

Ask GroupFund - Complete Documentation

Version: 1.0

Last Updated: December 2024

Table of Contents

- Overview
- Architecture
- Installation & Setup
- User Guide
- API Documentation
- Configuration
- Deployment
- Developer Guide
- Troubleshooting
- Security

Overview

Ask GroupFund is an intelligent AI-powered chatbot system that provides customer service support via SMS (Twilio) and web interface. The system uses RAG (Retrieval Augmented Generation) to answer questions based on a knowledge base of past customer interactions, documentation, and support materials.

Key Features

- Multi-Channel Support: SMS via Twilio and web-based chat interface
- RAG-Powered Responses: Semantic search with LLM-generated answers
- Knowledge Base Management: Web GUI for uploading and processing training data
- Multiple File Formats: Supports .eml, .docx, .pdf, .csv, .txt, .json, .xml
- Role-Based Access Control: Admin, Internal, Sales Rep, and Customer roles

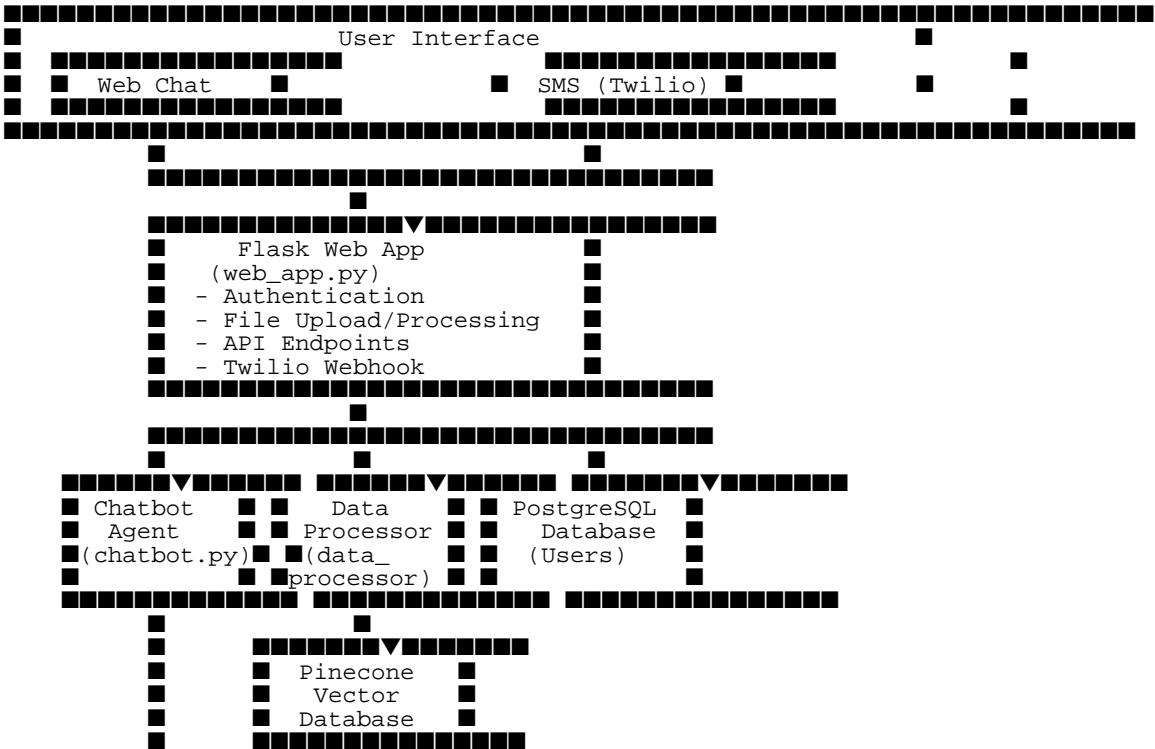
- Audience Labeling: Tag training data for specific audiences
- Conversation History: Maintains context across multiple messages
- Source Attribution: Shows which documents were used to generate responses
- GitLab Integration: Automated ingestion of commits, READMEs, and release notes
- FAQ Analysis: Automatically extracts frequently asked questions

