ADD REVIEWS TO OUR APP

Add Reviews to the App

In this section we are going to add 3 new important features to our app:

- a text box to allow the user to provide a review
- interpretation of the text, using Amazon Comprehend's powerful NLP features, to pull out sentiment and key words
- save the review and interpretation data using the AppSync API

To add these features, we need to update our **App.js** file. Replace the contents of App.js with this file and **save App.js**:

```
import React, { Fragment, Component } from "react";
import "./App.css";
import { Container, Row, Col, Button, Input } from "reactstrap";
import Header from "./components/header";
import SideCard from "./components/sideCard";
import MenuItem from "./components/menu";
import OrderHistory from "./components/orders";
import Amplify, {Auth, Hub, Cache, API, graphqlOperation} from "aws-
amplify";
import { Authenticator, ChatBot  } from "aws-amplify-react";
import Predictions, { AmazonAIPredictionsProvider} from "@aws-
amplify/predictions";
import { createOrder, createItem, updateOrder,  createReview,
createReviewPhrase } from "./graphql/mutations";
import awsconfig from "./aws-exports";

Amplify.addPluggable(new AmazonAIPredictionsProvider());
Amplify.configure(awsconfig);

const signUpConfig = {
  header: "Welcome!",
  signUpFields: [
    {
      label: "First Name",
```

```
      key: "given_name",
      placeholder: "First Name",
      required: true,
      displayOrder: 5
    },
    {
      label: "Last Name",
      key: "family_name",
      placeholder: "Last Name",
      required: true,
      displayOrder: 6
    },
    {
      label: "Address",
      key: "address",
      placeholder: "Address",
      required: true,
      displayOrder: 7
    }
  ]
};

class App extends Component {
  state = {
    showType: "",
    loggedIn: false,
    currentUser: null
  };

  listProductsWithVariant = `query ListProducts(
    $filter: ModelProductFilterInput
    $limit: Int
    $nextToken: String
  ) {
    listProducts(filter: $filter, limit: $limit, nextToken: $nextToken) {
      items {
        id
        productId
        productName
        category
        description
        defaultPrice
        sizes {
            items {
              price
              size
            }
          }
      }
      nextToken
    }
```

```
    }
    `;

    async loadExistingOrder(orderId) {
      const getOrderWithItems = `query GetOrder($id: ID!) {
        getOrder(id: $id) {
          id
          name
          user
          phone
          email
          orderDate
          orderTotal
          deliveryType
          deliveryDate
          status
          items {
            items {
              id
              itemName
              comments
              quantity
              size
              unitPrice
              totalPrice
            }
            nextToken
          }
        }
      }
      `;
      // Now we want to update the state with the new order data
      const orderInput = {
        id: orderId
      };
      const getOrderResult = await API.graphql(
        graphqlOperation(getOrderWithItems, orderInput)
      );
      this.setState({
        currentOrder: getOrderResult.data.getOrder
      });
    }

    createNewItem = async itemInput => {
      const newItem = await API.graphql(
        graphqlOperation(createItem, {
          input: itemInput
        })
      );
      return newItem;
    };
```

```
createNewOrder = async orderInput => {
  const newOrder = await API.graphql(
    graphqlOperation(createOrder, {
      input: orderInput
    })
  );
  return newOrder;
};

appendLeadingZeroes = n => {
  if (n <= 9) {
    return "0" + n;
  }
  return n;
};

createOrderName(today) {
  return (
    today.getFullYear() +
    "-" +
    this.appendLeadingZeroes(today.getMonth() + 1) +
    "-" +
    this.appendLeadingZeroes(today.getDate())
  );
}

getOrderDate(today) {
  return (
    today.getFullYear() +
    "-" +
