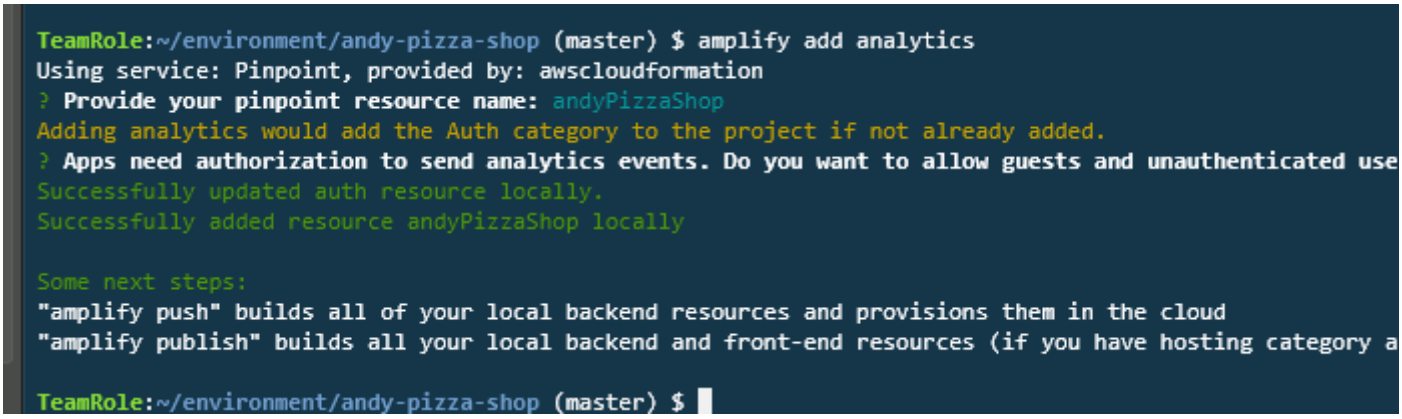
ADD ANALYTICS SERVICE

Use Amazon Pinpoint to collect analytics

One of the requirements was to collect analytics about our end users so that we can later do analysis and forecasting. Amazon Pinpoint makes this very easy! To add analytics, first use the amplify **add analytics** command:

```
amplify add analytics
```

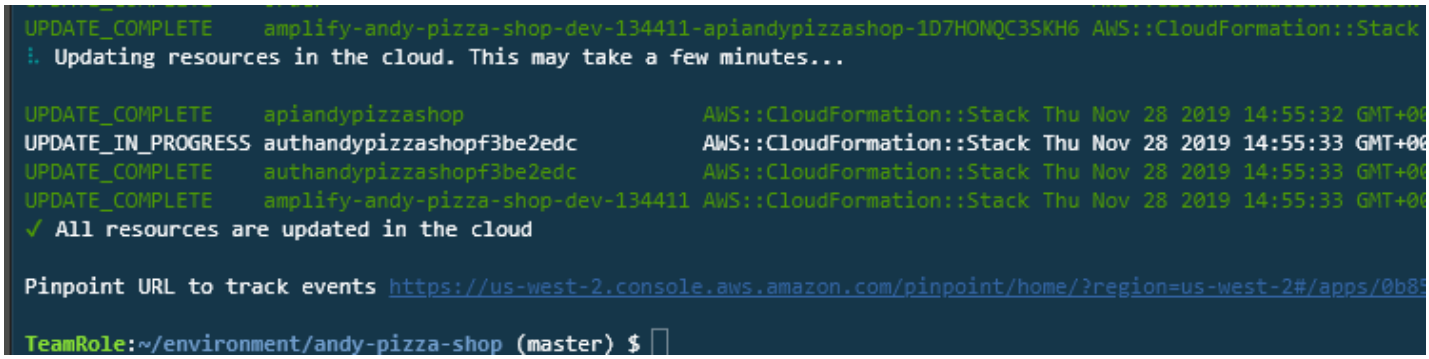
Enter `andyPizzaShop` for the name and **Yes** to allow all users



Use **amplify push** to deploy the Pinpoint service to AWS

```
amplify push
```

When the deployment completes, the terminal will display the pinpoint URL.