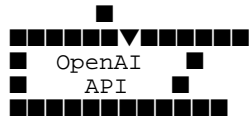
Technology Stack

- Backend: Python 3.9+, Flask
- Vector Database: Pinecone (serverless)
- LLM: OpenAI GPT-4
- SMS: Twilio
- Database: PostgreSQL (for user management)
- Deployment: Railway
- Frontend: HTML, CSS, JavaScript (Bootstrap)

Architecture

System Components





Data Flow

- Training Data Ingestion:
 - Files uploaded via web UI → data_processor.py
 - Files parsed and chunked → Embedded using OpenAI
 - Embeddings stored in Pinecone with metadata
- Query Processing:
 - User question → chatbot.py
 - Semantic search in Pinecone → Retrieve relevant context
 - Context + question → OpenAI GPT-4 → Generate response
 - Response + source attribution → User
- SMS Flow:
 - SMS received → Twilio webhook → /sms endpoint
 - Processed through chatbot → Response sent via Twilio

Key Files

- web_app.py: Main Flask application, API endpoints, authentication
- chatbot.py: RAG system, semantic search, response generation
- data_processor.py: File processing, chunking, embedding generation
- templates/index.html: Web UI template
- static/js/app.js: Frontend JavaScript
- static/css/style.css: Styles
- scripts/gitlab_connector.py: GitLab integration

Installation & Setup

Prerequisites

- Python 3.9 or higher
- Twilio account with phone number

- OpenAI API key
- Pinecone account and API key
- PostgreSQL database (for production)
- GitLab access token (optional, for GitLab integration)

Local Development Setup

- Clone the repository:

```
git clone https://github.com/evonow/twilio-chatbot.git
cd twilio-chatbot
```

- Create virtual environment:

```
python3 -m venv venv
source venv/bin/activate # On Windows: venv\Scripts\activate
```

- Install dependencies:

```
pip install -r requirements.txt
```

- Create .env file:

```
cp .env.example .env
```

- Configure environment variables (see Configuration section)

- Initialize Pinecone index:

- Go to <https://app.pinecone.io>
- Create a new index:
- Name: customer-service-kb
- Dimensions: 1536 (OpenAI text-embedding-3-small)
- Metric: cosine
- Pod Type: serverless (recommended)

- Start the application:

```
python web_app.py
```

- Access the web interface:

- Open browser to <http://localhost:5001>
- Login with PIN 0000 (default admin)

Production Setup (Railway)

See Deployment section for detailed Railway deployment instructions.

User Guide

Login

- Navigate to the web interface
- Enter your 4-digit PIN
- The system will auto-login when 4 digits are entered
- Default admin PIN: 0000

User Roles

- Admin: Full access to all features, user management
- Internal: Access to all data (no restrictions)
- Sales Rep: Access to sales rep labeled data only
- Customer: Access to customer labeled data only

Uploading Training Data

- Navigate to "Upload Files" section (Admin only)
- Select files:
 - Supported formats: .eml, .docx, .pdf, .csv, .txt, .json, .xml
 - Drag and drop or click to browse
 - Multiple files can be selected
- Select audience labels:
 - Check one or more: Sales Reps, Customers, Internal Use
 - Documents can be labeled for multiple audiences
- Click "Process Files":
 - Files are uploaded and queued for processing
 - Processing happens in the background
 - Check "Processing Status" section for progress
- View processed files:
 - Go to "Uploaded Files" section
 - Filter by file type or audience
 - See processing status for each file

Using the Chat Interface

- Navigate to "Ask GroupFund" section
- Enter your question in the text area
- Press Enter or click "Ask" button
- View response:
- Answer appears in the chat interface
- Source attribution shows which documents were used
- Conversation history is maintained

Managing Users (Admin Only)

