Customer Agent

Your task is to create a LLM based tester agent that mimics customers placing orders with Lilac’s order-taking agent (the endpoint provided via text-to-text).

Back story: Quality and representative labeled data is required to evaluate and fine-tune LLM. But we have limited customer interactions and labeling them is costly. Simulation is an attentive way to collect such data. We can control the direction of the conversation tree, complicity of the orders, and more. This project is to build the simplest version of that.

DETAILS

Provided:

- **Menu in JSON Format**: Provides the items, their options (required/optional), and possible modifiers.
- **API Endpoints (find details in page 8)**:
    - Start an order.
    - Send chat messages to Lilac’s order-taking agent. Return the response and the current order.
    - Retrieve the order.

Requirement:

1. **Order Generation**:
    a. Create realistic order goals for your customer agent derived from the JSON menu.
    b. Include required and optional options for each item.
    c. Cover a range of scenarios:
        i. Simple orders: A single item with minimal customization.
        ii. Medium complexity: A meal with sides and drinks.
        iii. Complex orders: Multiple items, customizations, and modifications.

    For example:

    Goal:

    - Order one “Classic Hot Dog” with no onions meal with regular fires and coke
    - Order another “Classic Hot Dog” just the hotdog (a la carte)
    - Order one large “Cherry Coke”
    - Add ketchup to the a la carte “Classic Hot Dog”

    Final order that the lilac endpoint returns should look like:

```
[
 {
   itemName: 'Classic Hot Dog',
   optionKeys: ['customizations', 'meal option', 'side options', 'drink options'],
   optionValues: [['no onions'], ['meal'], ['regular fries'], ['coke']],
 },
 {
   itemName: 'Classic Hot Dog',
   optionKeys: ['customizations', 'meal option', 'side options', 'drink options'],
   optionValues: [['add ketchup'], ['a la carte'], [], []],
 },
 {
   itemName: 'Cherry Coke',
   optionKeys: ['size'],
   optionValues: [['large']],
 },
];
```

Notes:

- An order contains > 0 items. Each item has their perspective options, some are optional, some are required. (see menu for more detailed metadata)

- It is possible `optionKeys` with empty options don’t show up on the `order` returned from the API
- Ensure all generated orders meet the menu’s required and optional options.

Here’s definition of each item in orders

```
{
     type: 'object',
     properties: {
       itemName: {
         type: 'string',
         enum: allItems,
        description: 'The name of the food item to add to order',
      },
      optionKeys: {
        type: 'array',
        description: 'Keys (the option names) of Required options for this item. Default is an empty array.',
        items: {
          type: 'string',
        },
        default: [],
      },
      optionValues: {
        type: 'array',
        description:
          'Selected values of Required options for this item. This should match the index of requiredOptionKeys and have the same length. Default is an array of empty arrays.',
        items: {
          type: 'array',
          items: {
            type: 'array',
            items: {
              type: 'string',
            },
          },
        },
        default: [],
      },
    },
    required: ['itemName', 'optionKeys', 'optionValues'],
  },
```

2. **Conversation Simulation**:
    a. Mimic natural conversations with Lilac’s agent, including:
        i. Correct order mistakes based on the returned order state
        ii. Asking about items or options.

iii. Adding items to the order.
iv. Removing or modifying items.