Recording analytics in our App

To begin recording events and sending those events to Amazon Pinpoint, we need to add a few lines of code to our App.js file.

Replace the **App.js** file contents with the following 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, Analytics} 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,
      recommendations: 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);
    const analyticsRecord = { name: 'ADD_ITEM', attributes: { SOURCE:
"menu", PRODUCT_ID: item.itemId, ITEM_NAME: item.itemName,
ITEM_CATEGORY: item.category, SIZE: item.size, ORDER: currentOrderId,
USER: this.state.currentUser }, metrics: { QUANTITY: item.quantity,
TOTAL_PRICE: totalPrice, UNIT_PRICE: item.price}};
    Analytics.record(analyticsRecord);
    this.loadExistingOrder(currentOrderId);
  };
```

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

    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;
    };

    getTotalFloat = 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;
    };

    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 totalPrice =
    this.getTotal(this.state.currentOrder.items.items);
      const totalPriceFloat =
    this.getTotalFloat(this.state.currentOrder.items.items);
      const totalItems = this.state.currentOrder.items.items.length;
      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: totalPrice,
        deliveryType: "Carryout",
        deliveryDate: this.state.currentOrder.deliveryDate,
        status: "COMPLETE"
      };

      const analyticsRecord = { name: 'COMPLETE_ORDER', attributes: {
    SOURCE: "checkout", ORDER_TYPE: "Carryout", ORDER:
    this.state.currentOrder.id, USER: this.state.currentUser }, metrics: {
    QUANTITY: totalItems, TOTAL_PRICE: totalPriceFloat}};
      Analytics.record(analyticsRecord);

      API.graphql(
        graphqlOperation(updateOrder, {
          input: orderInput
        })
      ).then(result => {
        this.setState({
          currentOrder: null
        });
        Cache.removeItem("currentOrder");
      });


    };

    loadRecommendations() {
      // Get recommendation

      const getRecos = `
      query getRecos {
        getRecommendations(filter: {
          userId: {
            eq: "${this.state.currentUser}"
          }
        }) {
          items {
            itemId
```

```
            userId
            priority
          }
        }
      }
      `;
      API.graphql(graphqlOperation(getRecos))
        .then(result => {
          var firstResult = result.data.getRecommendations.items[0];
          var filterResult = this.state.menuItems.filter(
            myItem => myItem.productId === firstResult.itemId
          );
          this.setState({
            recommendedItems: result.data.getRecommendations.items,
            recommendations: filterResult
          });
        })
        .catch(err => {
          console.log("RECO ERR", err);
        });

    }

    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"
    });
  };

  handleAddSpecial = rec => {
    var item = {
      itemName: rec.productName,
      quantity: 1,
      price: rec.sizes.items[0].price,
      size: rec.sizes.items[0].size,
      itemId: rec.productId,
      category: rec.category
    };
    this.putTheChatOrder(item, "SPECIAL");
  };

  // ChatBot Helper Functions
  putTheChatOrder = async (item, sourceType) => {
    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);
    const analyticsRecord = { name: 'ADD_ITEM', attributes: { SOURCE:
sourceType, PRODUCT_ID: item.itemId, ITEM_NAME: item.itemName,
ITEM_CATEGORY: item.category, SIZE: item.size, ORDER: currentOrderId,
```

```
    USER: this.state.currentUser }, metrics: { QUANTITY: item.quantity,
    TOTAL_PRICE: totalPrice, UNIT_PRICE: item.price}};
        Analytics.record(analyticsRecord);
        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,
          itemId: pid,
          category: "PIZZA"
        };
        this.putTheChatOrder(item, "CHAT");
        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}
Recommendations={this.state.recommendations} onHandleAddSpecial=
{this.handleAddSpecial} />
                </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 Analytics component from Amplify – `import { Analytics } from 'aws-amplify';`
- Calls the Analytics.record function for when items are added to the cart or order is completed – `Analytics.record({...event info...})`

We also need to make a minor change to **components/menu.jsx**. Replace the contents with the following code and **save menu.jsx**