- Navigate to "User Management" section
- Add new user:
- Enter PIN (4 digits)
- Enter name
- Select role
- Click "Save User"

- Delete user:
- Click "Delete" button next to user
- Confirm deletion

GitLab Integration (Admin Only)

- Navigate to "GitLab Integration" section
- Enter GitLab details:
- Repository URL (e.g., <https://gitlab.com/username/repo>)
- Access token (with read_api scope)
- Select content types: Commits, READMEs, Release Notes

- Click "Ingest Repository"
- Monitor progress in Processing Status section

Knowledge Base Statistics

View comprehensive statistics in the header:

- Total document count
- Top document types
- Audience breakdown
- Customer service interactions
- Release notes features
- GitLab documents

API Documentation

Authentication

Most endpoints require authentication via session cookie. Admin endpoints require Admin role.

Web Application Endpoints

GET /

- Description: Main web interface
- Authentication: Required
- Response: HTML page

GET /login

- Description: Login page
- Authentication: Not required
- Response: HTML login page

POST /login

- Description: Authenticate user
- Authentication: Not required
- Body: {"pin": "1234"}
- Response: Redirect to main page or error

POST /logout

- Description: Logout user
- Authentication: Required
- Response: Redirect to login page

GET /api/auth/status

- Description: Get current user authentication status
- Authentication: Required
- Response:

```
{
  "authenticated": true,
  "user_name": "John Doe",
  "user_role": "Admin"
}
```

POST /api/upload

- Description: Upload files for processing
- Authentication: Required (Admin)
- Body: Multipart form data with files
- Response:

```
{
  "success": true,
  "files": ["file1.eml", "file2.pdf"],
  "message": "Files uploaded successfully"
}
```

POST /api/process

- Description: Process uploaded files
- Authentication: Required (Admin)
- Body:

```
{
  "files": ["file1.eml"],
  "audience": "sales_reps,customers"
}
```

- Response:

```
{
  "success": true,
  "message": "Processing started"
}
```

POST /api/process-all

- Description: Process all uploaded files
- Authentication: Required (Admin)
- Body:

```
{
  "audience": "sales_reps,customers"
}
```

- Response:

```
{
  "success": true,
  "message": "Processing all files"
}
```


POST /api/query

- Description: Query the chatbot
- Authentication: Required
- Body:

```
{
  "query": "How do I reset my password?",
  "audience": "customers"
}
```

- Response:

```
{
  "response": "To reset your password...",
  "sources": [
    {
      "source": "email_123.eml",
      "relevance": 0.95
    }
  ],
  "conversation_id": "conv_123"
}
```

GET /api/stats

- Description: Get knowledge base statistics
- Authentication: Required
- Response:

```
{
  "total_documents": 1500,
  "document_types": {
    ".eml": 800,
    ".docx": 200,
    ".pdf": 300
  },
  "audience_breakdown": {
    "customers": 600,
    "sales_reps": 400,
    "internal": 500
  }
}
```

GET /api/files

- Description: List uploaded files
- Authentication: Required
- Query Parameters:
- filter: File type filter (optional)
- audience: Audience filter (optional)
- Response:

```
{
  "files": [
    {
      "name": "file1.eml",
      "size": 1024,
      "modified": "2024-12-01T10:00:00Z",
      "audience": ["customers"]
    }
  ]
}
```

DELETE /api/files/clear-all

- Description: Delete all uploaded files
- Authentication: Required (Admin)
- Response:

```
{
  "success": true,
  "message": "All files deleted"
}
```

GET /api/examples

- Description: Get example questions for placeholder
- Authentication: Required
- Query Parameters: role: User role (optional)
- Response:

```
{
  "examples": [
    "How do I reset my password?",
    "What are the fees?",
    "How do I create a campaign?"
  ]
}
```

POST /api/gitlab/ingest

- Description: Ingest GitLab repository
- Authentication: Required (Admin)
- Body:

```
{
  "repo_url": "https://gitlab.com/username/repo",
  "access_token": "token",
  "include_commits": true,
  "include_readmes": true,
  "include_release_notes": true
}
```