    this.appendLeadingZeroes(today.getMonth() + 1) +
    "-" +
    this.appendLeadingZeroes(today.getDate()) +
    "T" +
    this.appendLeadingZeroes(today.getHours()) +
    ":" +
    this.appendLeadingZeroes(today.getMinutes()) +
    ":" +
    this.appendLeadingZeroes(today.getSeconds()) +
    "-05:00:00"
  );
}

async createNewOrderConstruct() {
  var today = new Date();
  var orderName = this.createOrderName(today);
  var orderDate = this.getOrderDate(today);

  const orderInput = {
    name: "ORDER: " + orderName,
```

```
      user: this.state.currentUser,
      phone: this.state.currentUserData.attributes.phone_number,
      email: this.state.currentUserData.attributes.email,
      orderDate: orderDate,
      orderTotal: this.getTotal(this.state.currentOrder),
      deliveryType: "Carryout",
      deliveryDate: orderDate,
      status: "IN PROGRESS"
    };

    const newOrder = await this.createNewOrder(orderInput);
    return newOrder;
  }
  handleAddItem = async item => {
    var checkOrder = this.state.currentOrder;
    if (!checkOrder) {
      // Create new order
      //var cUser = await Auth.currentAuthenticatedUser();
      var today = new Date();
      const expiration = new Date(today.getTime() + 60 * 60000);
      var newOrder = await this.createNewOrderConstruct();
      Cache.setItem("currentOrder", newOrder.data.createOrder.id, {
        priority: 3,
        expires: expiration.getTime()
      });
      checkOrder = newOrder.data.createOrder;
    }

    var currentOrderId = checkOrder.id;

    const totalPrice = item.quantity * item.price;
    const itemInput = {
      itemName: item.itemName,
      comments: "No Comments",
      quantity: item.quantity,
      size: item.size,
      unitPrice: item.price,
      totalPrice: totalPrice,
      itemOrderId: currentOrderId
    };
    await this.createNewItem(itemInput);
    this.loadExistingOrder(currentOrderId);
  };

  loadCurrentUser() {
    Auth.currentAuthenticatedUser().then(userInfo => {
      this.setState({
        loggedIn: true,
        currentUser: userInfo.username,
        currentUserData: userInfo
      });
```

```
      });
    }

    getPriceForSize(pId, selSize) {
      const retVal = this.state.menuItems.filter(item => item.productId ===
  pId);
      const rVal2 = retVal[0].sizes.items.filter(
        item2 => item2.size.toUpperCase() === selSize.toUpperCase()
      );
      return rVal2[0].price;
    }

    isLoggedIn = async () => {
      return await Auth.currentAuthenticatedUser()
        .then(() => {
          return true;
        })
        .catch(() => {
          return false;
        });
    };

    getCurrentUser = async () => {
      const user = await Auth.currentAuthenticatedUser();
      return user;
    };

    getTotal = items => {
      var totalPrice = 0;
      for (var i in items) {
        var qty = items[i]["quantity"];
        var price = items[i]["unitPrice"];
        var qtyPrice = qty * price;
        totalPrice += qtyPrice;
      }
      return totalPrice.toFixed(2);
    };

    createNewOrderConstructSync = () => {
      var today = new Date();
      var orderName = this.createOrderName(today);
      var orderDate = this.getOrderDate(today);

      const orderInput = {
        name: "ORDER: " + orderName,
        user: this.state.currentUser,
        phone: this.state.currentUserData.attributes.phone_number,
        email: this.state.currentUserData.attributes.email,
        orderDate: orderDate,
        orderTotal: this.getTotal(this.state.currentOrder),
        deliveryType: "Carryout",
```

```
      deliveryDate: orderDate,
      status: "IN PROGRESS"
    };

    this.createNewOrder(orderInput)
      .then(newOrder => {
        return newOrder;
      })
      .catch(err => {
        console.log(err);
      });
  };

  createNewItemSync = itemInput => {
    API.graphql(
      graphqlOperation(createItem, {
        input: itemInput
      })
    ).then(newItem => {
      return newItem;
    });
  };

  getItems = cOrder => {