```
import React, { Component, Fragment } from "react";
import { Tabs, Tab } from "react-bootstrap";
import { Button, Container, Row, Col } from "reactstrap";
import { API, graphqlOperation } from "aws-amplify";

class MenuItem extends Component {
  state = {
    isLoaded: false
  };

  imageLocation = "https://jah-lex-workshop-
2018.s3.amazonaws.com/mob302/images/"

  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
    }
  }
  `;

  getDefaultSizes(menuItems) {
    var sizeSel = [];
    for (var menuItem in menuItems) {
      var cItem = menuItems[menuItem];
      var sizeSelItem = {
        itemId: cItem.productId,
        size: cItem.sizes.items[0].size,
        price: cItem.sizes.items[0].price
      };
      sizeSel.push(sizeSelItem);
    }
    return sizeSel;
  }
  componentDidMount() {
    const limit = {
      limit: 100
    };
    API.graphql(graphqlOperation(this.listProductsWithVariant,
limit)).then(
      result => {
        const sizeSels =
this.getDefaultSizes(result.data.listProducts.items);
        this.setState({
          menuItems: result.data.listProducts.items,
          selectedSize: sizeSels,
          isLoaded: true
        });
```

```
        }
      );
    }

    getSelectedSize(pId) {
      const retVal = this.state.selectedSize.filter(item => item.itemId ===
  pId);
      return retVal[0];
    }
    getPriceForSize(pId, selSize) {
      const retVal = this.state.menuItems.filter(item => item.productId ===
  pId);
      const rVal2 = retVal[0].sizes.items.filter(item2 => item2.size ===
  selSize);
      return rVal2[0].price;
    }
    getPrice(pId) {
      const retVal = this.state.selectedSize.filter(item => item.itemId ===
  pId);
      return retVal[0].price.toFixed(2);
    }
    getItem(itemName, itemId, itemQuantity, itemCategory) {
      const sSize = this.getSelectedSize(itemId);
      const itemSize = sSize.size;
      const itemPrice = sSize.price;
      return {
        itemName: itemName,
        size: itemSize,
        price: itemPrice,
        quantity: itemQuantity,
        itemId: itemId,
        category: itemCategory
      };
    }

    onChangeSize(pid, event) {
      const selSize = event.target.value;
      var currSel = this.state.selectedSize;
      var newSel = [];
      for (var s in currSel) {
        var cItem = currSel[s];
        if (cItem.itemId === pid) {
          cItem = {
            itemId: pid,
            size: selSize,
            price: this.getPriceForSize(pid, selSize)
          };
        }
        newSel.push(cItem);
      }
      this.setState({
```

```
          selectedSize: newSel
        });
      }
      render() {
        return (
          <Fragment>
            <b>Add New Item</b>
            <Tabs defaultActiveKey="profile" id="uncontrolled-tab-example">
              <Tab eventKey="pizza" title="Pizza">
                <Container>
                  {this.state.isLoaded
                    ? this.state.menuItems
                        .filter(fItem => fItem.category === "PIZZA")
                        .map(menuItem => (
                          <Row key={menuItem.productId}>
                            <Col>
                              <img
                                src=
{`${this.imageLocation}${menuItem.productId}${".png"}`}
                                alt="Pizza"
                                width="100"
                                height="100"
                                className="position-relative img-fluid"
                              />
                            </Col>

                            <Col>
                              <b>{menuItem.productName}</b>
                              <br></br>
                              {menuItem.description}
                            </Col>
                            <Col>
                              <select
                                id="menuSize"
                                onChange={this.onChangeSize.bind(
                                  this,
                                  menuItem.productId
                                )}
                              >
                                {menuItem.sizes.items.map(sizeItem => (
                                  <option key={sizeItem.size} value=
{sizeItem.size}>
                                    {sizeItem.size}
                                  </option>
                                ))}
                              </select>
                              <Button
                                onClick={() =>
                                  this.props.onAddItem
                                    ? this.props.onAddItem(
                                        this.getItem(
```

