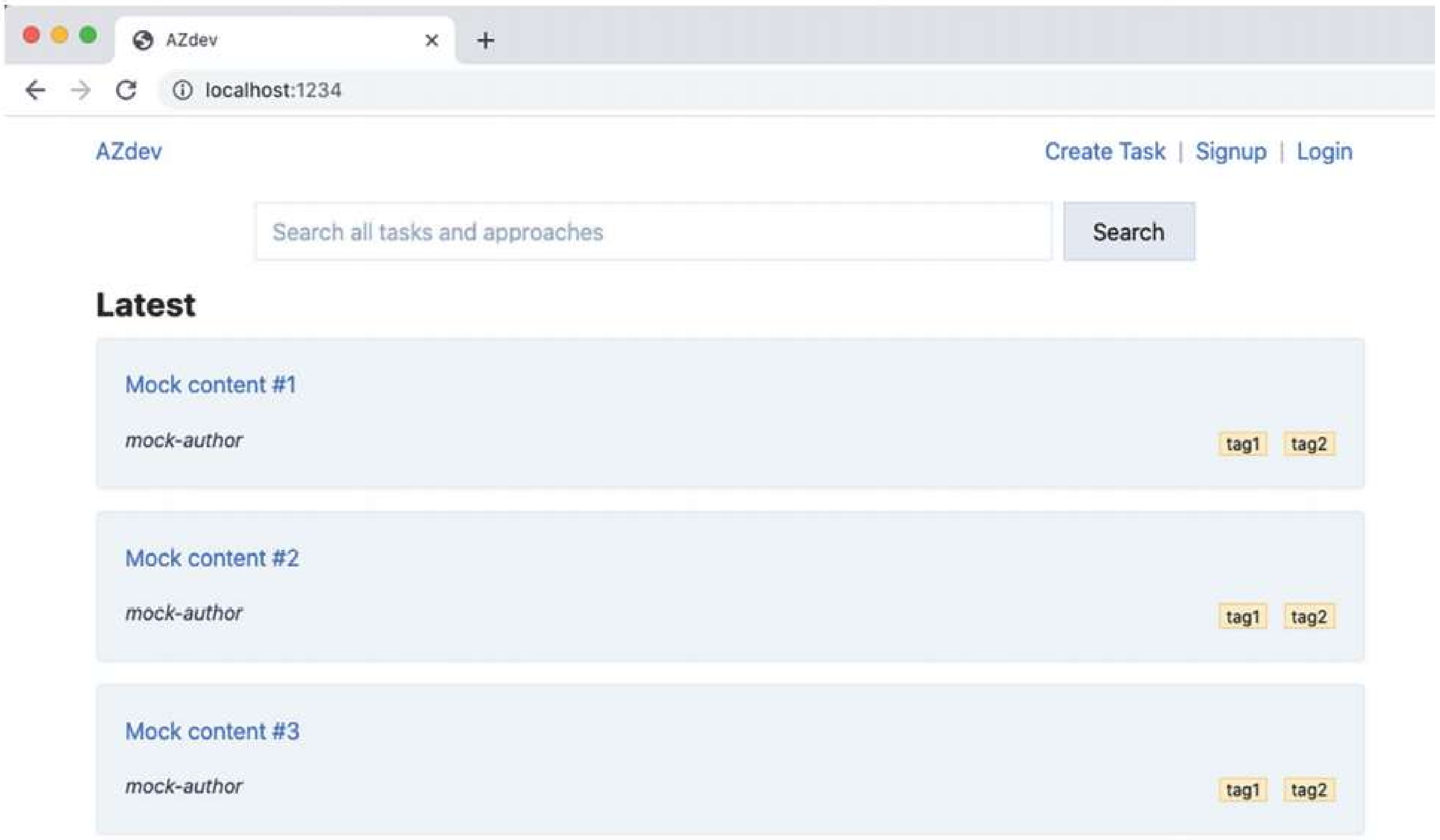


```
GRAPHQL-IN-ACTION
  > .github
  > api
  > dev-dbs
  > web / src
  .eslintrc.js
  .gitignore
  package.json
  package-lock.json
  README.adoc
```

Running the web server

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

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



AZdev   Create Task | Signup | Login

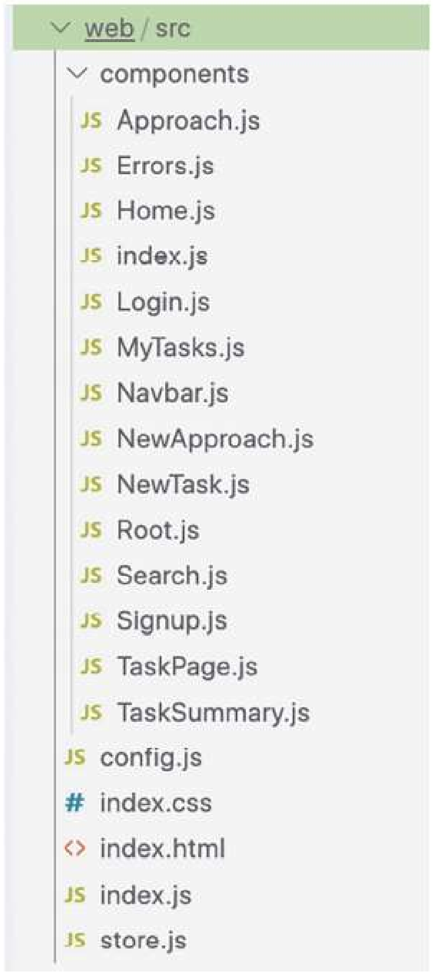
Search all tasks and approaches   Search

Latest

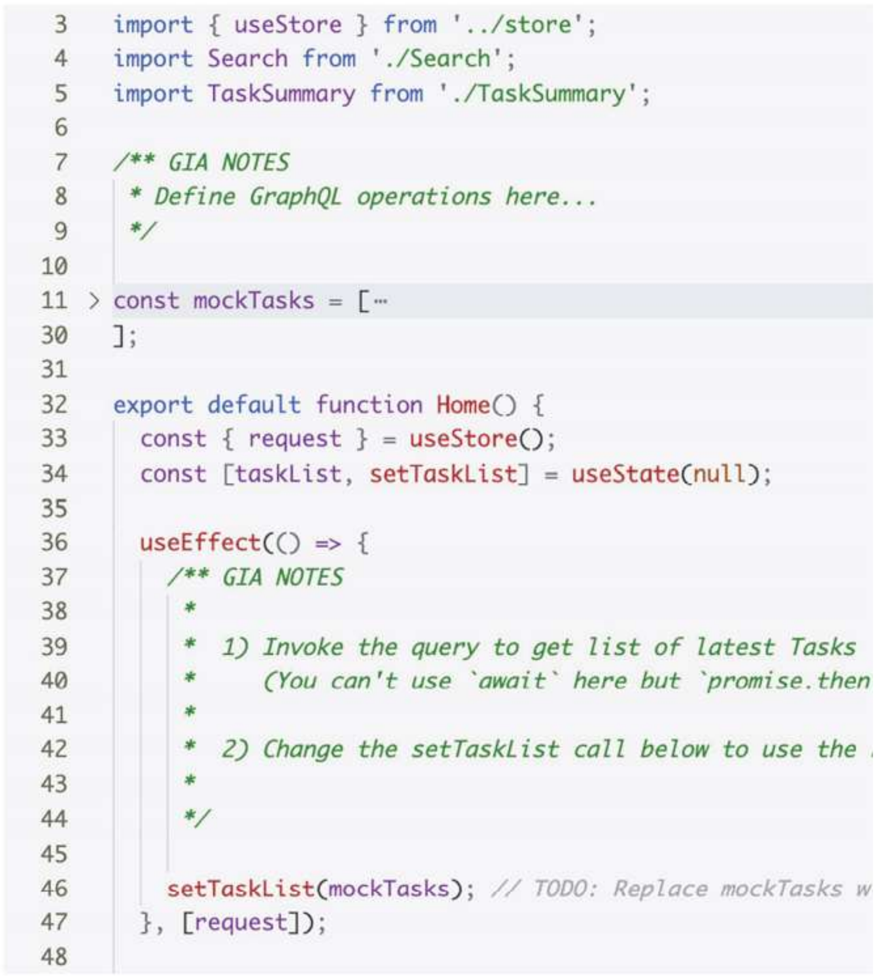
Mock content #1
mock-author   tag1 tag2

Mock content #2
mock-author   tag1 tag2

Mock content #3
mock-author   tag1 tag2



```
web / src
  components
    Approach.js
    Errors.js
    Home.js
    index.js
    Login.js
    MyTasks.js
    Navbar.js
    NewApproach.js
    NewTask.js
    Root.js
    Search.js
    Signup.js
    TaskPage.js
    TaskSummary.js
  config.js
  index.css
  index.html
  index.js
  store.js
```



```
import { useStore } from '../store';
import Search from './Search';
import TaskSummary from './TaskSummary';

/** GIA NOTES
 * Define GraphQL operations here...
 */

const mockTasks = [⋯
];

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

  useEffect(() => {
    /** GIA NOTES
     *
     *  1) Invoke the query to get list of latest Tasks
     *     (You can't use `await` here but `promise.then
     *
     *  2) Change the setTaskList call below to use the
     *
     */

    setTaskList(mockTasks); // TODO: Replace mockTasks w
  }, [request]);

```

Making Ajax requests

- To make a GraphQL request from a web application, we need to invoke an Ajax HTTP call.
- Remember that we made our GraphQL service available through HTTP POST requests.
- Up to this point, we’ve been sending these POST requests through the GraphiQL editor.
- Under the hood, the GraphiQL editor issues Ajax calls in the background when you execute any document there, Modern browsers have native APIs to make Ajax requests.

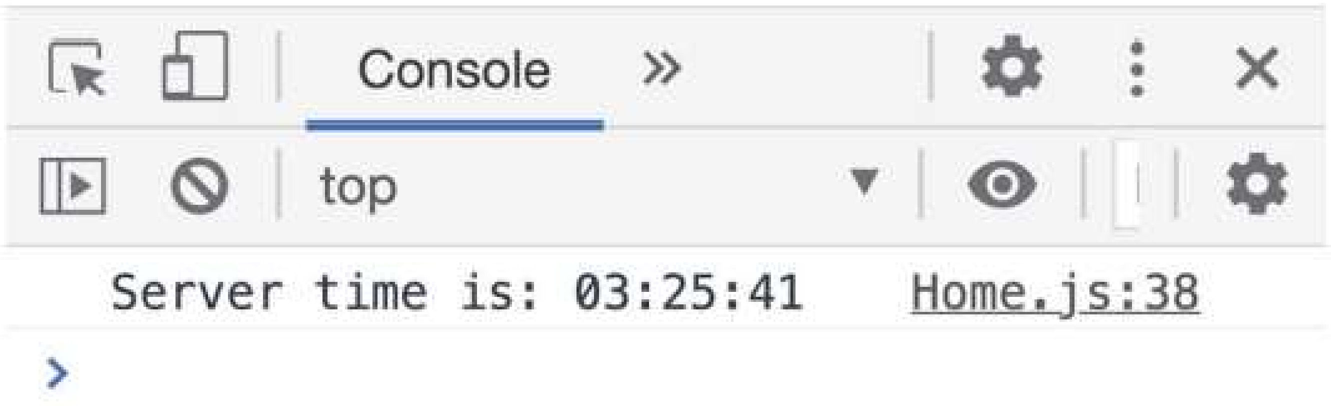
Making Ajax requests

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

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

    // -----
  }, [request]);

  return (
    // -----
  );
}
```

```
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

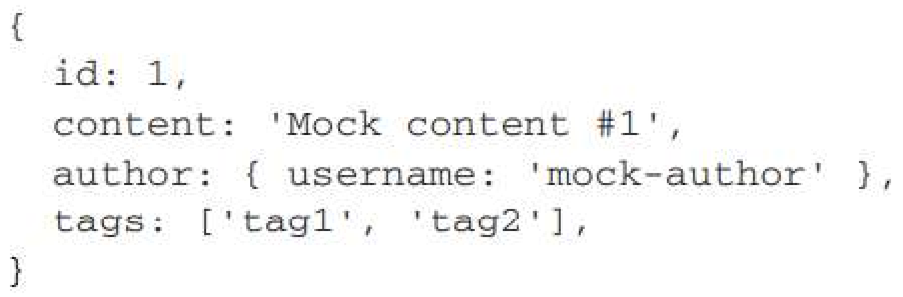
- You should now see the server time log message in your browser console



```
Server time is: 03:25:41    Home.js:38
```