    if (cOrder && cOrder.items) {
      return cOrder.items.items;
    } else {
      return null;
    }
  };

  completeOrder = () => {
    this.setState({ showType: "orderComplete" });
    const orderInput = {
      id: this.state.currentOrder.id,
      name: this.state.currentOrder.name,
      user: this.state.currentUser,
      phone: this.state.currentOrder.phone,
      email: this.state.currentOrder.email,
      orderDate: this.state.currentOrder.orderDate,
      orderTotal: this.getTotal(this.state.currentOrder.items.items),
      deliveryType: "Carryout",
      deliveryDate: this.state.currentOrder.deliveryDate,
      status: "COMPLETE"
    };

    API.graphql(
      graphqlOperation(updateOrder, {
        input: orderInput
      })
    ).then(result => {
```

```
      this.setState({
        currentOrder: null
      });
      Cache.removeItem("currentOrder");
    });
  };

  componentDidMount = () => {
    Hub.listen("auth", ({ payload: { event, data } }) => {
      switch (event) {
        case "signIn":
          this.setState({
            currentUser: data.username,
            currentUserData: data,
            loggedIn: true
          });
          break;
        case "signOut":
          this.setState({
            currentUser: null,
            loggedIn: false
          });
          break;
        default:
          break;
      }
    });
    this.loadCurrentUser();

    var currentOrderId = null;
    var checkOrder = this.state.currentOrder;
    if (checkOrder) currentOrderId = checkOrder.id;
    else currentOrderId = Cache.getItem("currentOrder");
    if (currentOrderId) {
      this.loadExistingOrder(currentOrderId);
    }

    // Get menu items
    const limit = {
      limit: 100
    };

    API.graphql(graphqlOperation(this.listProductsWithVariant,
limit)).then(result => {
      this.setState({
        menuItems: result.data.listProducts.items
      });
    });

  };
```

```
handleLogin = () => {
  this.setState({
    showType: "login"
  });
};

handleLogout = () => {
  this.setState({
    showType: "login"
  });
};

handleOrder = () => {
  this.setState({
    showType: "menu"
  });
};

handleHistory = () => {
  this.setState({
    showType: "orders"
  });
};

handleCheckout = () => {
  this.setState({
    showType: "checkout"
  });
};

handleChat = () => {
  this.setState({
    showType: "chat"
  });
};

handleReview = () => {
  this.setState({
    showType: "review"
  });
};

// ChatBot Helper Functions
putTheChatOrder = async item => {
  var checkOrder = this.state.currentOrder;
  if (!checkOrder) {
    // Create new order
    //var cUser = await Auth.currentAuthenticatedUser();
    var today = new Date();
    const expiration = new Date(today.getTime() + 60 * 60000);
    var newOrder = await this.createNewOrderConstruct();
```

```
      Cache.setItem("currentOrder", newOrder.data.createOrder.id, {
        priority: 3,
        expires: expiration.getTime()
      });
      checkOrder = newOrder.data.createOrder;
    }

    var currentOrderId = checkOrder.id;

    const totalPrice = item.quantity * item.price;
    const itemInput = {
      itemName: item.itemName,
      comments: "Ordered from chatbot",
      quantity: item.quantity,
      size: item.size,
      unitPrice: item.price,
      totalPrice: totalPrice,
      itemOrderId: currentOrderId
    };
    await this.createNewItem(itemInput);
    this.loadExistingOrder(currentOrderId);
  };

  chatItemHelper(specialty) {
    var specLower = ""
    if (specialty)
      specLower = specialty.toLowerCase();
    switch (specLower) {
      case "supreme":
        return "0002";
      case "ultimate":
        return "0001";
      case "veggie":
        return "0003";
      case "meat lovers":
        return "0008";
      default:
        return "0004";
    }
  }

  handleComplete(err, confirmation) {
    if (err) {
      console.log("Bot conversation failed");
      return;
    }

    var pid = this.chatItemHelper(confirmation.slots.specialty);
