Using Headless REST APIs with Liferay DXP

Learning Objectives

- Understand the advantages and disadvantages of using Liferay DXP's Headless features and capabilities
- Learn how to connect a remote application to Liferay DXP using Headless APIs

Tasks to Accomplish

- Identify the proper Headless API needed to use Liferay Objects data in a remote application
- Connect a remote application to Liferay DXP using Headless APIs

Exercise Prerequisites

- Java JDK installed to run Liferay
  - Download here: https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
  - Instructions on installation here: https://www.java.com/en/download/help/download_options.xml
- Unzipped module exercise files in the following folder structure:
  - Windows: `C:\liferay`
  - Unix Systems: `[user-home]/liferay`
- Node.js installed (v 16.13.2 used in exercise videos)
- Create React App installed
  - To install, run `npm install -g create-react-app` in your terminal
- A create-react-app project created with the appropriate files replaced with those located within the *prerequisites* folder

- The React-Bootstrap framework installed in your project
  - To install, first run `npm install react-bootstrap`, then `npm install bootstrap` in your terminal
- React Router v5.2 installed in your project
  - To install, run `npm install react-router-dom@5.2.0` in your terminal
- An IDE, such as Visual Studio Code, installed
-

An running instance of Liferay DXP 7.4 with the following contents:

- A picklist called *Account Types* with options for *Savings*, *Checking*, and *Individual Retirement*.
-

  A Liferay Object called *Bank Accounts* with the following fields:

Label	Type	Mandatory
Account Holder	Text	Yes
Account Number	Integer	Yes
Account Type	Picklist	Yes
Account Balance	Decimal	Yes

-

  At least two entries for the *Bank Accounts* Object with your Administrator as the *Account Holder* (three will be demonstrated)

Table of Contents

Use Headless APIs with a Remote React App

Exercise Goals

- Identify Liferay DXP Features to Leverage with Headless APIs
-

Integrate a Remote Application with Liferay DXP Using Headless APIs

Open the Faria Financial Management App

1. **Open** the Faria Financial Management App at `localhost:3000`.
    - Currently, the data displaying in the app is saved with the React Application. By using Headless APIs, we will replace the App data with Liferay data from a Liferay Object.
2. **Sign In** to your Liferay instance at `localhost:8080`.
3. **Open** the *Site Menu*.
4. **Open** the *Bank Accounts* Object under *People*.
    - The *Bank Accounts* Object should be scoped to the Site level and placed in the *People* section of the menu if you followed the prerequisites.

Identify the Bank Accounts Headless API

1. **Open** the Liferay API Explorer at `localhost:8080/o/api`.
2. **Click** the *REST Applications* dropdown to see a list of APIs that can be called.
3. **Click** the `c/bankaccounts` Headless API.
    - Note the reference is `http://localhost:8080/o/c/bankaccounts`. The Headless APIs for all Objects will automatically have the `c/` prefix followed by the Object name. This name is assigned when Liferay generates the Headless API.

Create the Account1.js, Account2.js, and Account3.js Files

1. **Go to** the `src` folder.
2. **Create** three new files called `Account1.js`, `Account2.js`, and `Account3.js`.

Add Code to the Account1.js File

1. **Open** the `Account1.js` file in your editor.
2. **Copy and Paste** the code snippet below into `Account1.js`:

```
import React from 'react';
import './style.css';

//Fetches data from the Headless API automatically generated when creating the Bank Accou
//This file corresponds to the checking account.

class Account1 extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      error: null,
      isLoaded: false,
      accountBalance: null,
      accountNumber: null,
      accountType: null
    };
  }

  componentDidMount() {
    fetch('/o/c/bankaccounts/42744', {
      method: 'GET',
      headers: {
        Authorization: 'Basic ' + btoa('test@liferay.com:test'),
        'Content-Type': 'application/json'
```

```
    }})
    .then(res => res.json())
    .then(
      (result) => {
        this.setState({
          isLoaded: true,
          accountBalance: result.accountBalance,
          accountNumber: result.accountNumber,
          accountType: result.accountType
        });
      },
      // Note: it's important to handle errors here
      // instead of a catch() block so that we don't swallow
      // exceptions from actual bugs in components.
      (error) => {
        this.setState({
          isLoaded: true,
          error
        });
      }
    )
}

render() {
  const { error, isLoaded, accountBalance, accountNumber, accountType} = this.state;
  if (error) {
    return <div>Error: {error.message}</div>;
  } else if (!isLoaded) {
    return <div>Loading...</div>;
  } else {
    return (
      <div className="Dashboard">
```

```
            <h2>{accountType.name} Account {accountNumber}</h2>
            <h3>${accountBalance}</h3>
        </div>
      );
    }
  }
}

export default Account1;
```

