Airline-Bot

A comprehensive conversational AI chatbot for airline services built with Amazon Lex V2 and AWS Lambda. This open-source project demonstrates best practices for building sophisticated airline service chatbots using the lex_helper framework, featuring flight bookings, cancellations, status updates, baggage tracking, and user authentication flows.

🚀 Features

The Airline-Bot provides a natural language interface for airline customers to:

- **Book flights** with authentication and comprehensive slot elicitation
- **Cancel existing reservations** with confirmation workflows
- **Change flight details** with validation and rebooking
- **Check flight status and delays** with real-time information
- **Track baggage** with status updates
- **User authentication** with callback flow demonstration
- **Multi-turn conversations** with persistent session state
- **Error handling** with graceful fallbacks and user-friendly messages

🏗️ Architecture

The bot is built using:

- **Amazon Lex V2** for natural language understanding and conversation management
- **AWS Lambda** for fulfillment logic and business processing
- **lex_helper Framework** for structured intent management and reduced boilerplate
- **CloudFormation** for infrastructure as code deployment
- **Modular Design** for easy extension and maintenance

📁 Project Structure

```
Airline-Bot/
├── cloudformation/                    # Infrastructure and deployment
│   ├── scripts/                       # Deployment automation scripts
│   │   ├── package-lambda-function.sh
│   │   ├── package-lex-helper-layer.sh
│   │   └── package-lex-export.sh
│   ├── deploy_airline_bot.sh          # Main deployment script
│   └── airline-bot-native-template.yaml # CloudFormation template
├── lambdas/                           # Lambda function code
│   └── fulfillment_function/          # Main fulfillment Lambda
│       ├── lambda_function.py         # Entry point and request router
│       ├── session_attributes.py      # Custom session state management
│       ├── intents/                   # Intent handlers
│       │   ├── authenticate.py        # User authentication flow
│       │   ├── 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 delays
│       │   ├── track_baggage.py       # Baggage tracking
│       │   ├── greeting.py            # Welcome interactions
│       │   ├── goodbye.py             # Farewell handling
│       │   ├── anything_else.py       # Additional assistance
│       │   └── fallback_intent.py     # Unrecognized input handling
│       └── utils/                     # Utility modules
│           └── enums.py               # Constants and enumerations
├── layers/                            # Lambda layers for dependencies
│   └── lex-helper-v*.zip             # lex_helper framework package
├── lex-export/                        # Lex bot configuration
│   └── LexBot/                        # Bot definition and intents
├── zip/                               # Generated deployment packages
├── DEPLOYMENT_GUIDE.md                # Detailed deployment
instructions
└── README.md                          # This file
```

🔧 Key Components

Lex Helper Framework Integration

This project showcases the **lex_helper framework**, a powerful toolkit that simplifies Amazon Lex chatbot development:

- **Structured Intent Management**: Organized handlers with consistent patterns
- **Type-Safe Session Attributes**: Pydantic models for conversation state
- **Automated Request/Response Handling**: Reduced boilerplate code
- **Channel-Aware Response Formatting**: Consistent messaging across platforms
- **Simplified Dialog State Management**: Easy slot elicitation and validation
- **Error Handling**: Built-in patterns for graceful error recovery

The lex-helper library can be downloaded from https://gitlab.aws.dev/lex/lex-helper

Fulfillment Lambda Architecture

The Lambda function demonstrates production-ready patterns:

- **Modular Intent Handlers**: Each intent in its own module with clear separation of concerns
- **Dialog vs. Fulfillment Hooks**: Proper handling of both dialog management and final processing
- **Session State Management**: Persistent data across conversation turns
- **Authentication Flow**: Complete authentication with callback to original intent
- **Error Handling**: Comprehensive error management with user-friendly responses
- **Logging**: Structured logging for debugging and monitoring

Demonstrated Patterns

1. **Multi-Turn Conversations**: Complex booking flow with multiple slot elicitation

2. **Intent Transitions**: Moving between intents (e.g., authentication flow)
3. **Session Management**: Persistent user data and conversation state
4. **Error Recovery**: Handling unknown inputs and system errors
5. **Production Readiness**: Proper logging, error handling, and deployment automation

🚀 Quick Start

1. Clone the repository
2. Set up dependencies:

```
mkdir -p layers/lex_helper/python
# Download lex-helper from https://gitlab.aws.dev/lex/lex-helper
unzip lex-helper-v*.zip -d layers/lex_helper/python
```

3. Deploy:

```
./deploy_airline_bot.sh
```

Prerequisites

- **AWS CLI** installed and configured with appropriate permissions
- **Python 3.12+** (to match Lambda runtime)
- **S3 bucket** for storing deployment artifacts
- **IAM permissions** for Lambda, Lex, and CloudFormation operations

1. Clone and Setup

```
git clone <repository-url>
cd Airline-Bot

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

# Download and extract lex-helper framework
# (Download lex-helper-v*.zip from https://gitlab.aws.dev/lex/lex-
helper)
unzip layers/lex-helper-v*.zip -d layers/lex_helper/python
```

2. Deploy to AWS

Option A: One-Command Deployment (Recommended)

```
cd cloudformation
./deploy_airline_bot.sh [optional-bot-alias-name]
```