For example:

```
[
 {
   role: 'assistant',
   content: "Welcome to Ben Frank's! How may I help you today?",
 },
 {
   role: 'user',
   content: 'I want a classic hot dog with no onions',
 },
 {
   role: 'assistant',
   content: 'As a meal?',
 },
 {
   role: 'user',
   content: 'yes, with coke',
 },
 {
   role: 'assistant',
   content: 'Added your Classic Hot Dog meal with no onions and a Coke. Anything else?',
 },
 {
   role: 'user',
   content: 'another classic hotdog ',
 },
 {
   role: 'assistant',
   content: 'With everything on it?',
 },
 {
   role: 'user',
   content: 'yeah',
},
{
  role: 'assistant',
  content: 'As a meal?',
},
{
  role: 'user',
  content: 'no',
},
{
  role: 'assistant',
  content: 'Added another Classic Hot Dog. Anything else?',
},
{
  role: 'user',
  content: 'I will have a Cherry Coke',
},
{
  role: 'assistant',
  content: 'Medium okay?',
},
{
  role: 'user',
  content: 'large',
},
{
  role: 'assistant',
  content: 'Added your Cherry Coke. Anything else?',
},
{
  role: 'user',
  content: 'could you add ketchup to the second hotdog?',
},
{
  role: 'assistant',
  content: 'Ketchup added to the second hotdog. Anything else?',
 },
 {
   role: 'user',
   content: 'no',
 },
 {
   role: 'assistant',
   content: 'Your total is xyz, please pull forward.',
 },
]
```

Key Capabilities:

1. **Order Generation**:
    a. Dynamically generate order goals based on the menu structure.
    b. Handle varying levels of order complexity.
2. **Natural Conversation Simulation**:
    a. Simulate diverse customer behaviors with varying levels of detail and interaction.
3. **Order Management**:
    a. Correct order state. The /chat endpoint returns the current order state. If this differs from what your agent expects, correct it. (imagine this is the order confirmation board at the drive thru)
    b. Support actions to:
        i. Add items.
        ii. Remove items.
        iii. Modify existing items.

API ENDPOINTS

1. Start new order

Initializes a new chat session for a given location with an empty order and chat state.

`POST` https://test.lilaclabs.ai/lilac-agent/start

Header

```
{
  x-api-key: Bearer <api_token>
}
```

Request Body

```
{
  location: "ben-franks"
}
```

Response

```
{
  orderId: string
}
```

2. Send Chat Message

Generate the next response and update the order accordingly based on your input

`POST` https://test.lilaclabs.ai/lilac-agent/chat

Header

```
{
  x-api-key:
Bearer
<api_token>
}
```

Request Body

```
{
  orderId: string,
  input: string,
  location:
"ben-franks"

}
```

Response

```
{
  orderId: string,
  order:
Array<{
 itemName: string;
 optionKeys: string[];
 optionValues: string[][];
 id: string;
 price: number;
}>
  messages:
Array<{ role: 'user' | 'assistant';
content: string }>

}
```

3. Retrieve Order

Retrieves the current state of the order and associated chat messages for a given orderId.

`GET https://test.lilaclabs.ai/lilac-agent/order/{orderId}`

Header

```
{
  x-api-key: Bearer
<api_token>
}
```

Response

```
{
  order:
Array<{
 itemName: string;
 optionKeys: string[];
 optionValues: string[][];
 id: string;
 price: number;
}>
  messages:
Array<{ role: 'user' | 'assistant'; content: string }>

}
```

WHAT WE ARE LOOKING FOR

Description	Weight
runnable	10 %
code quality and cleanliness	
Readability	15 %
Design pattern / data structure	10 %
concurrency	5 %
docs on all public functions / design docs	10 %
good visual outputs to see what is going on	5 %
functionality	
Order goal generation	
output varying levels of order complexity	3 %
Handle options correctly - customization with modifiers - Ie. no onion, extra ketchup, add bacon - …	5 %
Generate multi step goals - Actions: set(add, remove, modify) - Ie. [add, add, modify, remove, add…]	7 %
Conversation simulation	
Simulate diverse customer behaviors with varying levels of detail and interaction.	10 %
Support multi steps - Actions: set(add, remove, modify) - Ie. [add, add, modify, remove, add…]	20 %
bonus: support order correction during the ordering process. when the output order state mismatches your agent's exceptions. This does not happen very often, but happens.	10 %
Total	110

TIMELINES & EXPECTATIONS

- This take-home requires full-time, intensive effort. We generally hear back from folks within 48 hours.
- You can use any programming language, framework, and IDE you’d like;