    var price = this.getPriceForSize(pid, confirmation.slots.size);
    var specName = confirmation.slots.specialty;
    if (!specName) specName = "Cheese Pizza";
```

```
    var item = {
      itemName: specName,
      quantity: 1,
      price: price,
      size: confirmation.slots.size
    };
    this.putTheChatOrder(item);
    return "Great, I am adding that to your order!";
  }

  // Review Helper Functions
  saveReview(comments, results) {
    var today = new Date();
    const reviewInput = {
      comments: comments,
      username: this.state.currentUser,
      dateAdded: this.getOrderDate(today),
      sentiment: results.textInterpretation.sentiment.predominant
    };
    API.graphql(
      graphqlOperation(createReview, {
        input: reviewInput
      })
    );

    var i = 0;
    for (i in results.textInterpretation.keyPhrases) {
      const reviewPhraseInput = {
        phraseText: results.textInterpretation.keyPhrases[i].text,
        phraseType: "KEY PHRASE",
        dateAdded: this.getOrderDate(today),
        username: this.state.currentUser
      };

      API.graphql(
        graphqlOperation(createReviewPhrase, {
          input: reviewPhraseInput
        })
      );
    }

    for (i in results.textInterpretation.textEntities) {
      const reviewPhraseInput = {
        phraseText: results.textInterpretation.textEntities[i].text,
        phraseType: results.textInterpretation.textEntities[i].type,
        dateAdded: this.getOrderDate(today),
        username: this.state.currentUser
      };

      API.graphql(
        graphqlOperation(createReviewPhrase, {
```

```
            input: reviewPhraseInput
          })
        );
      }
    }

    updateComments(event) {
      this.setState({
        reviewComments: event.target.value
      });
    }

    submitFeedback = event => {
      Predictions.interpret({
        text: {
          source: {
            text: this.state.reviewComments
          },
          type: "ALL"
        }
      })
        .then(result => {
          const sentiment =
  result.textInterpretation.sentiment.predominant;
          var sentimentMessage =
            "Thank you for your feedback.  We appreciate your comments and
  love to hear from our customers!";
          switch (sentiment) {
            case "POSITIVE":
              sentimentMessage =
                "We are ECSTATIC to hear about your positive experience
  with our store.  We hope we can continue to reach your expectations and
  hope you order from us again!";
              break;
            case "NEGATIVE":
              sentimentMessage =
                "We are VERY SORRY to hear about your experience with us.
  Please know that we take your comments very seriously and hope to earn
  your business in the future. Give us another chance!";
              break;
            default:
              break;
          }
          this.setState({
            reviewResponse: sentimentMessage
          });
          this.saveReview(this.state.reviewComments, result);
        })
        .catch(err => {
          console.log("Prediction error", err);
        });
```

```
      this.setState({
        showType: "reviewComplete"
      });
    };

    render() {
      return (
        <Fragment>
          <Header
            onHandleLogin={this.handleLogin}
            onHandleLogout={this.handleLogout}
            onHandleHistory={this.handleHistory}
            onHandleReview={this.handleReview}
            loggedIn={this.state.loggedIn}
            userName={this.state.currentUser}
            onHandleOrder={this.handleOrder}
          />
          <div className="my-5 py-5">
            <Container className="px-0">
              <Row
                noGutters
                className="pt-2 pt-md-5 w-100 px-4 px-xl-0 position-
relative"
              >
                <Col
                  xs={{ order: 2 }}
                  md={{ size: 4, order: 1 }}
                  tag="aside"
                  className="pb-5 mb-5 pb-md-0 mb-md-0 mx-auto mx-md-0"
