Airline-Bot Fulfillment Lambda

This directory contains the AWS Lambda fulfillment function for the Airline-Bot, built using the lex_helper framework. The Lambda function handles all intent processing, slot elicitation, and business logic for the conversational AI airline service bot.

Overview

The fulfillment Lambda serves as the backend for an Amazon Lex V2 chatbot that provides airline services including:

- Flight booking with authentication flow
- Flight status and delay updates
- Baggage tracking
- Flight cancellations and changes
- General customer service interactions

Architecture

The Lambda function uses the lex_helper framework for structured intent management and follows a modular architecture with separate handlers for each intent.

Directory Structure

```
fulfillment_function/
├── lambda_function.py          # Main Lambda entry point and request
router
├── session_attributes.py       # Custom session attributes definition
├── intents/                    # Intent handlers directory
│   ├── __init__.py
│   ├── authenticate.py         # User authentication handler
│   ├── book_flight.py          # Flight booking with slot elicitation
│   ├── cancel_flight.py        # Flight cancellation handler
│   ├── change_flight.py        # Flight modification handler
│   ├── flight_delay_update.py  # Flight status and delay information
│   ├── track_baggage.py        # Baggage tracking handler
│   ├── greeting.py             # Welcome and greeting handler
│   ├── goodbye.py              # Farewell handler
│   ├── anything_else.py        # Additional assistance handler
│   └── fallback_intent.py      # Fallback for unrecognized inputs
├── utils/                      # Utility modules
│   ├── __init__.py
│   ├── enums.py              # Enumerations for invocation sources and
statuses
│   └── config.py             # Configuration utilities for paths and
resources
├── messages/                   # Internationalized message files
│   ├── messages_en_US.yaml   # English (US) messages
│   └── messages_it_IT.yaml    # Italian messages
└── README.md                  # This file
```

Key Components

Main Lambda Handler (`lambda_function.py`)

- Entry point for all Lex requests
- Initializes lex_helper framework with custom session attributes
- Routes requests to appropriate intent handlers
- Handles errors and provides fallback responses

Session Attributes (`session_attributes.py`)

- Defines custom attributes that persist across conversation turns
- Includes flight booking data, authentication state, and error tracking
- Extends base SessionAttributes from lex_helper framework

Intent Handlers (`intents/`)

Each intent handler follows a consistent pattern:

- **Dialog Hook**: Manages slot elicitation and validation
- **Fulfillment Hook**: Processes completed intents and performs business logic
- **Main Handler**: Routes between dialog and fulfillment based on invocation source

Key Features Demonstrated

1. **Authentication Flow**: BookFlight intent shows how to redirect to authentication and return to original intent
2. **Slot Elicitation**: Systematic collection of required information with custom prompts
3. **Error Handling**: Graceful handling of unknown inputs and system errors
4. **Intent Transitions**: Moving between intents for complex workflows
5. **Session Management**: Persistent data across conversation turns

Local Development

Prerequisites

- Python 3.12+ (to match Lambda runtime)
- lex_helper framework (v0.0.14 or later)

Setup

1. **Install Dependencies**:

   ```
   # Create the layer directory structure
   mkdir -p layers/lex_helper/python
   ```

```
# Extract the lex-helper package (download from
https://gitlab.aws.dev/lex/lex-helper)
unzip layers/lex-helper-v*.zip -d layers/lex_helper/python
```

2. **Local Path Configuration**:
   The `lambda_function.py` automatically adds the layer to Python path when running locally (detected by absence of `AWS_EXECUTION_ENV` environment variable).

3. **Testing**:

```
# Run individual intent handlers for testing
python -m intents.book_flight

# Test the main lambda handler
python lambda_function.py
```

Development Guidelines

- Follow the established pattern for new intent handlers
- Use the lex_helper framework functions for consistent behavior
- Add comprehensive logging for debugging
- Handle both dialog and fulfillment invocation sources
- Store relevant data in session attributes for multi-turn conversations

Message Configuration

The bot supports internationalization through YAML message files located in the `messages/` directory. The configuration system:

1. **Automatically detects environment**: Works in both Lambda and local development
2. **Leverages MessageManager's search paths**: Uses the built-in search capabilities of the lex_helper framework
3. **Supports environment variable override**: Set `MESSAGES_DIR` to customize the messages directory location
4. **Handles multiple locales**: Currently supports `en_US` and `it_IT`

To add a new locale:

1. Create a new file in the `messages/` directory named `messages_LOCALE_CODE.yaml`
2. Add the locale code to the supported locales list in `lambda_function.py`
3. Translate all message keys from an existing locale file

The message configuration is handled by the `utils/config.py` module, which:

- In Lambda: Uses the default paths (messages are deployed to `/var/task`)
- In local development: Adds the project directory to the Python path

- With environment variable: Uses the custom path specified in `MESSAGES_DIR`

This approach works seamlessly with the MessageManager from the lex_helper framework, which searches for message files in multiple locations including the Python path and current working directory.

Deployment

Using CloudFormation (Recommended)

```
cd <PROJECT_ROOT>/cloudformation
./deploy_airline_bot.sh [bot-alias-name]
```

This automated deployment:

1. Packages the lex_helper layer
2. Packages the Lambda function code
3. Packages the Lex bot configuration
4. Deploys everything via CloudFormation

Manual Deployment

1. **Package Dependencies**:

   ```
   cd layers/lex_helper
   zip -r ../../lex-helper-layer.zip python/
   ```

2. **Package Lambda Function**:

   ```
   cd fulfillment_function
   zip -r ../fulfillment-function.zip . -x "*.pyc" "__pycache__/*"
   ```

3. **Deploy via AWS CLI or Console**

Configuration

Environment Variables

- `AWS_EXECUTION_ENV`: Set by Lambda runtime (used to detect local vs. Lambda execution)
- `MESSAGES_DIR`: (Optional) Custom path to the messages directory

IAM Permissions

The Lambda function requires:

- Basic Lambda execution role

- CloudWatch Logs permissions for logging

Lex Integration

- Configure the Lex bot to use this Lambda as the fulfillment function
- Enable code hooks for intents that require dialog management
- Set appropriate timeout values (recommended: 30 seconds)

Testing

Test Events

Use the provided test events in the CloudFormation directory:

```
aws lambda invoke --function-name AirlineBotFulfillment \
  --payload file://test-event.json output.json
```

Integration Testing

Test through:

- Amazon Lex console test interface
- AWS CLI `recognize-text` commands
- Integrated messaging platforms (if configured)

Production Considerations

Performance

- Lambda cold start optimization through provisioned concurrency if needed
- Efficient session attribute management
- Proper error handling to prevent timeouts

Security

- Input validation and sanitization
- Secure handling of authentication data
- Proper logging without exposing sensitive information

Monitoring

- CloudWatch metrics and alarms
- Custom metrics for business logic
- Structured logging for debugging

Integration Points

The current implementation uses mock data. For production:

- Replace authentication logic with real identity providers
- Integrate booking system APIs
- Connect to flight tracking services
- Implement real baggage tracking systems

Contributing

When adding new intents or modifying existing ones:

1. Follow the established handler pattern
2. Add comprehensive documentation
3. Include appropriate error handling
4. Test both dialog and fulfillment flows
5. Update this README if adding new functionality

License

This project is licensed under the MIT License - see the LICENSE file for details.