```
                                    `${menuItem.productName}`,
                                    `${menuItem.productId}`,
                                    1,
                                    `${menuItem.category}`
                                  )
                                )
                              : null
                          }
                        >
                          Add To Order
                        </Button>
                      </Col>
                      <Col>$ {this.getPrice(menuItem.productId)}</Col>
                    </Row>
                  ))
              : null}
          </Container>
        </Tab>
        <Tab eventKey="subs" title="Subs">
          <Container>
            {this.state.isLoaded
              ? this.state.menuItems
                  .filter(fItem => fItem.category === "SUB")
                  .map(menuItem => (
                    <Row key={menuItem.productId}>
                      <Col>
                        <img
                          src=
{`${this.imageLocation}${menuItem.productId}${".png"}`}
                          alt="Sub"
                          width="100"
                          height="100"
                          className="position-relative img-fluid"
                        />
                      </Col>

                      <Col>
                        <b>{menuItem.productName}</b>
                        <br></br>
                        {menuItem.description}
                      </Col>
                      <Col>
                        <select
                          id="menuSize"
                          onChange={this.onChangeSize.bind(
                            this,
                            menuItem.productId
                          )}
                        >
                          {menuItem.sizes.items.map(sizeItem => (
                            <option key={sizeItem.size} value=
```

```
    {sizeItem.size}>
                                    {sizeItem.size}
                                  </option>
                                ))}
                              </select>
                              <Button
                                onClick={() =>
                                  this.props.onAddItem
                                    ? this.props.onAddItem(
                                        this.getItem(
                                          `${menuItem.productName}`,
                                          `${menuItem.productId}`,
                                          1,
                                          `${menuItem.category}`
                                        )
                                      )
                                    : null
                                }
                              >
                                Add To Order
                              </Button>
                            </Col>
                            <Col>$ {this.getPrice(menuItem.productId)}</Col>
                          </Row>
                        ))
                    : null}
                </Container>
              </Tab>
              <Tab eventKey="sides" title="Sides">
                <Container>
                  {this.state.isLoaded
                    ? this.state.menuItems
                        .filter(fItem => fItem.category === "SIDE")
                        .map(menuItem => (
                          <Row key={menuItem.productId}>
                            <Col>
                              <img
                                src=
    {`${this.imageLocation}${menuItem.productId}${".png"}`}
                                alt="Side"
                                width="100"
                                height="100"
                                className="position-relative img-fluid"
                              />
                            </Col>