                >
                  <SideCard currentOrder={this.state.currentOrder}
onHandleCheckout={this.handleCheckout} onHandleChat={this.handleChat}/>
                </Col>

                <Col
                  xs={{ order: 1 }}
                  md={{ size: 7, offset: 1 }}
                  tag="section"
                  className="py-5 mb-5 py-md-0 mb-md-0"
                >
                  {this.state.showType === "" ? "This is the main content"
: null}
                  {this.state.showType === "login" ? (
                    <Authenticator signUpConfig={signUpConfig} />
                  ) : null}
                  {this.state.showType === "menu" ? (<MenuItem onAddItem=
{this.handleAddItem}></MenuItem>) : null}
                  {this.state.showType === "orders" ? (
                    <OrderHistory userName={this.state.currentUser} />
                  ) : null}
                  {this.state.showType === "checkout" ? (
```

```
                  <Fragment>
                    <Container>
                      <Row className="font-weight-bold">
                        <Col>Item Name</Col>
                        <Col>Options</Col>
                        <Col>Price</Col>
                      </Row>
                      {this.getItems(this.state.currentOrder)
                        ? this.getItems(this.state.currentOrder).map(
                            orderInfo => (
                              <Row key={orderInfo.id}>
                                <Col>{orderInfo.itemName}</Col>
                                <Col>Qty: {orderInfo.quantity}</Col>
                                <Col>{orderInfo.totalPrice}</Col>
                              </Row>
                            )
                          )
                        : null}
                      <Row>
                        <Col>TOTAL</Col>
                        <Col></Col>
                        <Col>
                          $
                          {this.getTotal(
                            this.getItems(this.state.currentOrder)
                          )}
                        </Col>
                      </Row>
                    </Container>
                    <Button onClick={this.completeOrder}>Complete
    Order</Button>
                  </Fragment>
                ) : null}
                {this.state.showType === "orderComplete" ? (
                  <div>Thank you for your order!</div>
                ) : null}
                {this.state.showType === "chat" ? (

                  <ChatBot
                    title="Place an Order"
                    botName="AndyPizzaOrder_dev"
                    welcomeMessage={
                      "Hi " +
                      this.state.currentUser +
                      ", how can i assist you today?"
                    }
                    onComplete={this.handleComplete.bind(this)}
                    clearOnComplete={false}
                    conversationModeOn={true}
                  />
                ) : null}
```

```
                {this.state.showType === "reviewComplete" ? (
                  <Fragment>
                    <div>{this.state.reviewResponse}</div>
                  </Fragment>
                ) : null}
                {this.state.showType === "review" ? (
                  <Fragment>
                    <Input
                      type="textarea"
                      rows="6"
                      onChange={this.updateComments.bind(this)}
                    ></Input>
                    <br></br>
                    <Button
                      type="submit"
                      onClick={this.submitFeedback.bind(this)}
                    >
                      Submit Feedback
                    </Button>
                  </Fragment>
                ) : null}

              </Col>
            </Row>
          </Container>
        </div>
      </Fragment>
    );
  }
}

export default App;
```