Performing GraphQL query requests

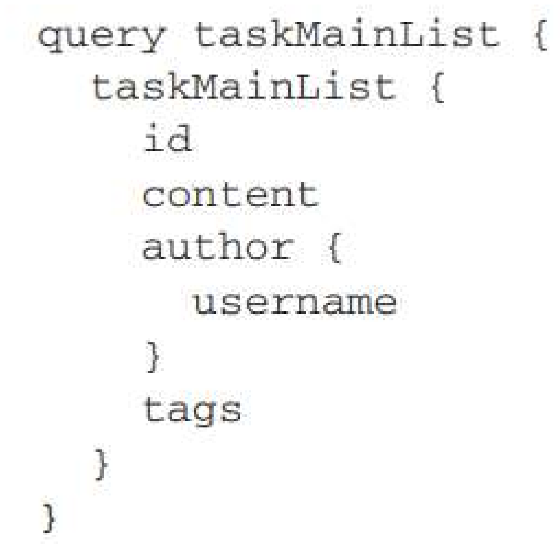
- You can look at the code in web/src/components/Home.js to see what subfields must be included in this taskMainList query.
- The structure of the data required for this view is the same as that for the mockTasks object



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

Performing GraphQL query requests

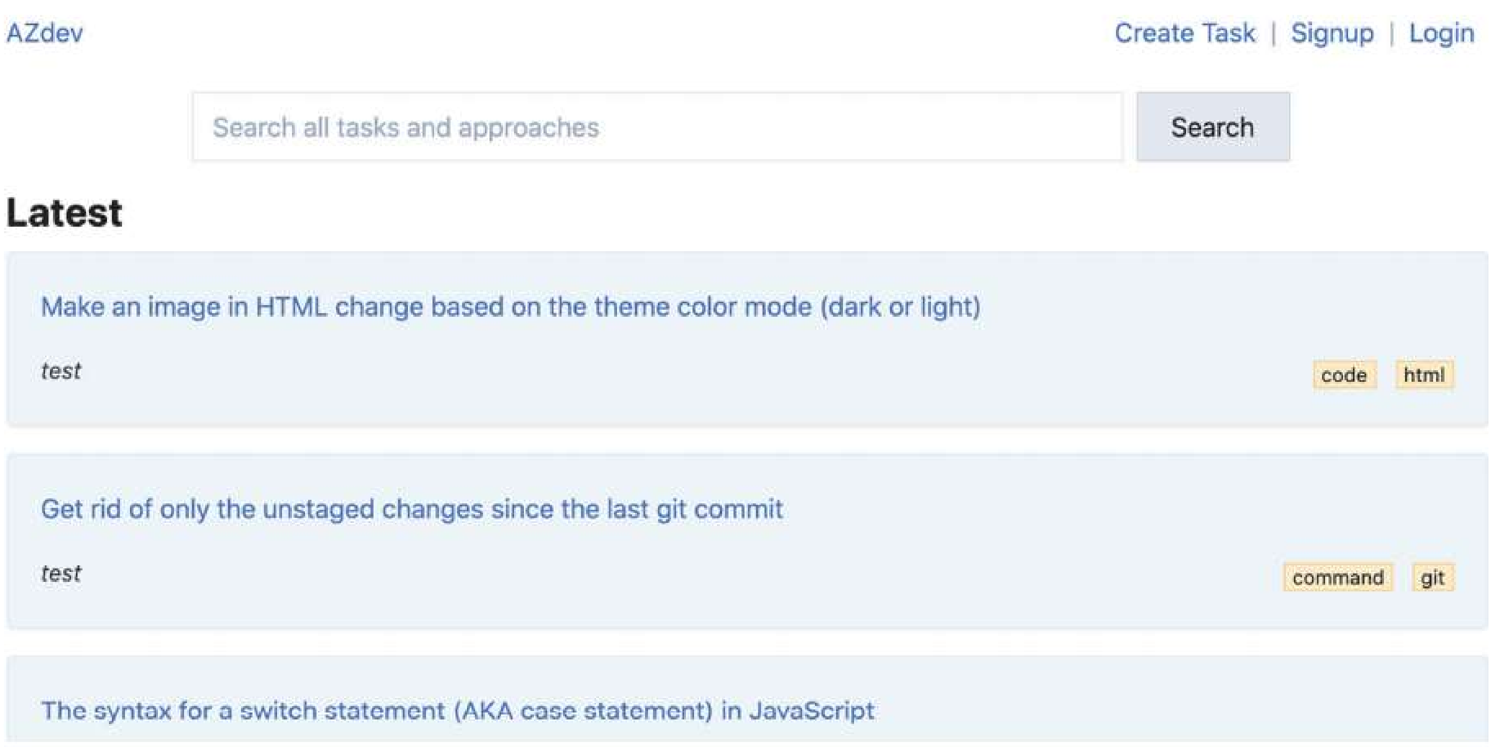
- Relying on the structure of that object, here's the GraphQL query required by the HOME component.



```
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



AZdev

Create Task | Signup | Login

Search all tasks and approaches

Search

Latest

Make an image in HTML change based on the theme color mode (dark or light)

test

code html

Get rid of only the unstaged changes since the last git commit

test

command git

The syntax for a switch statement (AKA case statement) in JavaScript

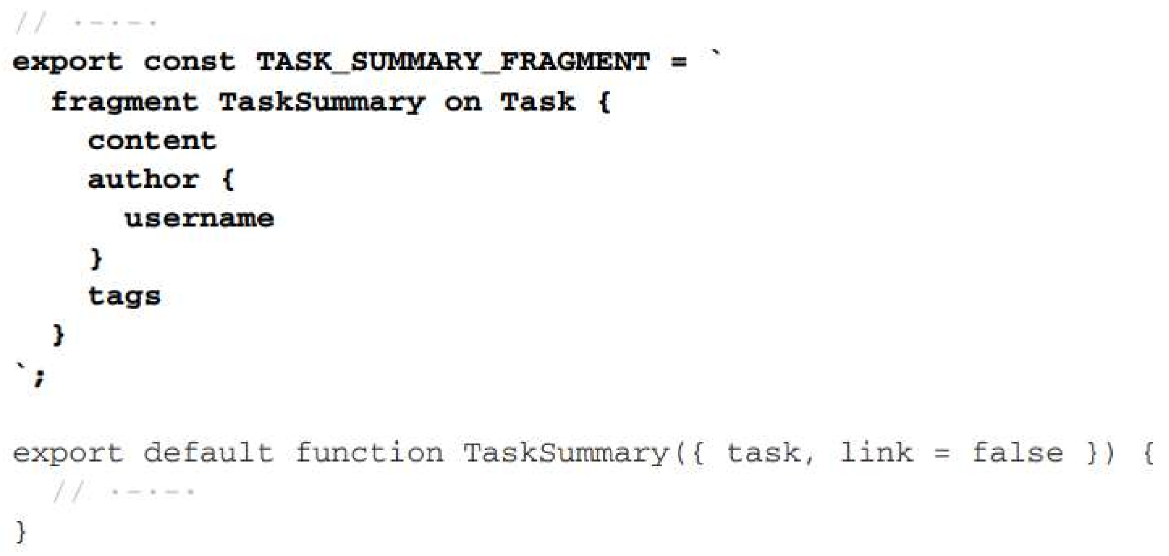
Using GraphQL fragments in UI components

- The mockTasks object in the Home component helps us easily figure out what GraphQL fields are required for that component.
- But what if we did not have that object? Also, what if we did not have the UI component? Is it better to start with a UI component or with a GraphQL query text?
- The concept that helps me wrap my thoughts around these questions is one that we briefly touched on in previous lessons, but it's now time to bring it to the forefront.

```
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>
  );
}
```

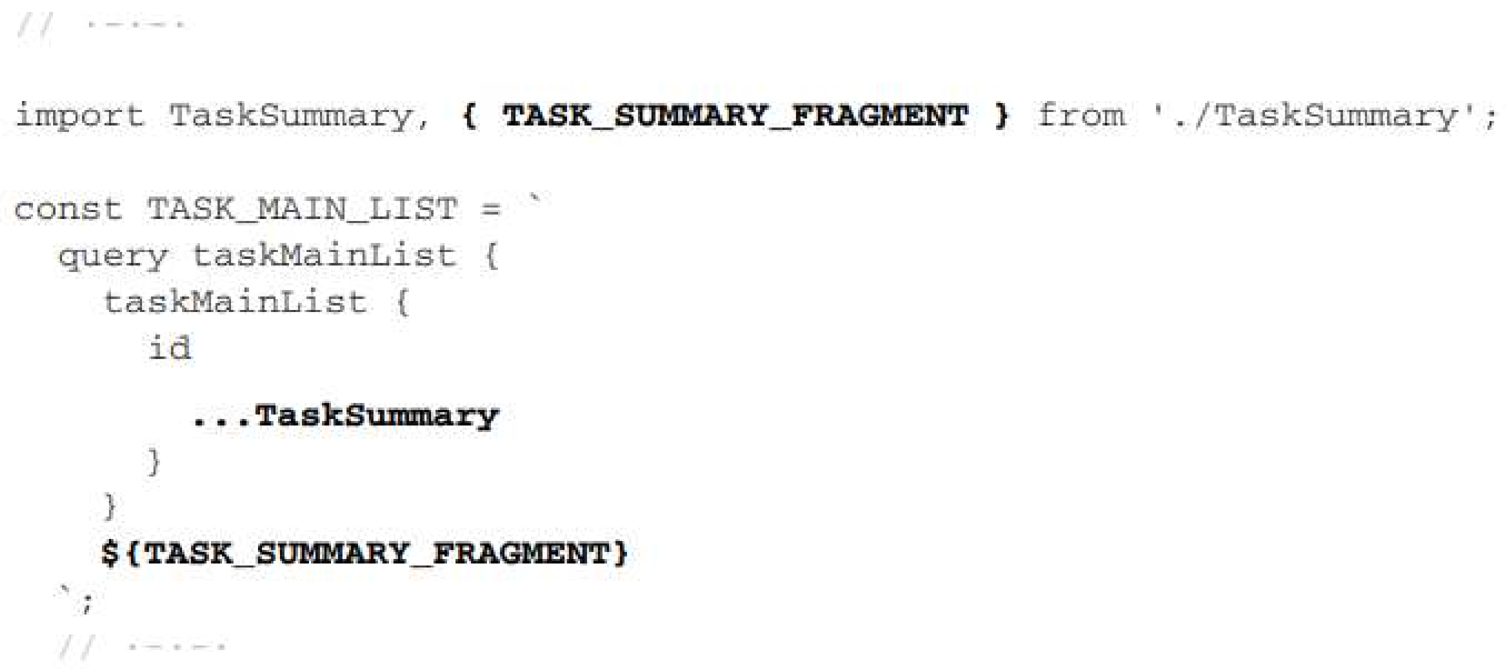
Using GraphQL fragments in UI components

- For example, we can define the following GraphQL fragment in the TaskSummary component.



