ADD THE CHATBOT TO THE APP

Adding the ChatBot to the application

Now that we have added the chatbot feature to our AWS environment, we can now add it to our application. To do this easily, use the ChatBot component provided by Amplify React.

Replace the contents of **App.js** with the following code and **save App.js**:

```
import React, { Fragment, Component } from "react";
import "./App.css";
import { Container, Row, Col, Button } 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 { createOrder, createItem, updateOrder } from
"./graphql/mutations";
import awsconfig from "./aws-exports";

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"
    });
  };

  // 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!";
  }


  render() {
```

```
    return (
      <Fragment>
        <Header
          onHandleLogin={this.handleLogin}
          onHandleLogout={this.handleLogout}
          onHandleHistory={this.handleHistory}
          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}

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

```
    }
  }

  export default App;
```

The code above does the following:

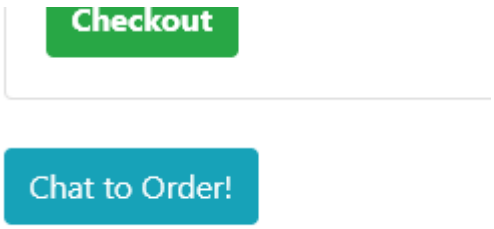
- Imports the ChatBot component – `import { Authenticator, ChatBot } from "aws-amplify-react";`
- Adds the ChatBot component – `<ChatBot title="Place an Order" botName="AndyPizzaOrder_dev" welcomeMessage=.`
- Adds additional helper functions to respond to ChatBot fullfillment – `handleComplete(err, confirmation) {`

Note

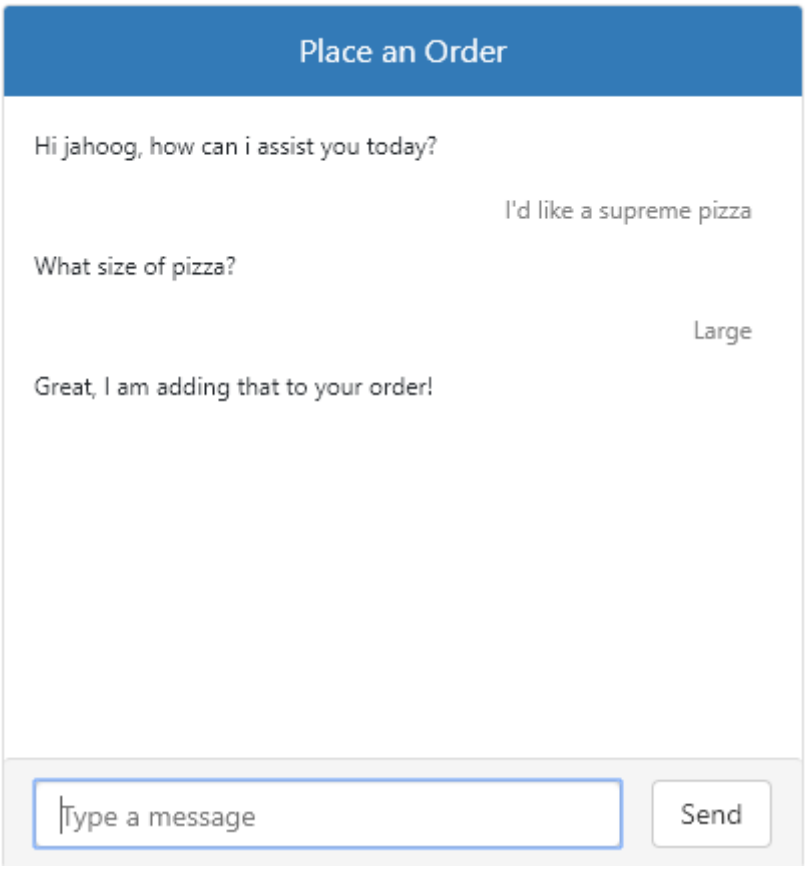
Amazon Lex has back-end code hooks for slot validation and fulfillment. We are not using these in our chatbot, but if you wanted to call other services (such as check inventory), you could do that with these hooks. Learn more at https://docs.aws.amazon.com/lex/latest/dg/programming-model.html

Test the Chatbot

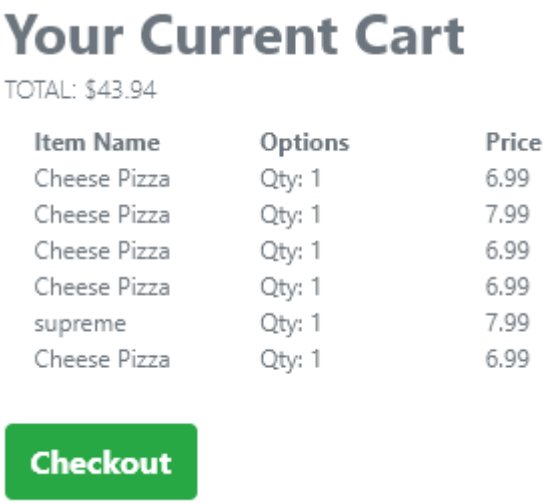
Return to the application preview tab, and click the “Chat to Order” button.



In the chat window, ask for a supreme pizza using the utterance **I’d like a supreme pizza**, and ensure that the ChatBot responds with the slot request to get the size:



Type **Large** and confirm that the item was added to your order:



You can also try other pizza types such as **Ultimate**, or just simply type **I'd like a pizza** to get a cheese pizza.

Bingo! You have a chatbot powered by Amazon Lex's natural language understanding!