- Response:

```
{
  "success": true,
  "message": "GitLab ingestion started"
}
```

GET /api/users

- Description: List all users
- Authentication: Required (Admin)
- Response:

```
{
```

```
"users": [  
  {  
    "pin": "1234",  
    "name": "John Doe",  
    "role": "Admin"  
  }  
]
```

POST /api/users

- Description: Create new user
- Authentication: Required (Admin)
- Body:

```
{  
  "pin": "1234",  
  "name": "John Doe",  
  "role": "Internal"  
}
```

- Response:

```
{  
  "success": true,  
  "message": "User created"  
}
```

DELETE /api/users/

- Description: Delete user
- Authentication: Required (Admin)
- Response:

```
{  
  "success": true,  
  "message": "User deleted"  
}
```

Twilio Webhook Endpoints

POST /sms

- Description: Receive SMS from Twilio
- Authentication: Not required (Twilio signature validation)
- Body: Twilio webhook format
- Response: TwiML XML response

GET /api/twilio/status

- Description: Get Twilio integration status

- Authentication: Required (Admin)

- Response:

```
{
  "configured": true,
  "phone_number": "+13108736329",
  "webhook_url": "https://askgf.up.railway.app/sms"
}
```

GET /api/twilio/conversations

- Description: List all SMS conversations

- Authentication: Required (Admin)

- Response:

```
{
  "conversations": [
    {
      "phone_number": "+1234567890",
      "message_count": 5,
      "last_message": "2024-12-01T10:00:00Z"
    }
  ]
}
```

GET /api/twilio/conversations/

- Description: Get conversation history for a phone number

- Authentication: Required (Admin)

- Response:

```
{
  "phone_number": "+1234567890",
  "messages": [
    {
      "role": "user",
      "content": "Hello",
      "timestamp": "2024-12-01T10:00:00Z"
    }
  ]
}
```

DELETE /api/twilio/conversations/

- Description: Clear conversation history for a phone number

- Authentication: Required (Admin)

- Response:

```
{
  "success": true,
  "message": "Conversation cleared"
}
```

Configuration

Environment Variables

Create a .env file in the project root with the following variables:

Required Variables

```
# OpenAI Configuration
OPENAI_API_KEY=sk-proj-...

# Pinecone Configuration
PINECONE_API_KEY=your-pinecone-api-key

# Twilio Configuration
TWILIO_ACCOUNT_SID=AC...
TWILIO_AUTH_TOKEN=your-auth-token
TWILIO_PHONE_NUMBER=+13108736329

# Flask Configuration
FLASK_SECRET_KEY=your-secret-key-here
FLASK_DEBUG=false # Set to true for development
```

Optional Variables

```
# Data Directory (for persistent storage)
DATA_DIR=/data

# PostgreSQL Database (for user persistence)
DATABASE_URL=postgresql://user:pass@host:port/dbname

# GitLab Integration (optional)
GITLAB_ACCESS_TOKEN=your-gitlab-token

# Google Docs Integration (optional)
GOOGLE_DOCS_CREDENTIALS_PATH=path/to/credentials.json
```

Pinecone Index Configuration

- Create index at <https://app.pinecone.io>:
- Name: customer-service-kb (or customize in code)
- Dimensions: 1536 (for OpenAI text-embedding-3-small)
- Metric: cosine
- Pod Type: serverless (recommended for cost efficiency)

- Get API key from Pinecone dashboard

Twilio Configuration

- Get credentials from Twilio Console:
 - Account SID
 - Auth Token
 - Phone Number
- Configure webhook:
 - For production: <https://your-domain.railway.app/sms>
 - For local testing: Use ngrok (see Deployment)
- A2P 10DLC Registration (required for US SMS):
 - Register brand at <https://console.twilio.com/us1/develop/sms/a2p-messaging>
 - Create campaign with "Customer Service" use case
 - Wait for approval (1-3 business days)