                            <Col>
                              <b>{menuItem.productName}</b>
                              <br></br>
                              {menuItem.description}
                            </Col>
```

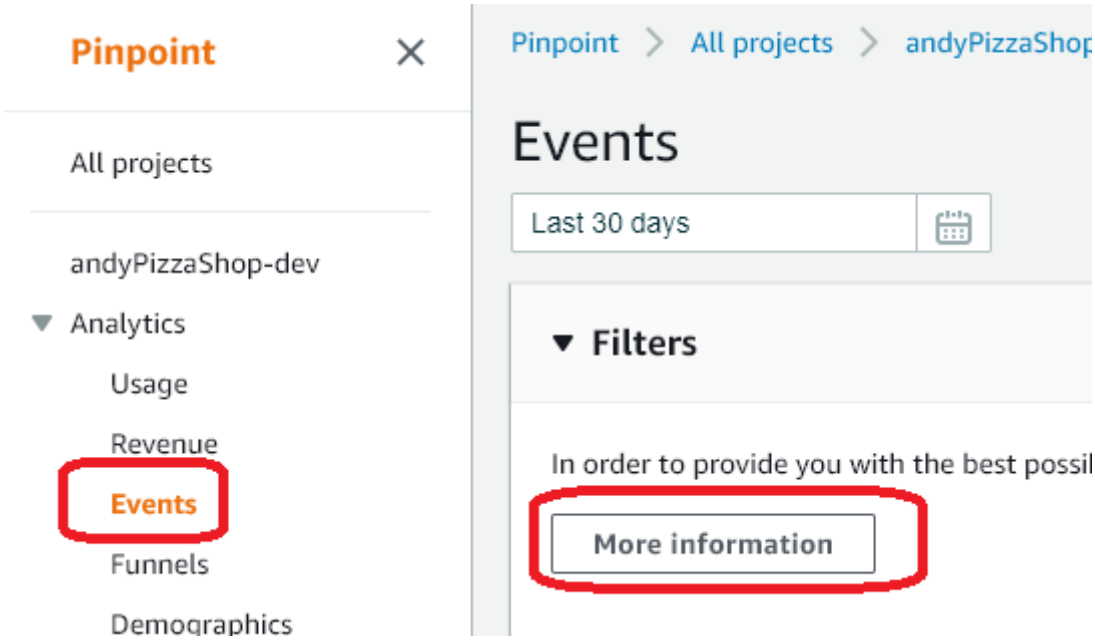
```
                        <Col>
                          <select
                            id="menuSize"
                            onChange={this.onChangeSize.bind(
                              this,
                              menuItem.productId
                            )}
                          >
                            {menuItem.sizes.items.map(sizeItem => (
                              <option key={sizeItem.size} value=
{sizeItem.size}>
                                {sizeItem.size}
                              </option>
                            ))}
                          </select>
                          <Button
                            onClick={() =>
                              this.props.onAddItem
                                ? this.props.onAddItem(
                                    this.getItem(
                                      `${menuItem.productName}`,
                                      `${menuItem.productId}`,
                                      1,
                                      `${menuItem.category}`
                                    )
                                  )
                                : null
                            }
                          >
                            Add To Order
                          </Button>
                        </Col>
                        <Col>$ {this.getPrice(menuItem.productId)}</Col>
                      </Row>
                    ))
                  : null}
              </Container>
            </Tab>
          </Tabs>
        </Fragment>
      );
    }
  }