```
// ...
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}
  `;
  // ...
```

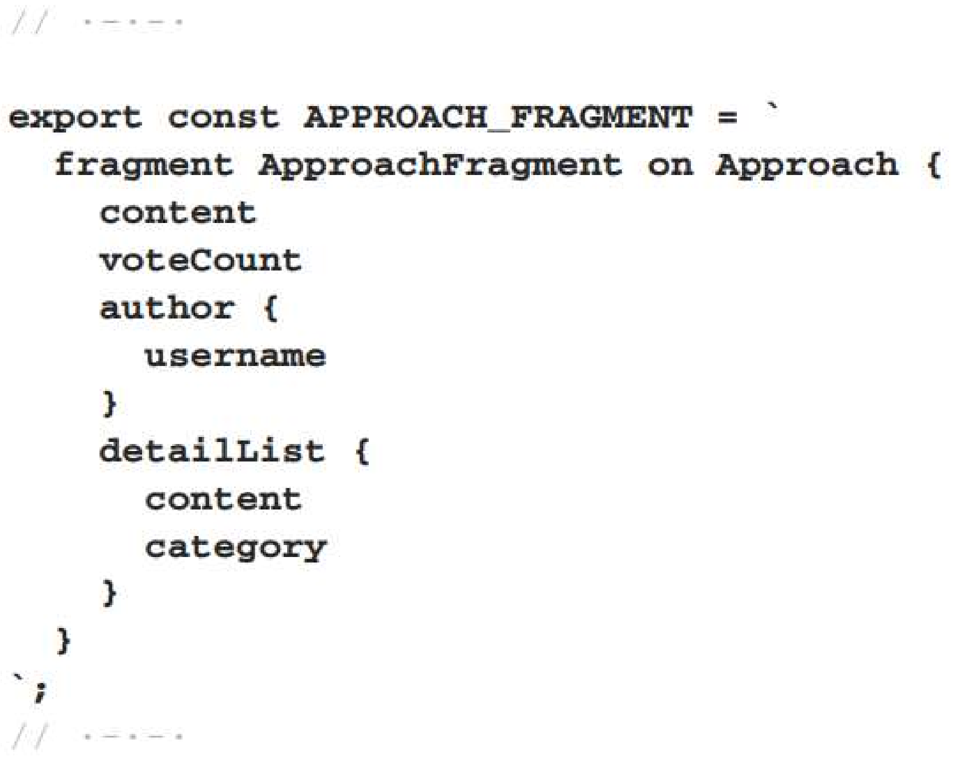
Including variables in requests

- The home page has a list of the latest Task records, The user can click a Task record to see its Detail page.
- The navigation is already implemented, but the data requirement for the page is not.
- Currently, when you navigate to a Task record page, you see mocked data.
- We can display this page with the partial data that we’ve already fetched for a Task record through the taskMainList request (and then fetch the rest of the required data, like the list of Approaches). .

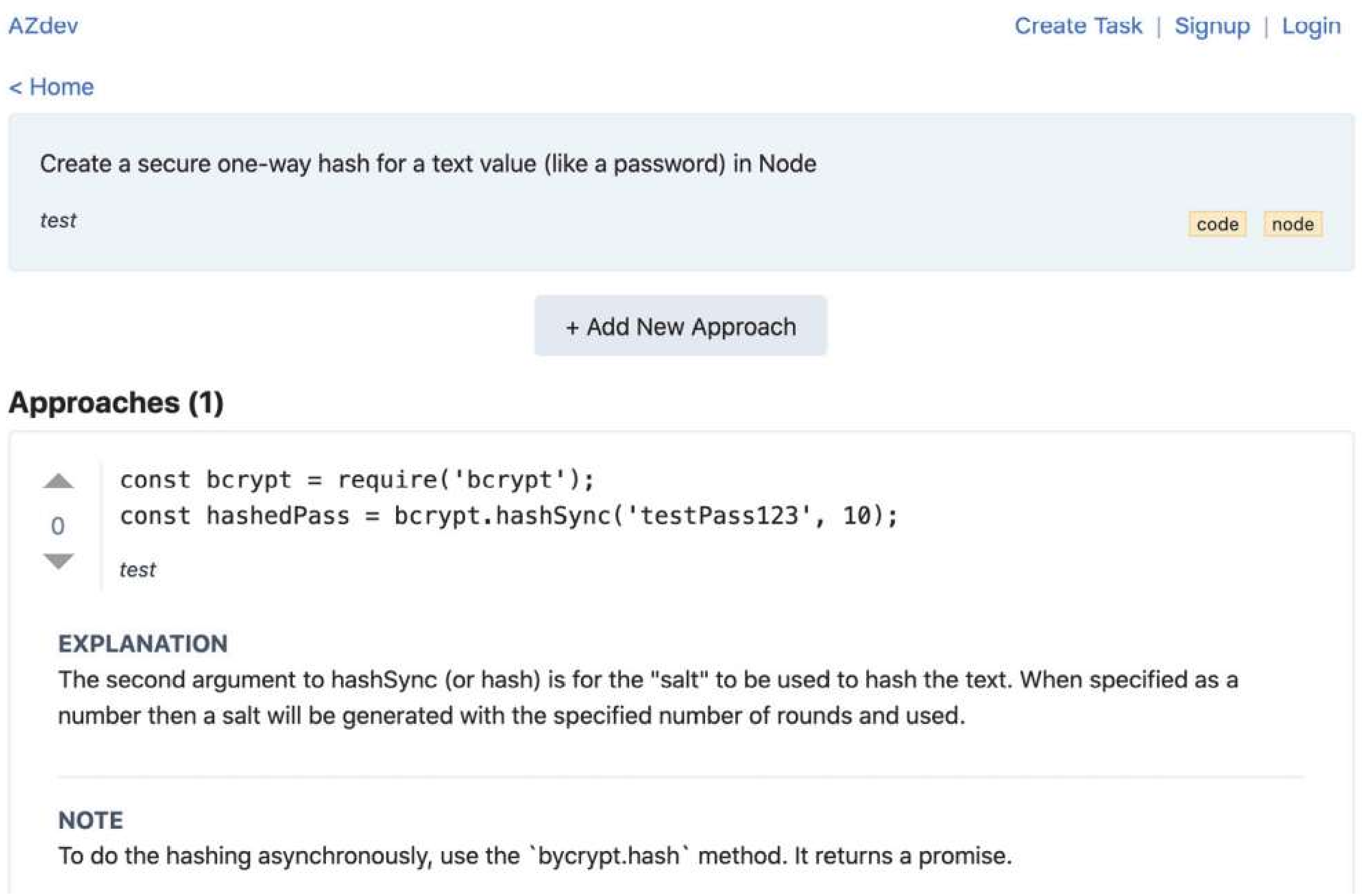
Including variables in requests

```
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>
  );
}
```

Including variables in requests



```
// ...

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

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}
`;
// ·-·-·
// 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]);

  // ·-·-·
}
```



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

test

Approaches (1)

```
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.

Performing GraphQL mutation requests

- A GraphQL mutation is usually invoked as part of an onSubmit event in an HTML form.
- The mutation input is generally read from the HTML form elements (text input, select value, check box, and so on).
- We have five basic forms in this project: login, signup, create Task, add an Approach to a Task, and search all Tasks and Approaches.

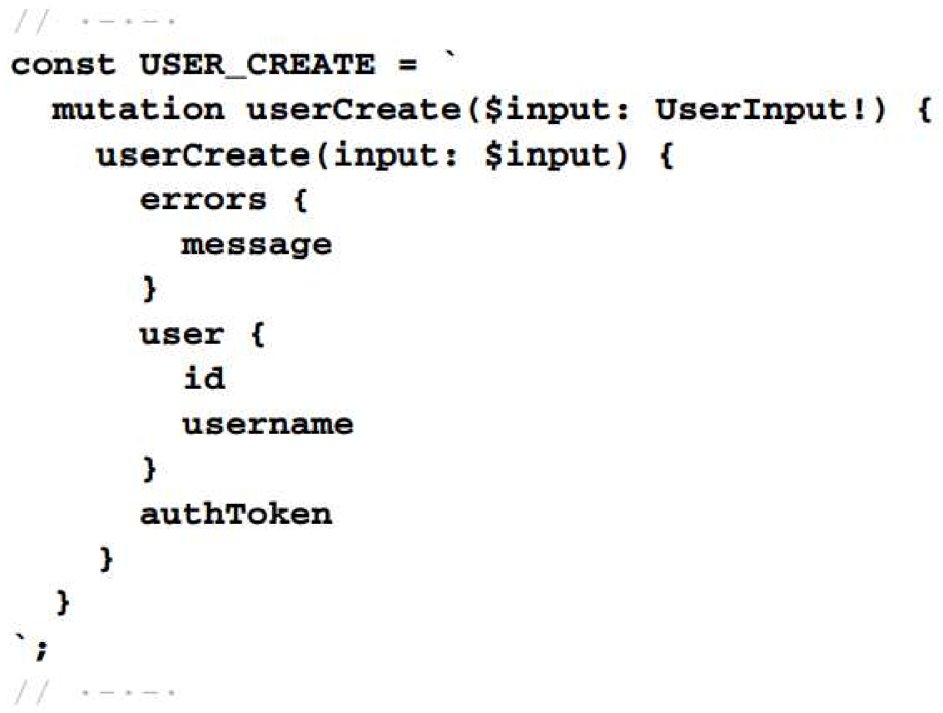
The login/signup forms

You can get to the login/signup forms using the links in the top-right corner of the home page. The components used for these pages are as follows:

- The Login component is under web/src/components/Login.js.
- The Signup component is under web/src/components/Signup.js.

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

The login/signup forms



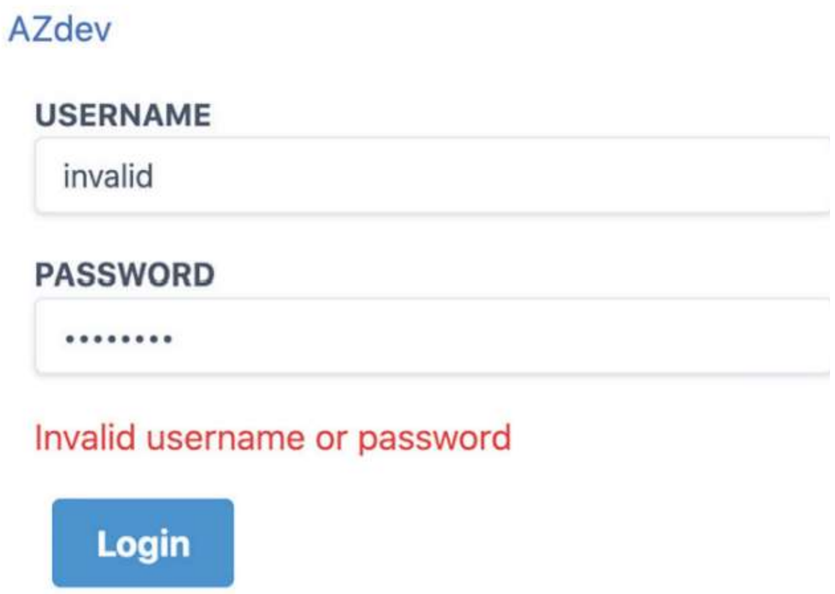
```
// ·-·-·
const USER_CREATE = `
  mutation userCreate($input: UserInput!) {
    userCreate(input: $input) {
      errors {
        message
      }
      user {
        id
        username
      }
      authToken
    }
  }
`;
// ·-·-·
```

The login/signup forms

```
// ·-·-·

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,
            password: input.password.value,
          },
        },
      });
      const { errors, user, authToken } = data.userLogin;
      if (errors.length > 0) {
        return setUIErrors(errors);
      }
      user.authToken = authToken;
      window.localStorage.setItem('azdev:user', JSON.stringify(user));
      setLocalAppState({ user, component: { name: 'Home' } });
    };

    // ·-·-·
}
```

The input data is read with a DOM API call. The state of this form is not controlled with React.