Option B: Step-by-Step Deployment

```
cd cloudformation/scripts

# Package components
./package-lex-helper-layer.sh
./package-lambda-function.sh
./package-lex-export.sh

# Deploy
cd ..
./deploy_airline_bot.sh [optional-bot-alias-name]
```

3. Test Your Bot

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

# Test through Lex console or CLI
aws lexv2-runtime recognize-text \
  --bot-id <bot-id> \
  --bot-alias-id <alias-id> \
  --locale-id en_US \
  --session-id test-session \
  --text "I want to book a flight"
```

💻 Local Development

Setup Development Environment

```
# The lambda_function.py automatically detects local environment
# and adds the lex_helper layer to Python path

# For testing individual components:
cd lambdas/fulfillment_function
python -c "from intents.book_flight import handler; print('Import
successful')"
```

Development Guidelines

- **Follow Established Patterns**: Use existing intent handlers as templates
- **Comprehensive Logging**: Add debug logging for troubleshooting
- **Error Handling**: Always include try-catch blocks and user-friendly error messages
- **Documentation**: Add docstrings and inline comments
- **Session Management**: Store relevant data in session attributes for multi-turn conversations

Testing Locally

```
# Test individual intent handlers
python -m intents.book_flight

# Test session attributes
python -c "from session_attributes import AirlineBotSessionAttributes;
print(AirlineBotSessionAttributes())"
```

🔧 Configuration

Environment Variables

- `AWS_EXECUTION_ENV`: Automatically set by Lambda runtime (used for local vs. Lambda detection)

Required AWS Permissions

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

Lex Bot Configuration

- **Runtime**: Amazon Lex V2
- **Language**: English (US)
- **Fulfillment Lambda**: Point to deployed Lambda function
- **Session Timeout**: 300 seconds (recommended)
- **Code Hooks**: Enabled for dialog management

🧪 Testing

Automated Testing

```
# Test deployment
cd cloudformation
./test-deployment.sh

# Test individual intents
aws lambda invoke --function-name AirlineBotFulfillment \
  --payload '{"inputTranscript":"book a flight","sessionId":"test"}' \
  response.json
```

Manual Testing

- **Lex Console**: Use the built-in test interface
- **AWS CLI**: Use `recognize-text` commands
- **Integration**: Test with messaging platforms if configured

Test Scenarios

1. **Complete Booking Flow**: "I want to book a flight" → authentication → slot filling → confirmation
2. **Flight Status**: "What's the status of flight AA123?"
3. **Error Handling**: Invalid inputs and system errors
4. **Authentication Flow**: Protected intents requiring authentication

🚀 Production Deployment

Performance Optimization

- **Provisioned Concurrency**: For consistent response times
- **Memory Configuration**: Optimize based on usage patterns
- **Timeout Settings**: Set appropriate timeouts for external API calls

Security Best Practices

- **Input Validation**: Sanitize all user inputs
- **Authentication**: Implement proper user authentication
- **Logging**: Avoid logging sensitive information
- **IAM Roles**: Follow principle of least privilege

Monitoring and Observability

- **CloudWatch Metrics**: Monitor Lambda performance and errors
- **Custom Metrics**: Track business-specific metrics
- **Alarms**: Set up alerts for error rates and performance issues
- **Structured Logging**: Use consistent log formats for analysis

Integration Points for Production

The current implementation uses mock data. For production deployment:

1. **Authentication System**: Replace mock authentication with real identity providers (OAuth, SAML, etc.)
2. **Booking APIs**: Integrate with airline reservation systems
3. **Flight Data**: Connect to real-time flight tracking services
4. **Payment Processing**: Add secure payment handling
5. **Customer Database**: Integrate with customer management systems
6. **Notification Services**: Add email/SMS confirmations

📚 Documentation

- **Fulfillment Lambda README**: Detailed Lambda function documentation
- **Lex Bot Export README**: Amazon Lex bot configuration and structure
- **Deployment Guide**: Comprehensive deployment instructions
- **Lambda Layers README**: Working with Lambda layers

🤝 Contributing

We welcome contributions! Please follow these guidelines:

1. **Code Quality**: Follow the established patterns and include comprehensive documentation
2. **Testing**: Test both dialog and fulfillment flows for any changes
3. **Documentation**: Update relevant README files and inline documentation
4. **Error Handling**: Include proper error handling and logging
5. **Consistency**: Maintain consistent code style across the project

Adding New Intents

1. Create new handler in `lambdas/fulfillment_function/intents/`
2. Follow the established pattern (dialog_hook, fulfillment_hook, main handler)
3. Add comprehensive documentation and error handling
4. Update session attributes if needed
5. Test thoroughly and update documentation

📄 License

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

🙏 Acknowledgments

- **lex_helper Framework**: For simplifying Amazon Lex development
- **Amazon Lex Team**: For providing excellent conversational AI capabilities
- **AWS Lambda Team**: For serverless computing platform
- **Open Source Community**: For inspiration and best practices

📞 Support

For questions, issues, or contributions:

- **Issues**: Use GitHub Issues for bug reports and feature requests
- **Documentation**: Check the comprehensive documentation in each directory
- **Examples**: Review the intent handlers for implementation patterns

---