  export default MenuItem;
```

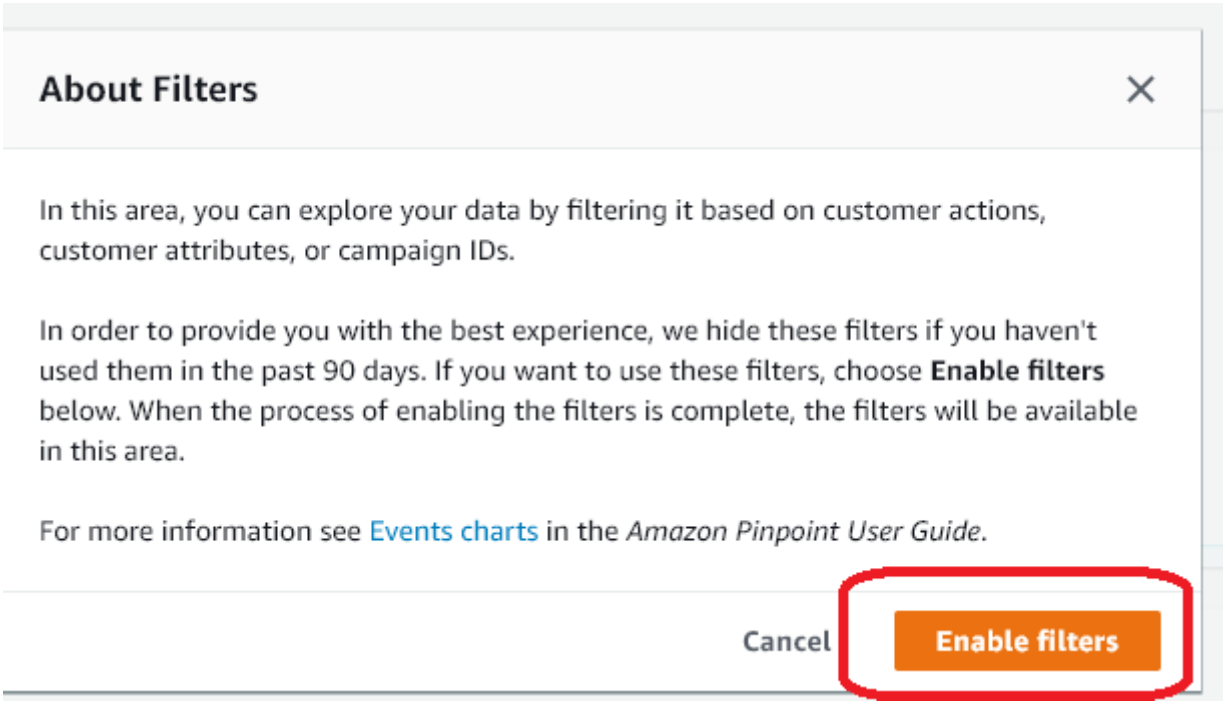
Now start adding items to your cart through all 3 methods: the menu, the chat bot, and the recommended special. After you have completed adding these items, visit the

AWS Pinpoint console.

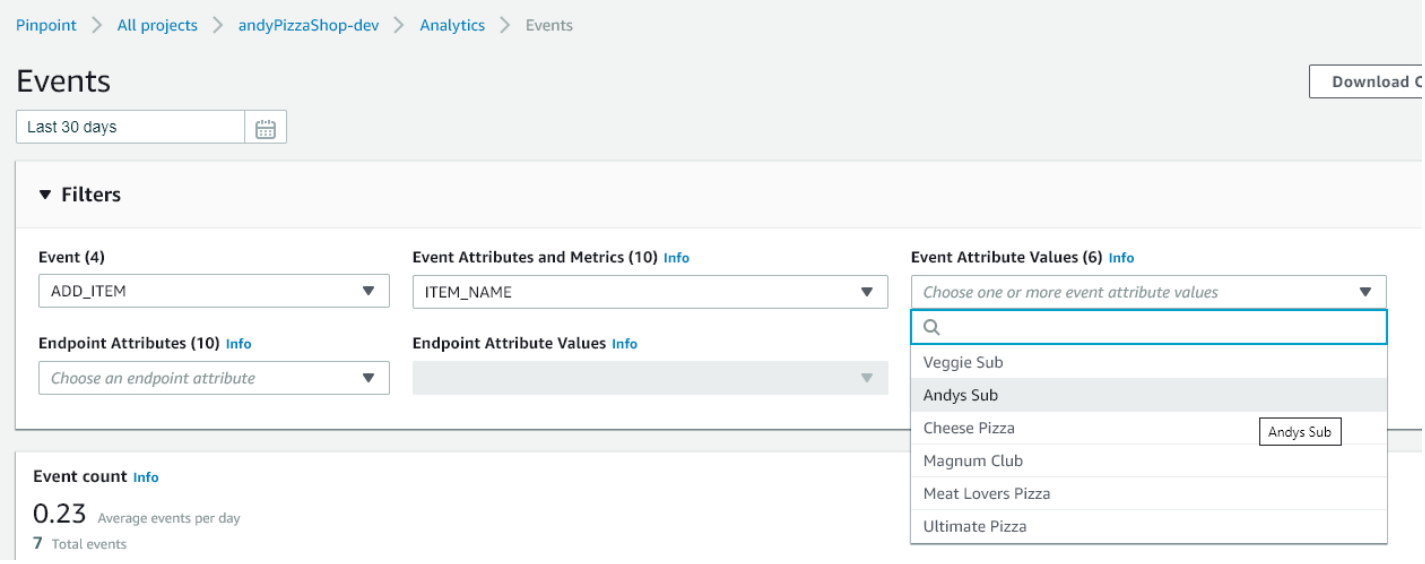
Load the Pinpoint project, expand the **Analytics** section, and then click **Events**. Expand the **Filters** section and click the **More Information** button:



Next, choose to **Enable filters**:



You may need to refresh the Pinpoint console page and wait a few minutes. You should begin seeing your events show up in the filters section.



Valuable analytics are now being collected from our user interactions. With Amazon Pinpoint, you can do things like create targeted messaging campaigns to your end users.