Checks for the existence of any user errors after the mutation, and sets these errors for the UI to display somewhere. This part of the code will display the “Invalid username or password” error message when you test the form with invalid credentials.


AZdev

USERNAME

invalid

PASSWORD

•••••••••

Invalid username or password

Login

Figure **`UserError` example for this mutation**

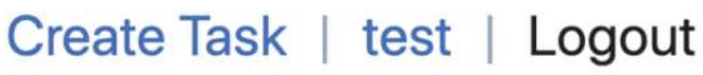
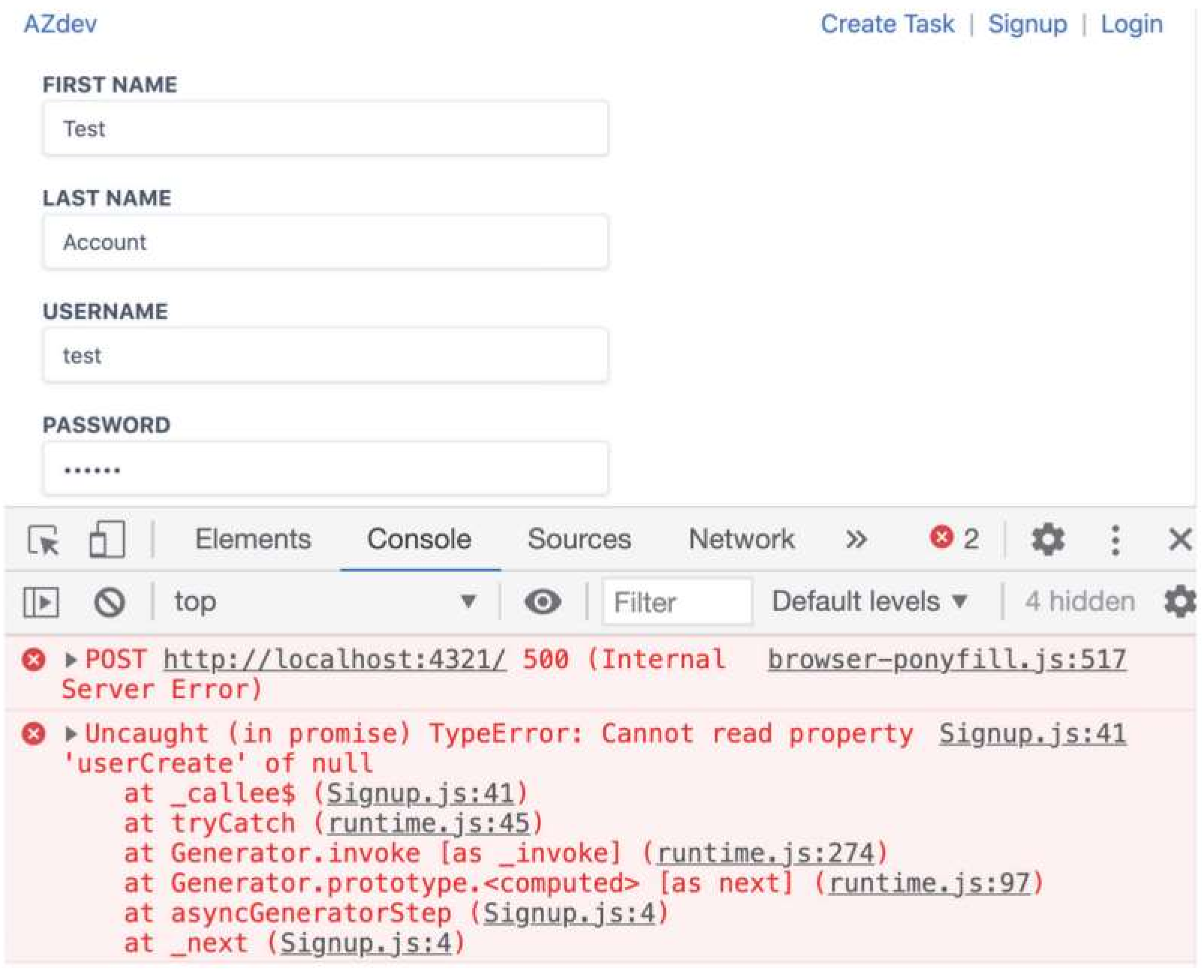

Create Task | test | Logout

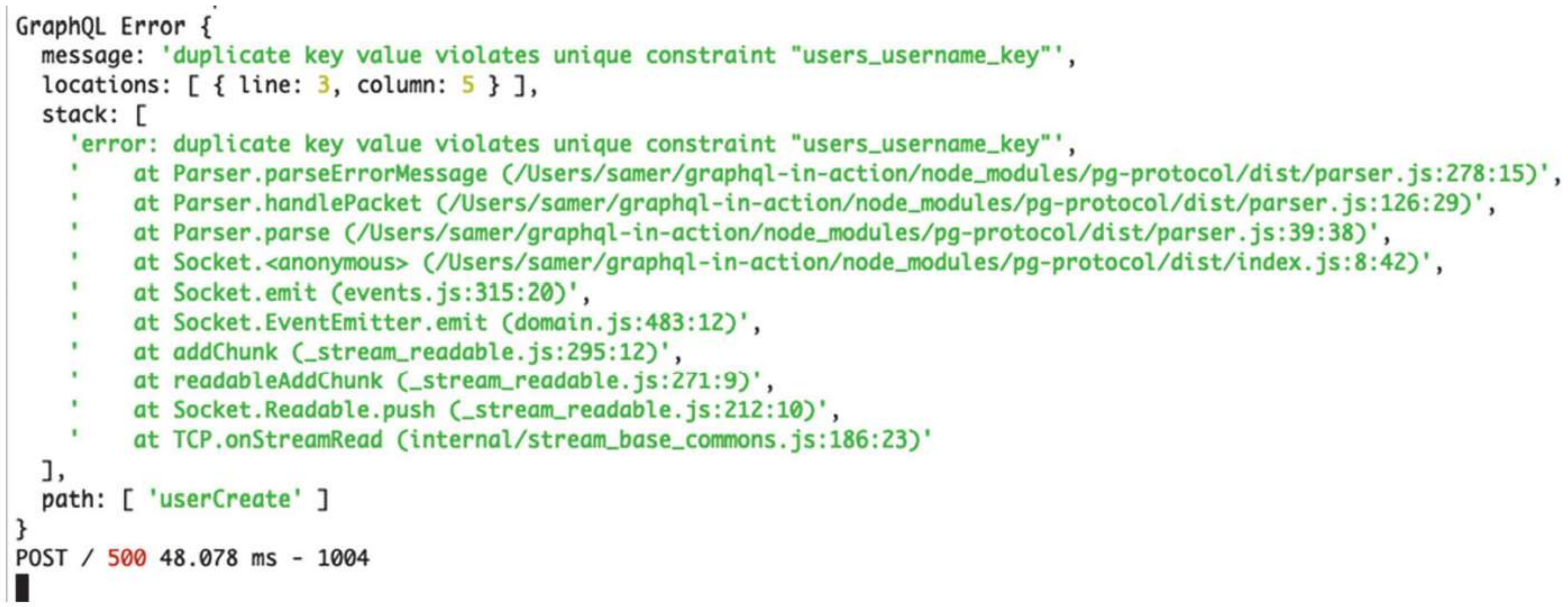
Figure **Navbar link for a logged-in user**

```
// ·-·-·

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' } });
  };
  // ·-·-·
}
```



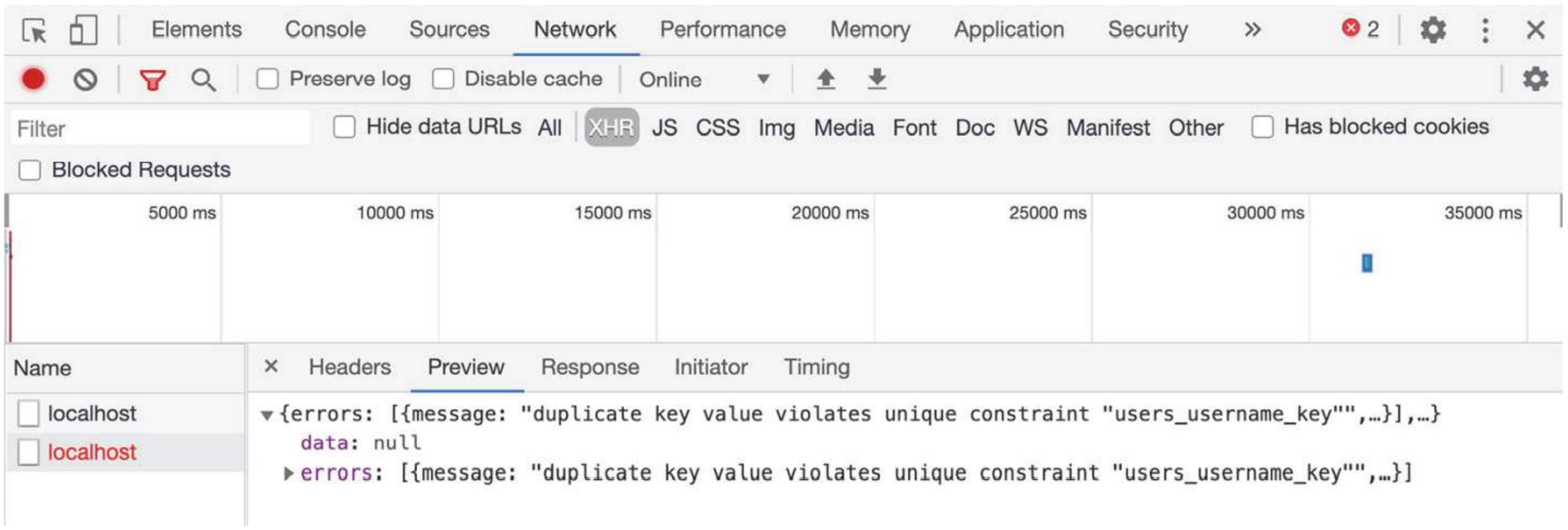
```
POST http://localhost:4321/ 500 (Internal   browser-ponyfill.js:517
Server Error)

Uncaught (in promise) TypeError: Cannot read property  Signup.js:41
'userCreate' of null
    at _callee$ (Signup.js:41)
    at tryCatch (runtime.js:45)
    at Generator.invoke [as _invoke] (runtime.js:274)
    at Generator.prototype.<computed> [as next] (runtime.js:97)
    at asyncGeneratorStep (Signup.js:4)
    at _next (Signup.js:4)
```