Add Code to the Account2.js File

1. **Open** the `Account2.js` file in your editor.
2. **Copy and Paste** the code snippet below into `Account2.js`:

```
import React from 'react';
import './style.css';

//Fetches data from the Headless API automatically generated when creating the Bank Accou
//This file corresponds to the savings account.

class Account2 extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      error: null,
      isLoaded: false,
      accountBalance: null,
      accountNumber: null,
      accountType: null
    };
  }
```

```
componentDidMount() {
  fetch('/o/c/bankaccounts/42746', {
    method: 'GET',
    headers: {
      Authorization: 'Basic ' + btoa('test@liferay.com:test'),
      'Content-Type': 'application/json'
    }})
    .then(res => res.json())
    .then(
      (result) => {
        this.setState({
          isLoaded: true,
          accountBalance: result.accountBalance,
          accountNumber: result.accountNumber,
          accountType: result.accountType
        });
      },
      // Note: it's important to handle errors here
      // instead of a catch() block so that we don't swallow
      // exceptions from actual bugs in components.
      (error) => {
        this.setState({
          isLoaded: true,
          error
        });
      }
    )
}

render() {
  const { error, isLoaded, accountBalance, accountNumber, accountType} = this.state;
  if (error) {
```

```
      return <div>Error: {error.message}</div>;
    } else if (!isLoaded) {
      return <div>Loading...</div>;
    } else {
      return (
        <div className="Dashboard">
            <h2>{accountType.name} Account {accountNumber}</h2>
            <h3>${accountBalance}</h3>
        </div>
      );
    }
  }
}

export default Account2;
```

Add Code to the Account3.js File

1. **Open** the `Account3.js` file in your editor.
2. **Copy and Paste** the code snippet below into `Account3.js`:

```
import React from 'react';
import './style.css';

//Fetches data from the Headless API automatically generated when creating the Bank Accou
//This file corresponds to the retirement account.

class Account3 extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      error: null,
      isLoaded: false,
```

```
    accountBalance: null,
    accountNumber: null,
    accountType: null
  };
}

componentDidMount() {
  fetch('/o/c/bankaccounts/42748', {
    method: 'GET',
    headers: {
      Authorization: 'Basic ' + btoa('test@liferay.com:test'),
      'Content-Type': 'application/json'
    }})
    .then(res => res.json())
    .then(
      (result) => {
        this.setState({
          isLoaded: true,
          accountBalance: result.accountBalance,
          accountNumber: result.accountNumber,
          accountType: result.accountType
        });
      },
      // Note: it's important to handle errors here
      // instead of a catch() block so that we don't swallow
      // exceptions from actual bugs in components.
      (error) => {
        this.setState({
          isLoaded: true,
          error
        });
      }
```

```
      )
  }

  render() {
    const { error, isLoaded, accountBalance, accountNumber, accountType} = this.state;
    if (error) {
      return <div>Error: {error.message}</div>;
    } else if (!isLoaded) {
      return <div>Loading...</div>;
    } else {
      return (
        <div className="Dashboard">
            <h2>{accountType.name} Account {accountNumber}</h2>
            <h3>${accountBalance}</h3>
        </div>
      );
    }
  }
}

export default Account3;
```

Add Imports to the Accounts.js File

1. **Open** the `Accounts.js` file.
2. **Add** imports for the three Accounts Class Objects you just created:

```
import Account1 from './Account1';
import Account2 from './Account2';
import Account3 from './Account3';
```

Add Account Classes to the Carousel

1. **Go to** the Carousel section in the `Accounts.js` file.
2. **Delete** the placeholder data under each `Carousel.Item`.
3. **Replace** with the three Account Classes:

```
<Carousel>
  <Carousel.Item interval={15000}>
    <Account1 />

  </Carousel.Item>
  <Carousel.Item interval={15000}>
    <Account2 />

  </Carousel.Item>
  <Carousel.Item interval={15000}>
    <Account3 />

  </Carousel.Item>

</Carousel>
```

1. **Open** the App in your browser to see the changes.

Test the Application

1. **Go to** the *Bank Accounts* Object in your Liferay instance.
2. **Click** the ID of one of the entries.
3. **Change** the Account Balance.
4. **Refresh** the App to view the changes.

Bonus Exercise

1. Add a new Financial Account entry for your Administrator. Adjust the app's code to display the new entity in the account carousel and test the application.

Using Headless REST APIs with Liferay Module Quiz

1. Which of the following Headless REST APIs are currently available on SwaggerHub?

   A. Headless Commerce Admin Account

   B. Headless Delivery

   C. Headless Workflow

   D. Headless Form

   E. Headless Commerce Forecast

2. Headless REST APIs are the only method of connecting clients to Liferay DXP via web API.

   A. True

   B. False

3. Which of the following is *not* generated when using REST Builder? (Choose all correct answers)

   A. JAX-RS endpoints

   B. Parsing

   C. XML

   D. OpenAPI definition

   E. Liferay Project

4. It is possible to create a Liferay project and run the REST Builder entirely from the command line in order to provide maximum flexibility for developers' preferred IDEs.

   A. True

   B. False

5. Which of the following entities will automatically generate a new headless API when created?

   A. Forms

   B. Objects

   C. Documents

   D. Blogs

Answer Key

1. C
2. False
3. D, E
4. True
5. B