PostgreSQL Configuration (Production)

- Add PostgreSQL to Railway project
- Get DATABASE_URL from Railway PostgreSQL service
- Add DATABASE_URL to Railway web service variables
- Users will persist across deployments

Deployment

Railway Deployment

- Create Railway account at <https://railway.app>
- Create new project:
 - Click "New Project"
 - Select "Deploy from GitHub repo"
 - Connect your GitHub repository
- Configure environment variables:
 - Go to Variables tab
 - Add all required variables (see Configuration)

- Add PostgreSQL (for user persistence):
- Click "New" → "Database" → "Add PostgreSQL"
- Railway will automatically add DATABASE_URL to your service
- If not automatic, manually add DATABASE_URL from PostgreSQL service
- Deploy:
- Railway will automatically deploy on git push
- Or click "Deploy" button
- Get public URL:
- Railway provides a public URL (e.g., <https://askgf.up.railway.app>)
- Update Twilio webhook to this URL + /sms

Custom Domain Setup

- In Railway dashboard:
- Go to Settings → Domains
- Click "Generate Domain" or "Add Custom Domain"
- Update domain:
- Change from default to desired subdomain (e.g., askgf)
- Result: askgf.up.railway.app
- For custom domain:
- Add CNAME record pointing to Railway
- Railway will provision SSL certificate automatically

Local Testing with ngrok

- Install ngrok:


```
brew install ngrok # macOS
# or download from https://ngrok.com
```
- Start ngrok:


```
ngrok http 5001
```
- Get HTTPS URL from ngrok output (e.g., <https://abc123.ngrok.io>)
- Update Twilio webhook:
- URL: <https://abc123.ngrok.io/sms>
- Method: POST
- Test SMS by sending to your Twilio number

Developer Guide

Project Structure

```
twilio-chatbot/
├── web_app.py           # Main Flask application
├── chatbot.py           # RAG chatbot agent
├── data_processor.py    # Data ingestion and processing
├── requirements.txt     # Python dependencies
├── .env                # Environment variables (gitignored)
├── Procfile            # Railway deployment config
├── runtime.txt         # Python version
├── templates/
│   ├── index.html      # Main UI template
│   ├── login.html      # Login page
│   └── static/
│       ├── css/
│       │   └── style.css # Styles
│       └── js/
│           └── app.js    # Frontend JavaScript
├── scripts/
│   ├── gitlab_connector.py # GitLab integration
│   └── google_docs_connector.py # Google Docs integration
└── uploads/            # Uploaded files (gitignored)
```

Adding New File Types

- Add processing method to `data_processor.py`:

```
def process_new_format(self, file_path: str, audience: str = None):
    # Parse file
    # Extract text
    # Chunk text
    # Add to knowledge base
    pass
```

- Update `ALLOWED_EXTENSIONS` in `web_app.py`:

```
ALLOWED_EXTENSIONS = {'eml', 'docx', 'pdf', 'csv', 'txt', 'newformat'}
```

- Add routing in `process_files_background`:

```
elif ext == 'newformat':
    processor.process_new_format(file_path, audience)
```

- Update UI in `templates/index.html`:

- Add file type to upload accept attribute
- Add filter button for new type

Adding New Integrations

- Create connector script in scripts/:

```
class NewIntegrationConnector:
    def fetch_data(self, config):
        # Fetch data from external source
        pass
```

- Add API endpoint in web_app.py:

```
@app.route('/api/new-integration/ingest', methods=['POST'])
@admin_required
def ingest_new_integration():
    # Process integration
    pass
```

- Add UI section in templates/index.html
- Add JavaScript handlers in static/js/app.js

Modifying Chatbot Behavior

In chatbot.py:

- Change embedding model: Update embedding_model in _get_embedding
- Modify search parameters: Adjust top_k in _retrieve_relevant_context
- Customize prompt: Edit _generate_response system prompt
- Change LLM model: Update model parameter in OpenAI API call