Handling generic server errors



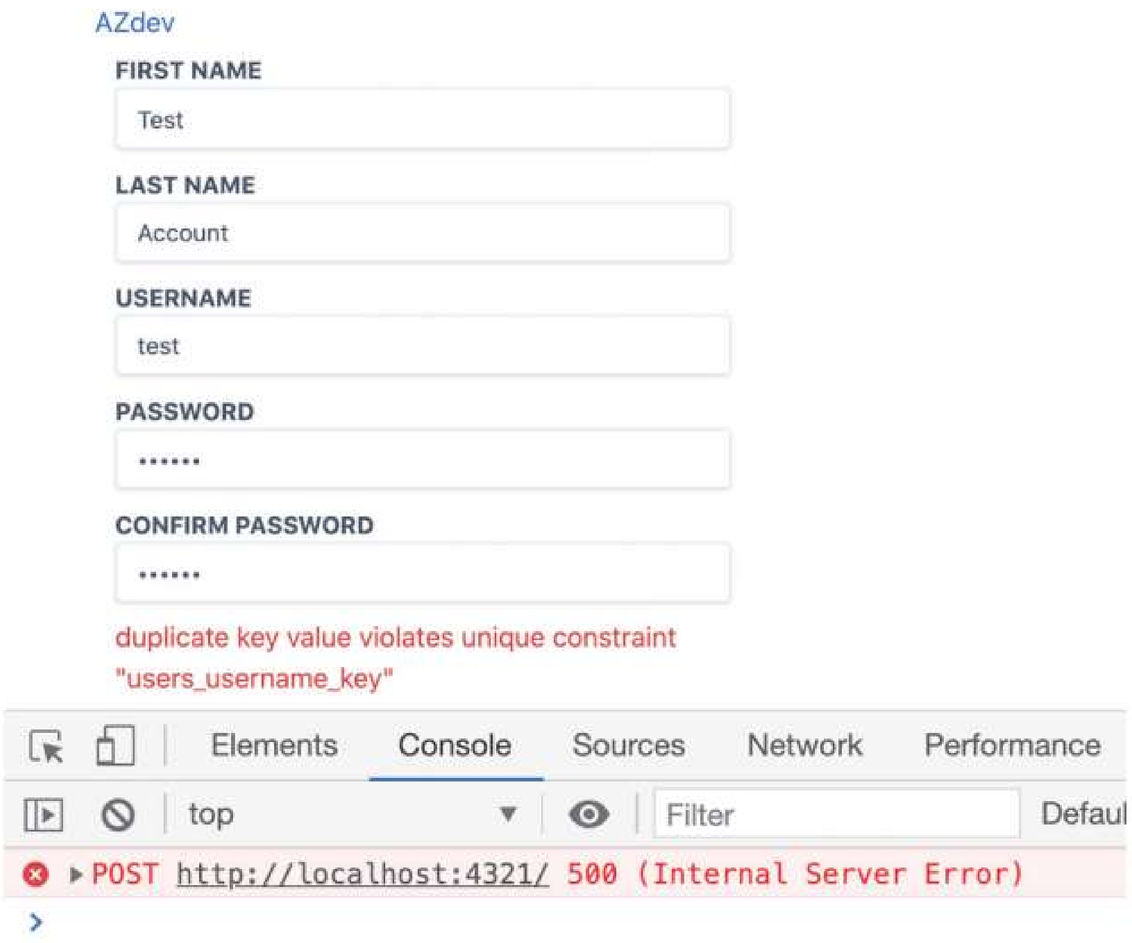
```
GraphQL Error {
  message: 'duplicate key value violates unique constraint "users_username_key"',
  locations: [ { line: 3, column: 5 } ],
  stack: [
    'error: duplicate key value violates unique constraint "users_username_key"',
    '    at Parser.parseErrorMessage (/Users/samer/graphql-in-action/node_modules/pg-protocol/dist/parser.js:278:15)',
    '    at Parser.handlePacket (/Users/samer/graphql-in-action/node_modules/pg-protocol/dist/parser.js:126:29)',
    '    at Parser.parse (/Users/samer/graphql-in-action/node_modules/pg-protocol/dist/parser.js:39:38)',
    '    at Socket.<anonymous> (/Users/samer/graphql-in-action/node_modules/pg-protocol/dist/index.js:8:42)',
    '    at Socket.emit (events.js:315:20)',
    '    at Socket.EventEmitter.emit (domain.js:483:12)',
    '    at addChunk (_stream_readable.js:295:12)',
    '    at readableAddChunk (_stream_readable.js:271:9)',
    '    at Socket.Readable.push (_stream_readable.js:212:10)',
    '    at TCP.onStreamRead (internal/stream_base_commons.js:186:23)'
  ],
  path: [ 'userCreate' ]
}
POST / 500 48.078 ms - 1004
```

Handling generic server errors



```
▼{errors: [{message: "duplicate key value violates unique constraint "users_username_key"",…}],…}
   data: null
  ▶errors: [{message: "duplicate key value violates unique constraint "users_username_key"",…}]
```

```
// ·-·-·

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

  const handleSignup = async (event) => {
    // ·-·-·
    const { data, errors: rootErrors } =
      await request(USER_CREATE, {
        variables: {
          input: {
            firstName: input.firstName.value,
            lastName: input.lastName.value,
            username: input.username.value,
            password: input.password.value,
          },
        },
      });
    if (rootErrors) {
      return setUIErrors(rootErrors);
    }
    const { errors, user, authToken } = data.userCreate;
    if (errors.length > 0) {
      return setUIErrors(errors);
    }
    // ·-·-·
  };
```


AZdev

FIRST NAME

Test

LAST NAME

Account

USERNAME

test

PASSWORD

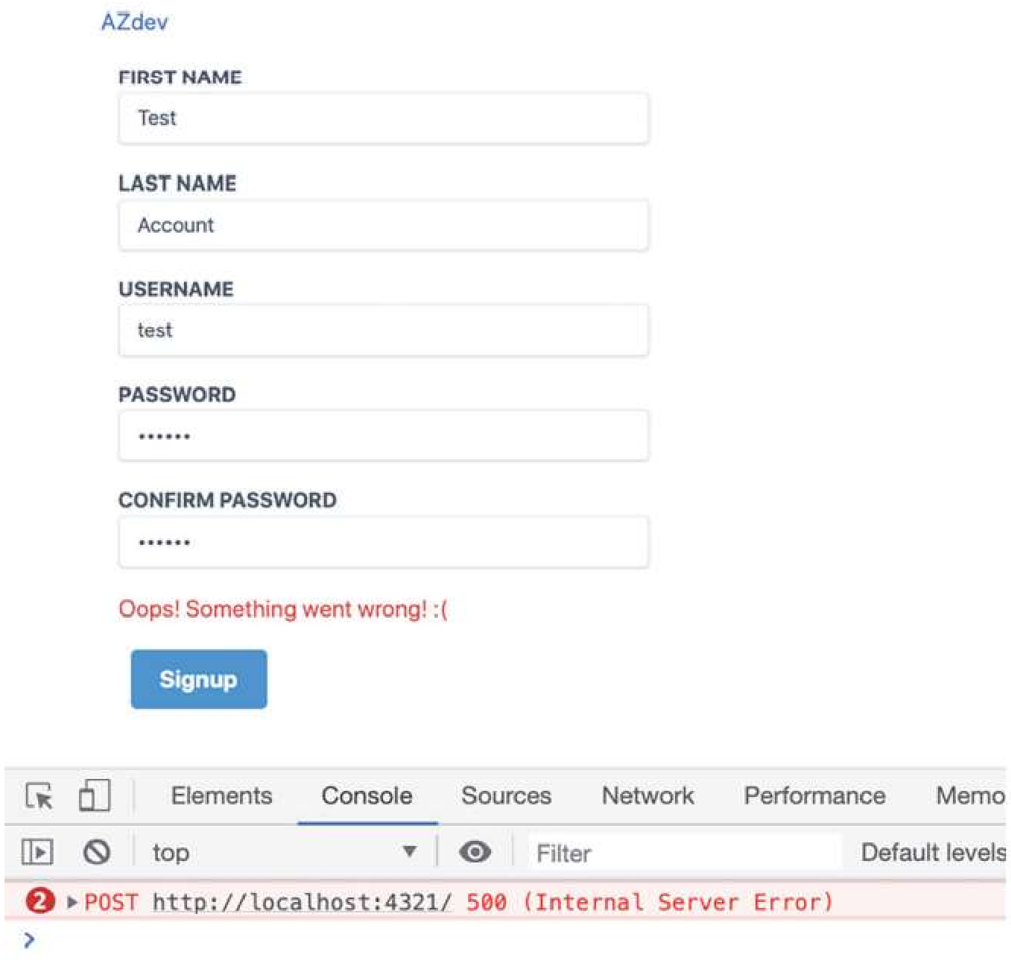
••••••

CONFIRM PASSWORD

••••••

duplicate key value violates unique constraint "users_username_key"

```
POST http://localhost:4321/ 500 (Internal Server Error)
```


AZdev

FIRST NAME
Test

LAST NAME
Account

USERNAME
test

PASSWORD
••••••

CONFIRM PASSWORD
••••••

Oops! Something went wrong! :(

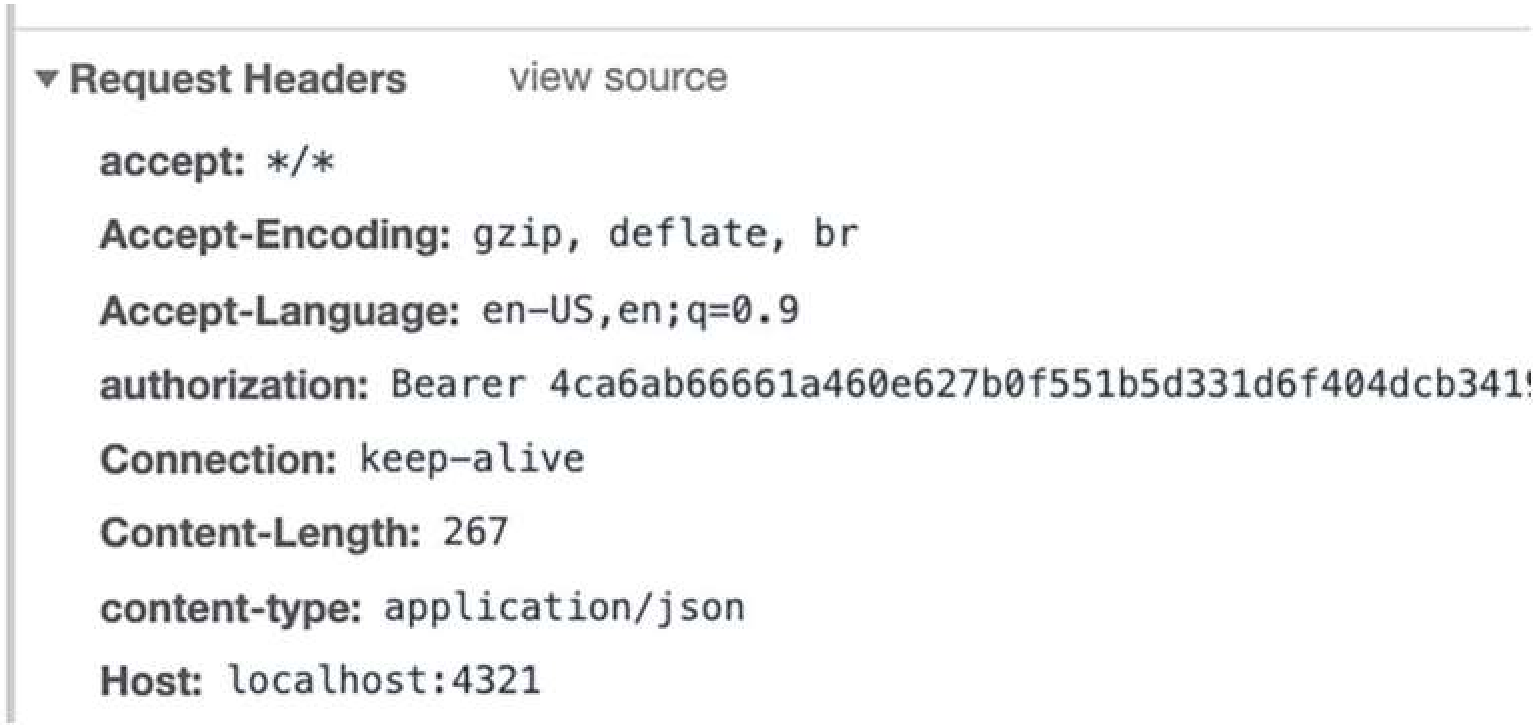
```
2 ▸ POST http://localhost:4321/ 500 (Internal Server Error)
```

```
userCreate: async ({ input }) => {
  // ·-·-·

  if (payload.errors.length === 0) {
    const authToken = randomString();
    try {
      const pgResp = await pgQuery(sqls.userInsert, {

          $1: input.username.toLowerCase(),
          $2: input.password,
          $3: input.firstName,
          $4: input.lastName,
          $5: authToken,
        });
        if (pgResp.rows[0]) {
          payload.user = pgResp.rows[0];
          payload.authToken = authToken;
        }
      } catch (err) {
        console.error(err);
        // Check the err object and either:
        // - Push a custom error message to payload
        // - Throw the err object again }
    }

    return payload;
  },
```