The above code does the following:

- Imports the new graphql queries and the Predictions component – `import Predictions, { AmazonAIPredictionsProvider } from "@aws-amplify/predic`
- Configures Amazon AI predictions provider, which handles the logic for the Comprehend service – `Amplify.addPluggable(new AmazonAIPredictionsProvider());`
- Adds logic to interpret the text and save the data, sentiment and key phrases – `saveReview(comments, results) {`
- Displays a message back to the user based on sentiment of their comments – `const sentiment = result.textInterpretation.sentiment.predominant;`

Test the app

Let's write a review and test our app. Return to the application preview tab and click on the **Tell us how we're doing** button and then write a very positive review:

```
Thank you for the great service that we received while we were in your
location in New York!  The food was very flavorful and a great value for
the price.  We live in Boston but were visiting relatives.  Maybe you can
put a location in the New England area?

Thank you and your servers so much for their great service!
```

Tell us how we're doing

Thank you for the great service that we received while we were in your location in New York! The food was very flavorful and a great value for the price. We live in Boston but were visiting relatives. Maybe you can put a location in the New England area?

Thank you and your servers so much for their great service!

Submit Feedback

The response will be customized based on **the positive sentiment** of the review.

Tell us how we're doing

We are ECSTATIC to hear about your positive experience with our store. We hope we can continue to reach your expectations and hope you order from us again!

Likewise, let's click on the **Tell us how we're doing** button again and provide a review that is a little more scathing...

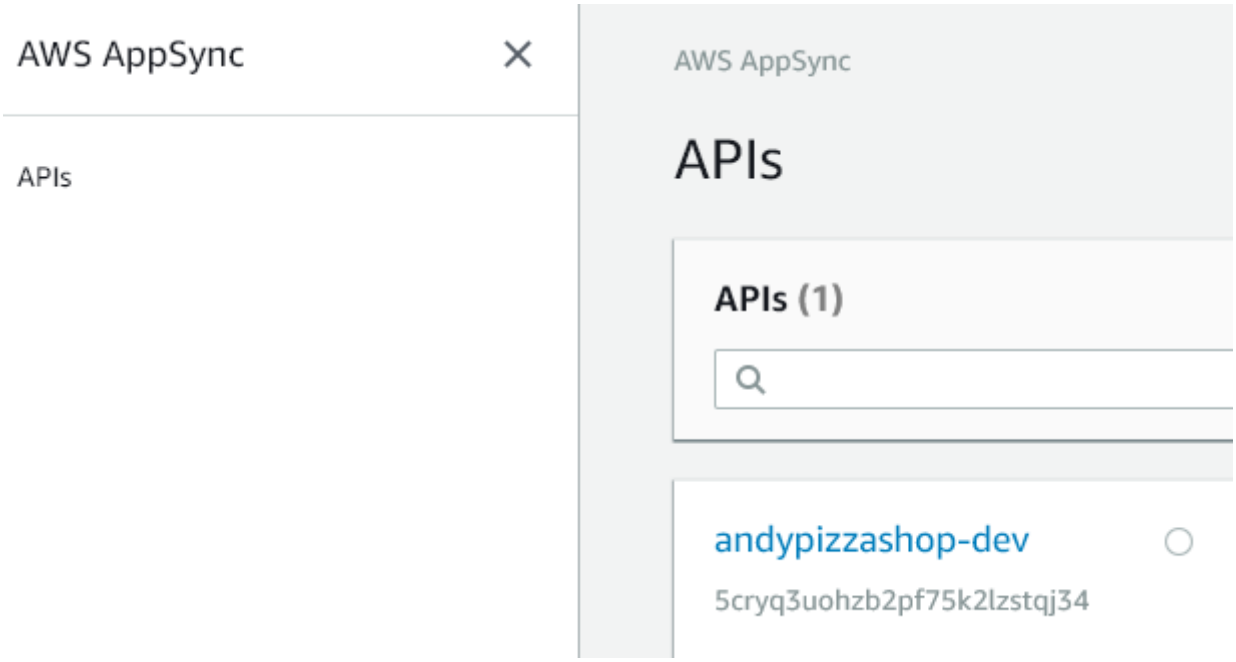
```
We recently ordered carryout from your Florida location and were not
satisfied with our experience.  At first, the delivery driver was 40
minutes late and then when we got our food only 2 of the 3 pizzas were
delivered.
By the time we ate the food it was cold and then my son was sick most of
the night.

I'm not sure we'll be back!
```

The response you get should be customized to the negative sentiment of the review.

Tell us how we're doing

We are VERY SORRY to hear about your experience with us. Please know that we take your comments very seriously and hope to earn your business in the future. Give us another chance!