Testing

- Unit tests (to be added):

```
pytest tests/
```

- Manual testing:
 - Upload test files
 - Query chatbot
 - Verify responses and sources

- Integration testing:

- Test Twilio webhook locally with ngrok
- Test GitLab integration
- Test user authentication

Code Style

- Follow PEP 8 Python style guide
- Use type hints where possible
- Add docstrings to functions and classes
- Keep functions focused and small

Troubleshooting

Common Issues

"OPENAI_API_KEY environment variable not set"

- Solution: Add OPENAI_API_KEY to .env file or Railway variables
- Verify: Check environment variables are loaded: `print(os.getenv('OPENAI_API_KEY'))`

"PINECONE_API_KEY environment variable not set"

- Solution: Add PINECONE_API_KEY to .env file or Railway variables
- Verify: Check Pinecone index exists and is accessible

"Database cleared after deployment"

- Solution: Ensure DATABASE_URL is set in Railway variables
- Verify: Check PostgreSQL connection in /api/db-status endpoint
- Note: Pinecone data persists automatically (serverless)

"Users disappeared after deployment"

- Solution: Add DATABASE_URL to Railway web service variables
- Verify: Check /api/db-status endpoint shows PostgreSQL connected
- Fallback: System uses JSON file if PostgreSQL unavailable

"Twilio webhook not receiving messages"

- Solution:
- Verify webhook URL is correct and accessible
- Ensure HTTPS is enabled (required for Twilio)
- Check Twilio logs for errors
- For local testing, use ngrok

"A2P 10DLC Error 30034"

- Solution:
- Register for A2P 10DLC at Twilio Console
- Use "Customer Service" use case for fastest approval
- Wait for campaign approval (1-3 business days)

"Files not processing"

- Solution:
- Check file format is supported
- Verify file size is under 500MB
- Check Processing Status section for errors
- Review server logs for detailed error messages

"CSV files not processing"

- Solution:
- Ensure CSV has headers
- Check CSV format (standard columns: from, to, body, date)
- System will attempt to parse generic CSV formats

Debugging

- Enable debug mode:


```
export FLASK_DEBUG=true
python web_app.py
```
- Check logs:
 - Railway: View logs in Railway dashboard
 - Local: Check terminal output
- Test API endpoints:


```
curl http://localhost:5001/api/stats
```
- Check database status:
 - Visit /api/db-status endpoint
 - Verify PostgreSQL connection

Security

Best Practices

- Never commit .env file:
- .env is in .gitignore
- Use environment variables in production

- Protect API keys:
- Rotate keys regularly
- Use different keys for development/production
- Never expose keys in client-side code

- Session security:
- SECRET_KEY should be random and secure
- Sessions expire after 24 hours
- Use HTTPS in production

- Authentication:
- PINs are hashed before storage
- Admin endpoints require Admin role
- User data filtered by role

- File upload security:
- File size limit: 500MB
- File type validation
- Files stored securely

- Twilio webhook security:
- Validate Twilio signatures (recommended)
- Use HTTPS for webhooks
- Verify phone numbers

Environment Variables Security

- Development: Use .env file (gitignored)
- Production: Use Railway Variables (encrypted)
- Never: Hardcode secrets in code

User Data Protection

- User PINs are hashed using SHA-256
- Conversation history stored securely
- Role-based access control enforced
- Customer data filtered by audience labels

Support

For issues, questions, or contributions:

- Email: hello@groupfund.us
- Phone: 888-390-7620
- Website: <https://www.groupfund.us>
- GitHub: <https://github.com/evonow/twilio-chatbot>

License

[Add your license here]

Changelog

Version 1.0 (December 2024)

- Initial release
- RAG chatbot with Pinecone
- Twilio SMS integration
- Web UI with role-based access
- GitLab integration
- Multi-format file support
- Conversation history
- Source attribution
- FAQ analysis

Documentation maintained by: GroupFund Development Team

Last reviewed: December 2024