Authenticating GraphQL requests

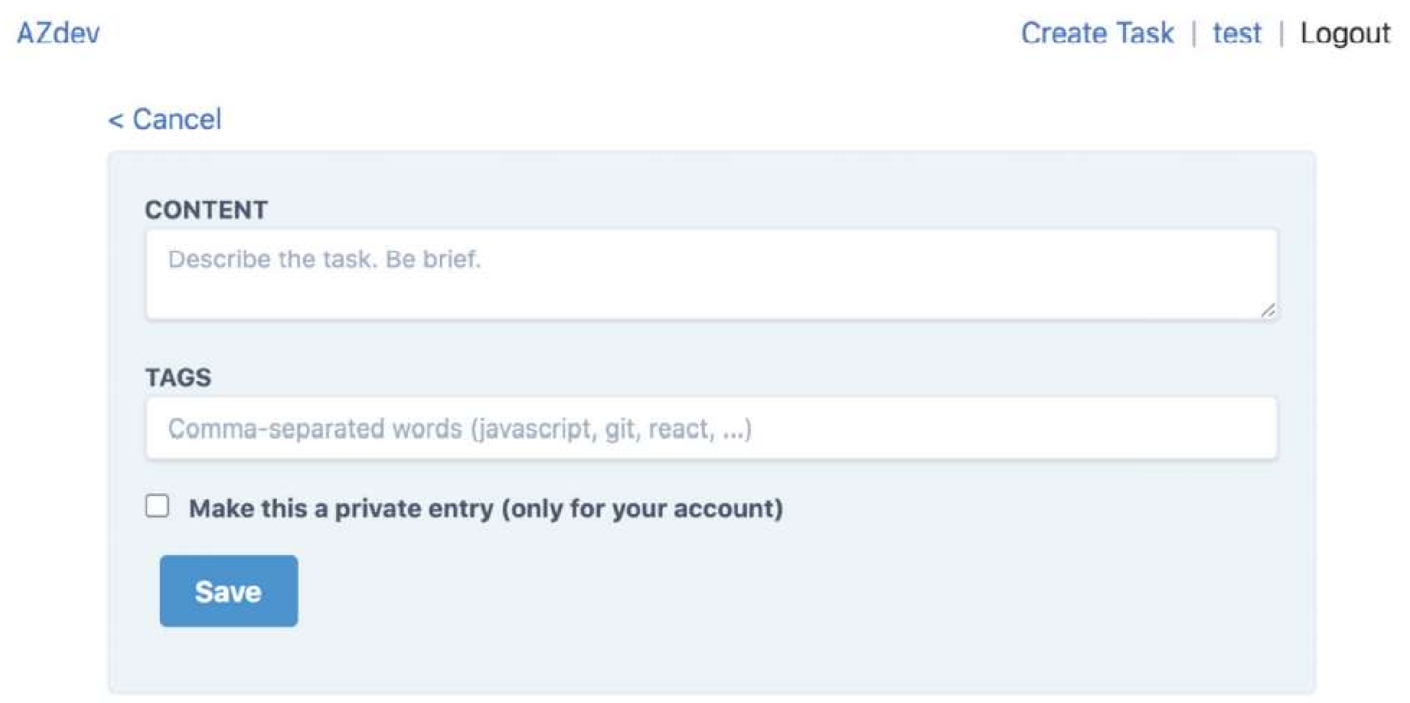
```
  const request = async (requestText, { variables } = {}) => {
    const headers = state.user
      ? { Authorization: 'Bearer ' + state.user.authToken }
      : {};
    const gsResp = await fetch(config.GRAPHQL_SERVER_URL, {
      method: 'post',
      headers: { ...headers, 'Content-Type': 'application/json'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

  }).then((response) => response.json());

  return gsResp;
};
```

Authenticating GraphQL requests



```
▼ Request Headers    view source

    accept: */*
    Accept-Encoding: gzip, deflate, br
    Accept-Language: en-US,en;q=0.9
    authorization: Bearer 4ca6ab66661a460e627b0f551b5d331d6f404dcb341
    Connection: keep-alive
    Content-Length: 267
    content-type: application/json
    Host: localhost:4321
```

The Create Task form



AZdev

Create Task | test | Logout

< Cancel

CONTENT

Describe the task. Be brief.

TAGS

Comma-separated words (javascript, git, react, ...)

☐ Make this a private entry (only for your account)

Save

- Here’s the implementation I came up with for this component.

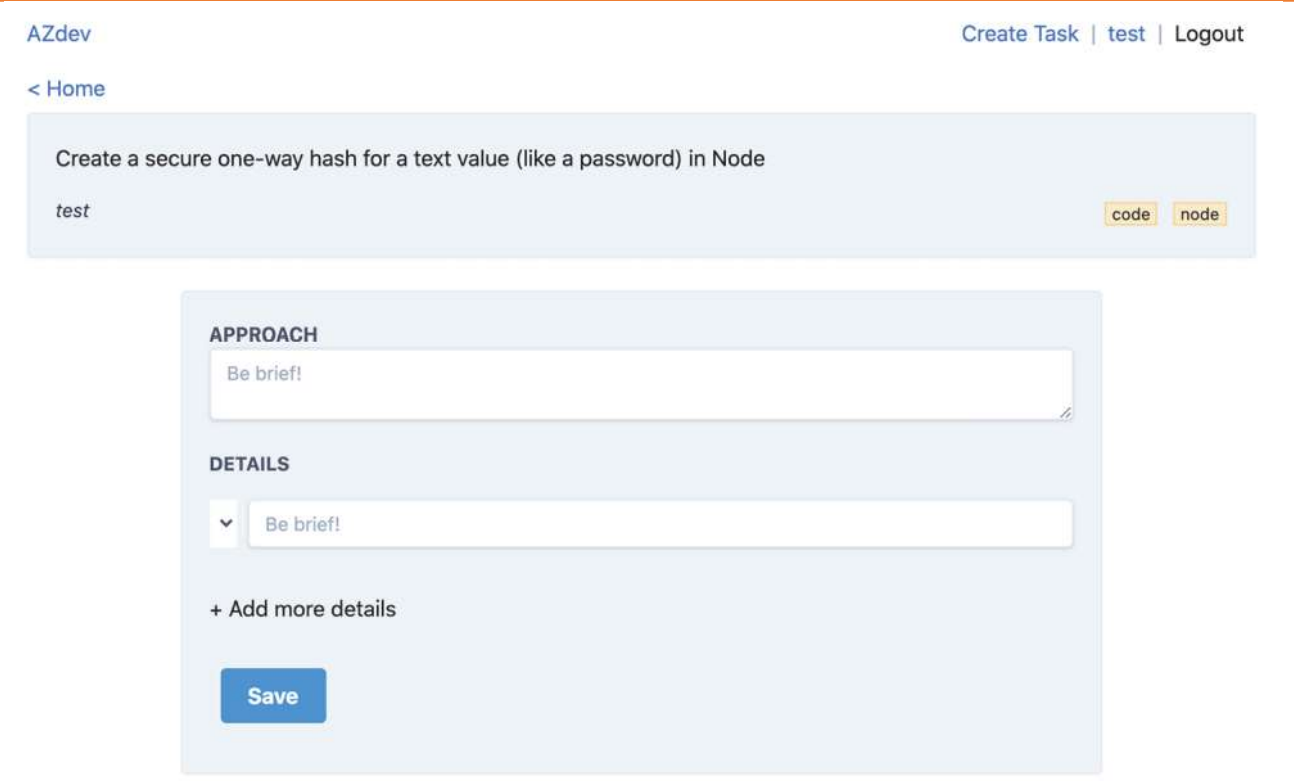
```
// ·-·-·
const TASK_CREATE = `
  mutation taskCreate($input: TaskInput!) {
    taskCreate(input: $input) {
      errors {
        message
      }
      task {
        id
      }
    }
  }
`;

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,
          },
        },
      });
      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 } },
    });
  };

  // -·-·-·
}
```

```
// ·-·-·
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}
`;
// -·-·-·
```

- Then modify the taskCreate mutation to ask for all the data required by TaskPage.

The Create Approach form


AZdev

Create Task | test | Logout

< Home

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

test

code node

APPROACH

Be brief!

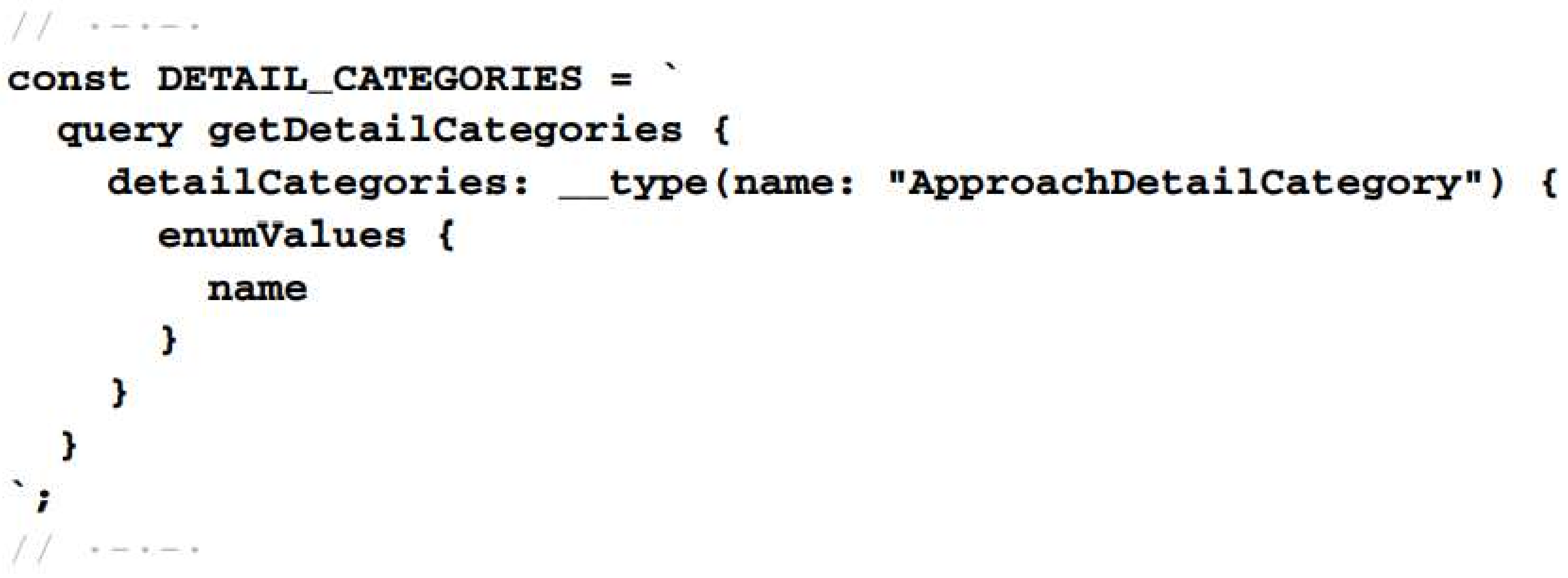
DETAILS

Be brief!