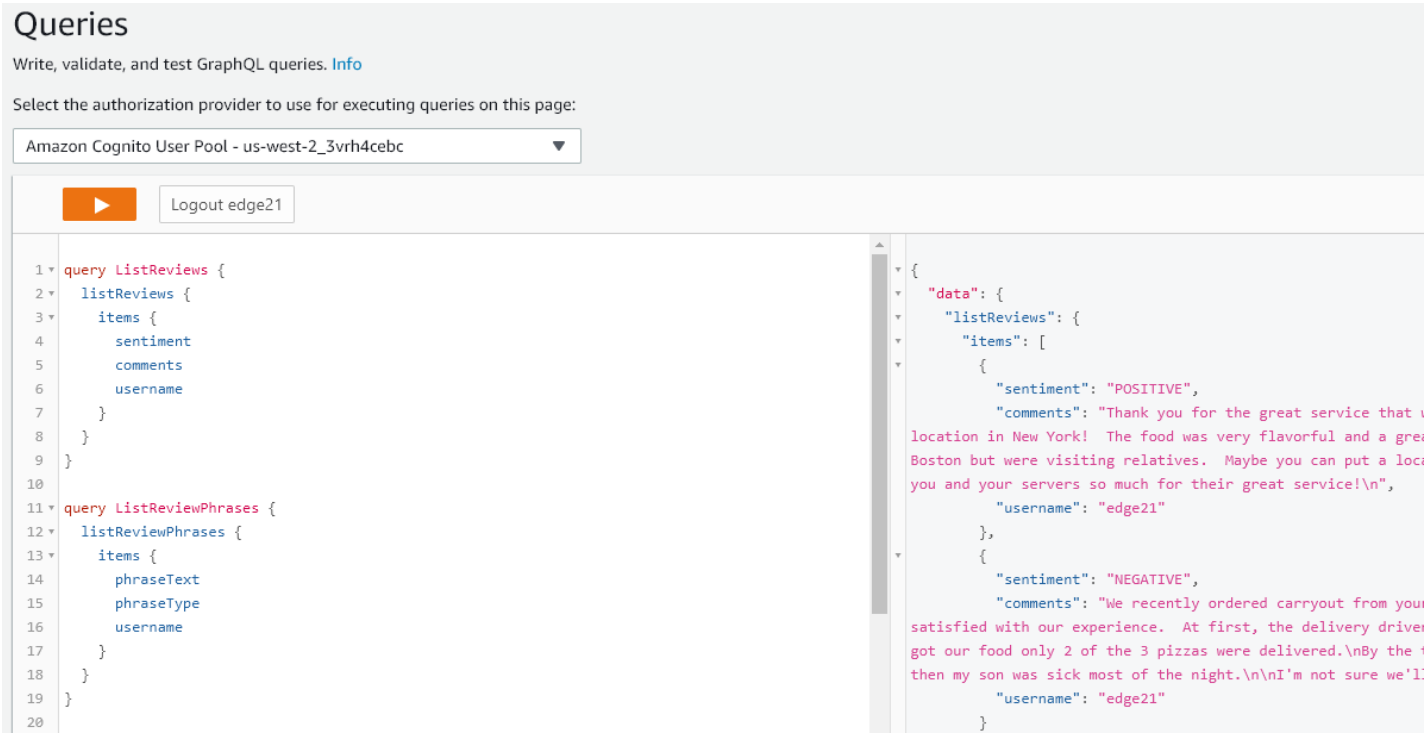
Lastly, let's verify that the data we received from the Comprhend service was stored with our reviews. Go the AWS AppSync console and click on the API



Open the Queries section and enter the following queries.

```
query ListReviews {
  listReviews {
    items {
      sentiment
      comments
      username
    }
  }
}

query ListReviewPhrases {
  listReviewPhrases {
    items {
      phraseText
      phraseType
      username
    }
  }
}
```

Click the orange ‘Play’ icon and run both queries to ensure you get data for each.



Note

If you receive an “UnauthorizedException”, you may have to login to AppSync Queries section by clicking the **Login with User Pools** button.

Congrats! You have added text analysis to your reviews!