+ Add more details

Save

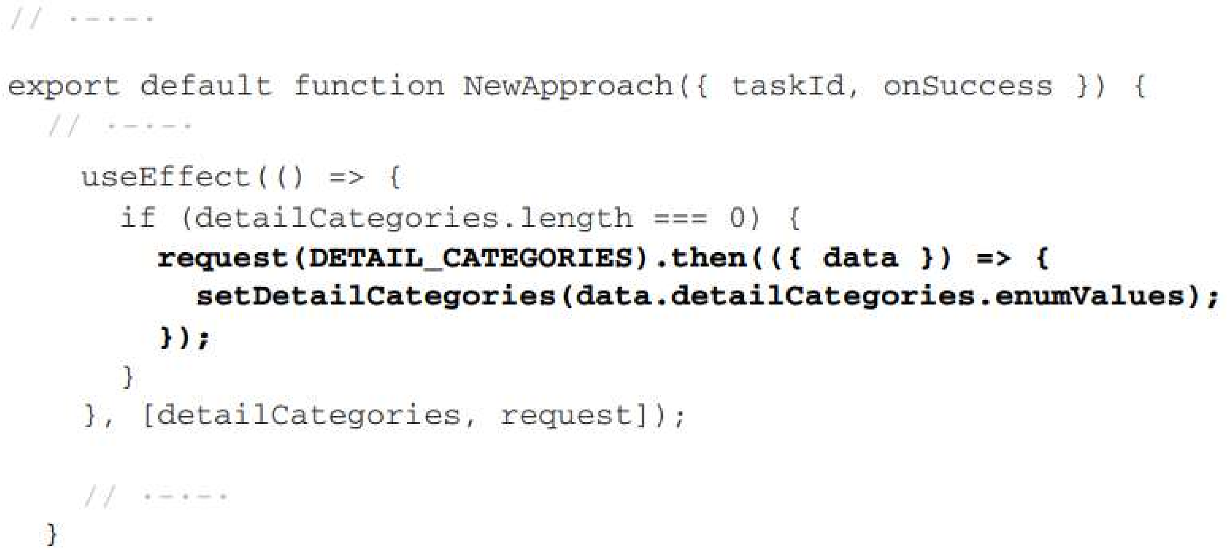
The Create Approach form



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

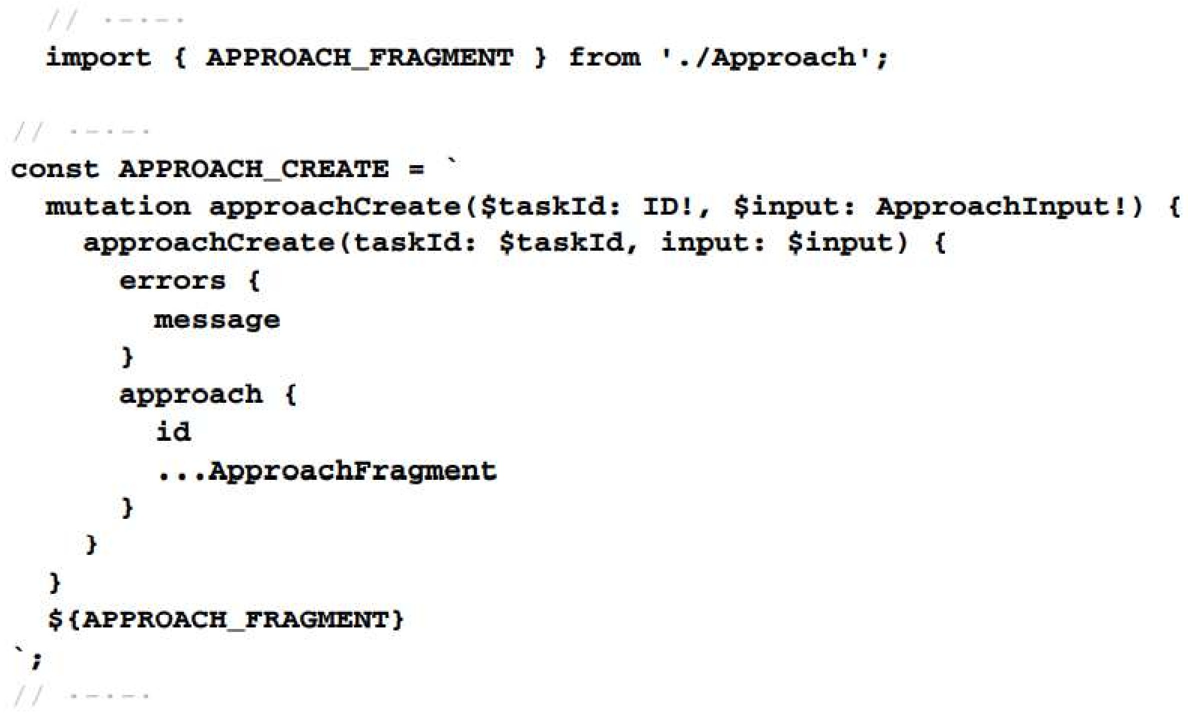
The Create Approach form

- For example, we can make it every time this form is rendered.
- That’s what I prepared in the code. Here’s how to do it



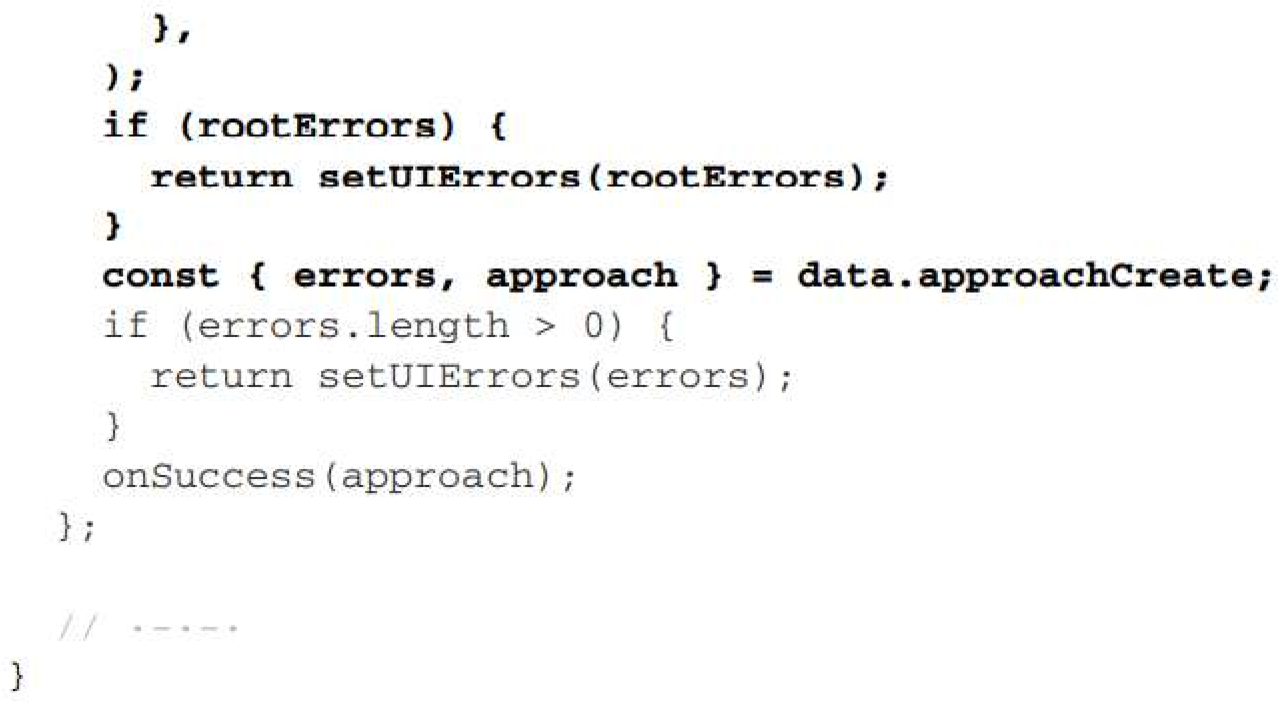
```
// ·-·-·

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

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

    // ·-·-·
}
```

- We’ve already created a fragment in the Approach component to represent these fields.
- We can just reuse it here.

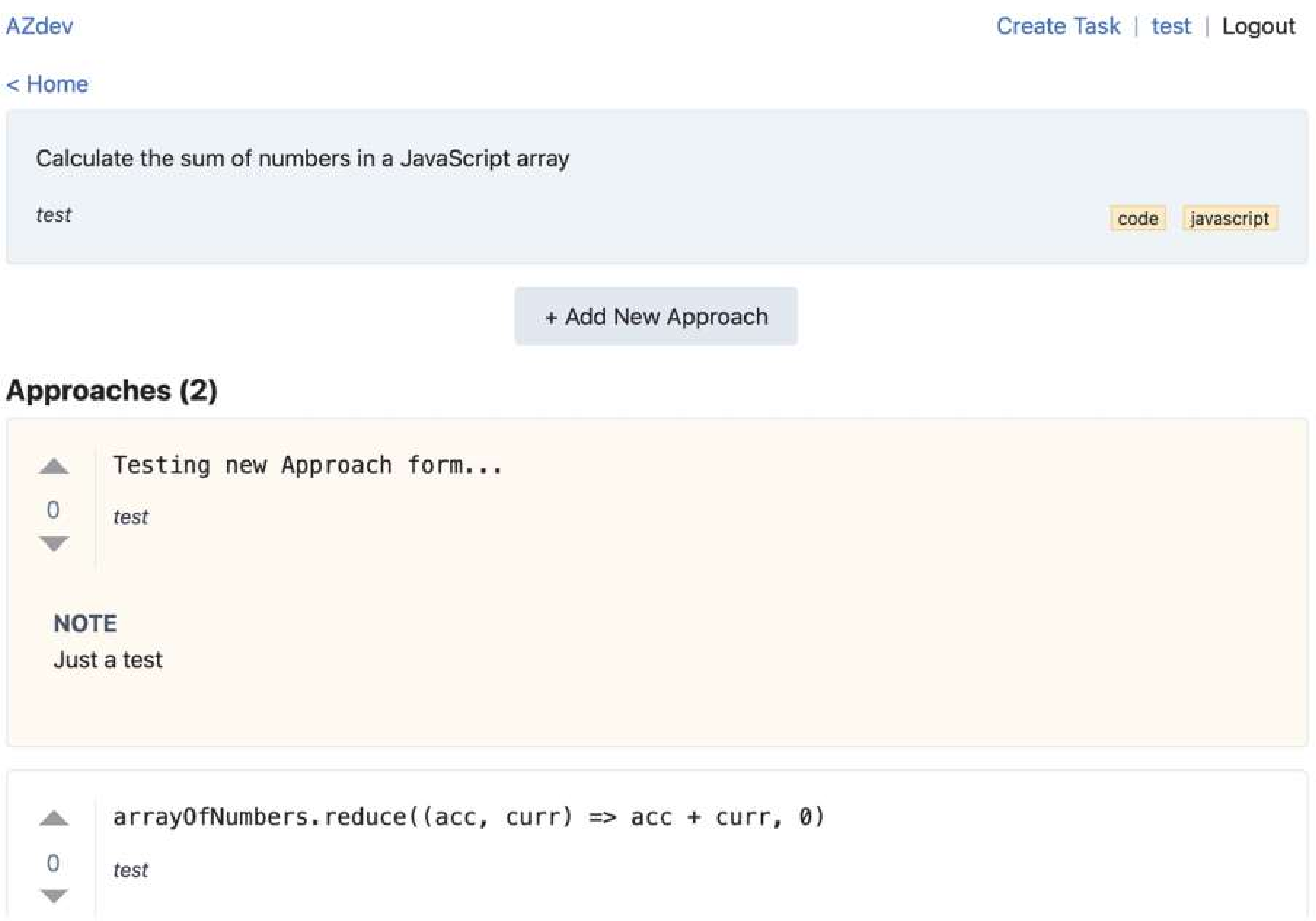


```
  // ·-·-·
  import { APPROACH_FRAGMENT } from './Approach';

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

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,
          },
        },
```

The Create Approach form



```
      },
    );
    if (rootErrors) {
      return setUIErrors(rootErrors);
    }
    const { errors, approach } = data.approachCreate;
    if (errors.length > 0) {
      return setUIErrors(errors);
    }
    onSuccess(approach);
  };

  // ·-·-·
}
```


AZdev

Create Task | test | Logout

< Home

Calculate the sum of numbers in a JavaScript array

test

code javascript

+ Add New Approach

Approaches (2)

```
Testing new Approach form...
```

0

test

NOTE
Just a test

```
arrayOfNumbers.reduce((acc, curr) => acc + curr, 0)
```

0

test

Voting on an Approach

- The vote count for each Approach is displayed between two arrows.
- Users should be able to click these arrows to up-vote or down-vote an Approach record.
- The API service schema provides the approachVote mutation for this feature.
- It expects an approachId field and an input object that has an up Boolean property.

```
// ·-·-·
const APPROACH_VOTE = `
  mutation approachVote($approachId: ID!, $up: Boolean!) {
    approachVote(approachId: $approachId, input: { up: $up }) {
      errors {
        message
      }
      updatedApproach: approach {
        id
        voteCount
      }
    }
  }
`;

export default function Approach({ approach, isHighlighted }) {
  // ·-·-·

  const handleVote = (direction) => async (event) => {
    event.preventDefault();
    const { data, errors: rootErrors } = await request(
      APPROACH_VOTE,
      {
        variables: {
          approachId: approach.id,
          up: direction === 'UP',
        },
      },
    );
    if (rootErrors) {
      return setUIErrors(rootErrors);
    }
```

Voting on an Approach

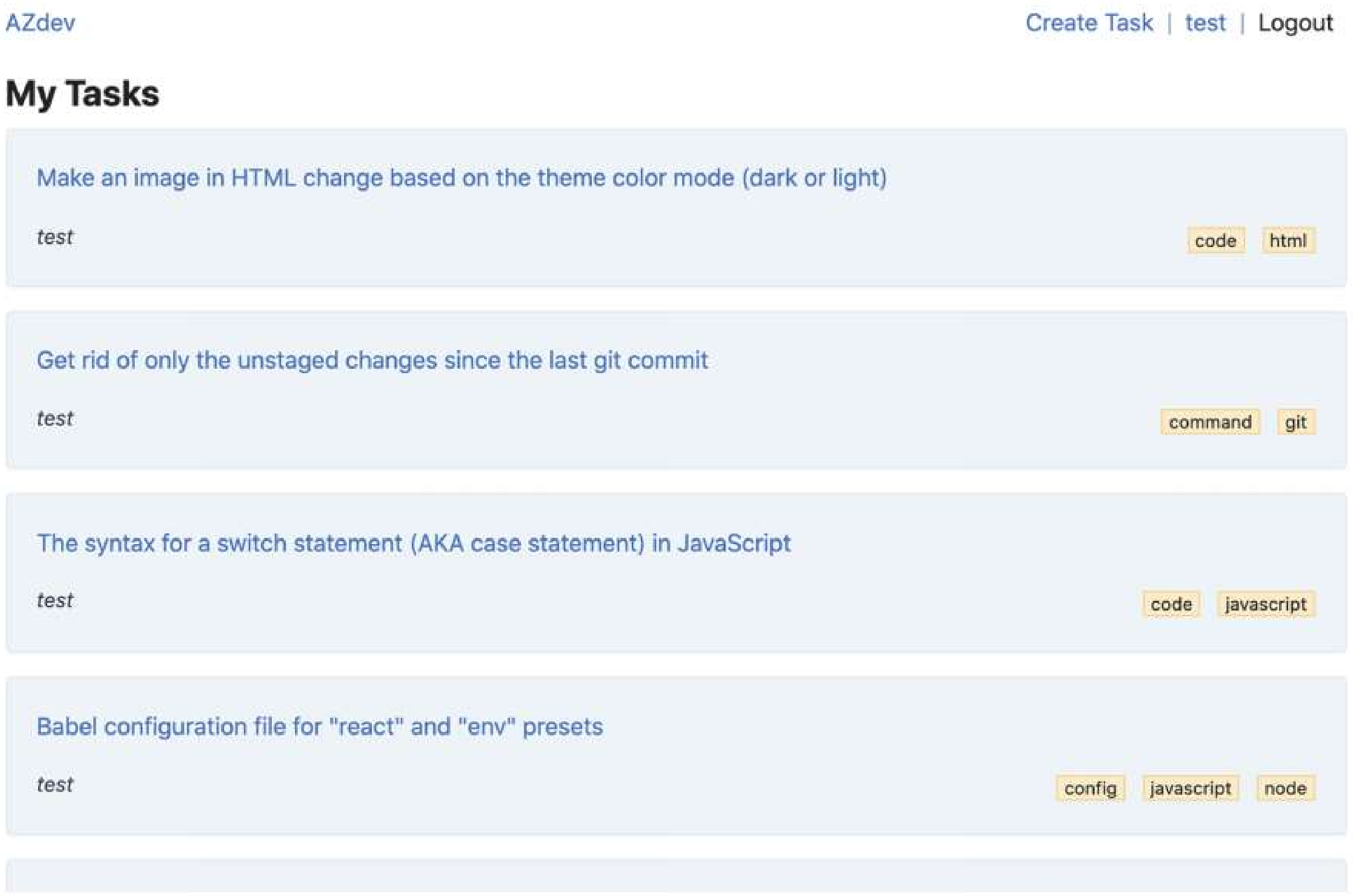
```
        const { errors, updatedApproach } = data.approachVote;
        if (errors.length > 0) {
          return setUIErrors(errors);
        }
        setVoteCount(updatedApproach.voteCount);
      };

      // ·-·-·
    }
```

Performing query requests scoped for a user

- As a logged-in user, you can click your username link in the top-right corner to navigate to a page that should list all the Task records you have created (including private ones).
- This page is similar to the home page, but the GraphQL query it requires is different.
- This one has to use the me root field and ask for the taskList field under it. It can reuse the TaskSummary fragment just as the home page did.

Performing query requests scoped for a user

```
// ·-·-·

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]);

  // ...
}
```


AZdev

Create Task | test | Logout

My Tasks

Make an image in HTML change based on the theme color mode (dark or light)

test

code html

Get rid of only the unstaged changes since the last git commit

test

command git

The syntax for a switch statement (AKA case statement) in JavaScript

test

code javascript

Babel configuration file for "react" and "env" presets

test

config javascript node

The Search form

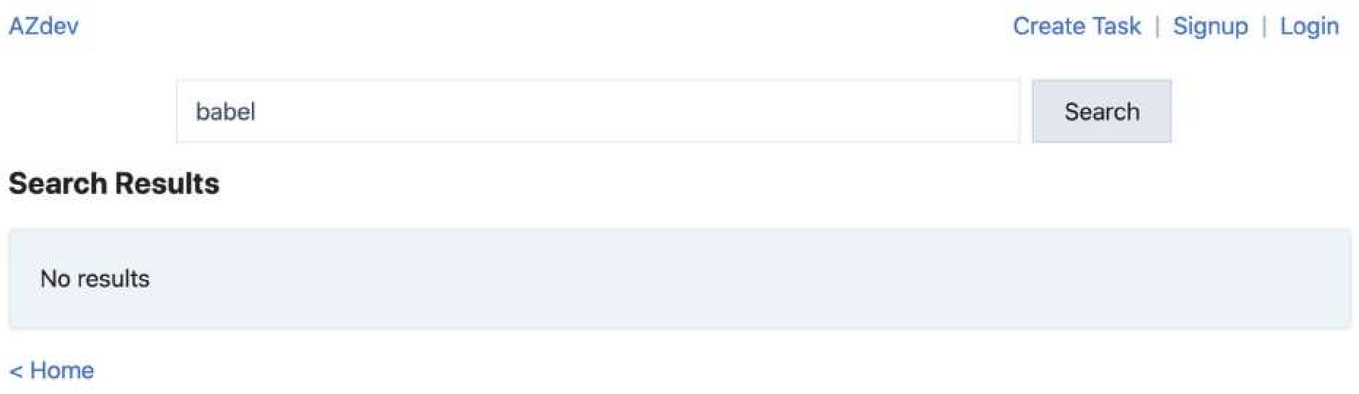
- The search feature is probably the most important feature in the AZdev application.
- It will be heavily used because it’s the entry point for all Task records. It’s also special because it should work with or without a user session.
- Without a user session, the API we implemented will exclude all private Task records.
- By including an authToken, the API will include the private Task records owned by an authenticated user.

```
<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]);

  // ...
}
```


AZdev

Create Task | Signup | Login

babel

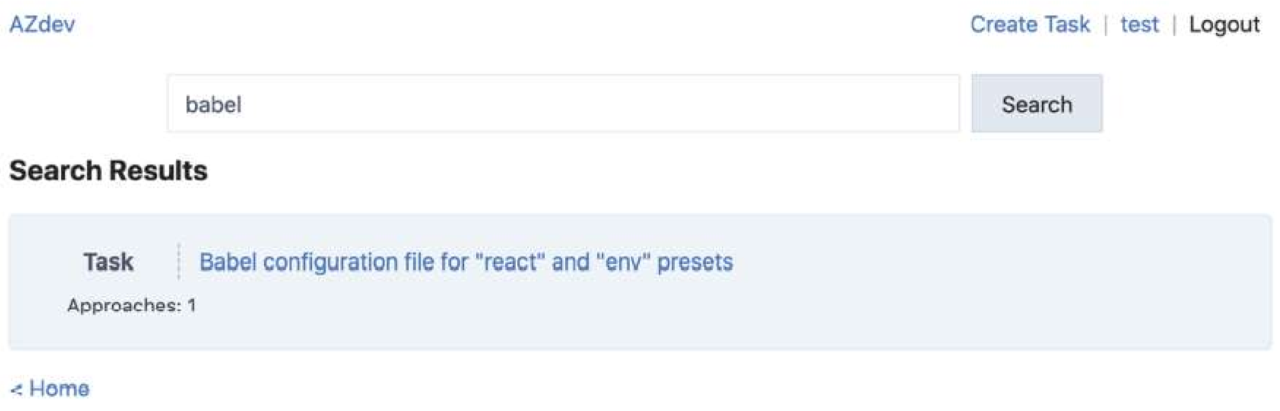
Search

Search Results

No results

< Home

A guest searching for a private Task


AZdev

Create Task | test | Logout

babel

Search

Search Results

Task	Babel configuration file for "react" and "env" presets
Approaches: 1	

< Home

An owner searching for a private Task

Summary

- To use a GraphQL operation in a frontend web application, you need to make Ajax calls.

- These calls usually cause the UI state to change, You'll have to make them in a place where it's possible to read and update the state of the UI.

- Components can define operations and fragments. An operation can be a query, a mutation, or a subscription

"Complete Lab"