RAG Chatbot - Multi-tenant RAG System

A production-ready, multi-tenant Retrieval-Augmented Generation (RAG) chatbot system with advanced features including authentication, multiple LLM providers, and a customizable widget interface.

Features

- 🏢 **Multi-tenant Architecture**: Support for multiple organizations and agents
- 🔐 **Authentication & Authorization**: JWT-based authentication with role-based access
- 🤖 **Multiple LLM Providers**: OpenAI, Anthropic, and Google Vertex AI support
- 📊 **Analytics Dashboard**: Comprehensive usage analytics and reporting
- 🎨 **Customizable Widget**: Embeddable chat widget with dark mode, voice input, and file attachments
- 📁 **Multiple Data Sources**: Ingest from websites, Google Drive, and local files
- 🖥️ **Admin Interface**: Web-based administration panel
- 🚀 **CLI Tools**: Rich terminal interface for management

Project Structure

```
rag_chatbot/
├── __init__.py             # Package initialization
├── main.py                 # FastAPI app entry point
├── auth.py                 # Authentication and user management
├── models.py               # Pydantic models and schemas
├── config.py               # Configuration management
├── database.py             # Database operations
├── vectorstore.py          # Vector store and RAG functionality
├── llm.py                  # LLM provider integrations
├── ingestion.py            # Document ingestion and processing
├── analytics.py            # Analytics and reporting
├── widget.py               # Widget generation
├── cli.py                  # CLI interface
├── routers/                # API route handlers
│   ├── auth_routes.py      # Authentication endpoints
│   ├── chat_routes.py      # Chat and RAG endpoints
│   ├── config_routes.py    # Configuration endpoints
│   ├── admin_routes.py     # Admin interface endpoints
│   ├── analytics_routes.py # Analytics endpoints
│   └── ingest_routes.py    # Ingestion endpoints
├── utils/                  # Utility modules
│   ├── google_drive.py     # Google Drive utilities
│   ├── web_scraper.py      # Web scraping utilities
│   └── file_processors.py  # File processing utilities
└── static/                 # Static files
    └── admin.html          # Admin interface
```

Installation

1. Clone the repository:

```bash
git clone <repository-url>
cd rag_chatbot
```

2. Create a virtual environment:

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

3. Install dependencies:

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

4. Set up environment variables:

```bash
export OPENAI_API_KEY="your-openai-api-key"
export JWT_SECRET_KEY="your-secret-key"
export GOOGLE_APPLICATION_CREDENTIALS="path/to/google-credentials.json"  # Optional
```

Quick Start

1. Start the Server

```bash
python -m rag_chatbot.cli serve
```

Or with auto-reload for development:

```bash
python -m rag_chatbot.cli serve --reload
```

2. Access the Interfaces

- **API Documentation**: http://localhost:8000/docs
- **Admin Interface**: http://localhost:8000/admin.html
- **Widget Embed**: http://localhost:8000/widget.js

3. Default Login

- **Username**: admin
- **Password**: admin

⚠️ **Important**: Change the default password immediately after first login!

CLI Commands

Interactive Dashboard

```bash
python -m rag_chatbot.cli dashboard
```

Create User

```bash
python -m rag_chatbot.cli create-user <username> <password> --tenant <tenant> --role <role>
```

Ingest Content

```bash
# From sitemap
python -m rag_chatbot.cli ingest <tenant> <agent> --sitemap <url>

# From Google Drive
python -m rag_chatbot.cli ingest <tenant> <agent> --drive <folder-id>

# From local files
python -m rag_chatbot.cli ingest <tenant> <agent> --file <path1> --file <path2>
```

Widget Integration

To embed the chat widget on your website, add this script tag:

```html
<script src="http://your-server.com/widget.js?tenant=your-tenant&agent=your-agent"></script>
```

API Usage

Authentication

```python
import requests

# Login
response = requests.post("http://localhost:8000/token",
    data={"username": "admin", "password": "admin"})
token = response.json()["access_token"]

# Use token in headers
headers = {"Authorization": f"Bearer {token}"}
```

Chat API

```python
# Send chat message
chat_data = {
    "messages": [
        {"role": "user", "content": "What is your return policy?"}
    ]
}

response = requests.post(
    "http://localhost:8000/chat?tenant=public&agent=default",
    json=chat_data,
    headers=headers
)

print(response.json())
# {"reply": "Our return policy...", "sources": [...]}
```

Configuration

Each tenant/agent combination has its own configuration file stored in `configs/<tenant>/<agent>.json`:

```json
{
    "bot_name": "Support Bot",
    "system_prompt": "You are a helpful customer support assistant.",
    "primary_color": "#1E88E5",
    "secondary_color": "#FFFFFF",
    "avatar_url": "https://example.com/avatar.png",
    "mode": "inline",
    "auto_open": false,
    "llm_provider": "openai",
    "llm_model": "gpt-4o-mini",
    "temperature": 0.3,
    "allowed_domains": ["*"],
    "enable_voice": true,
    "enable_files": true,
    "enable_tts": false,
    "enable_dark_mode": true,
    "widget_position": "bottom-right",
    "widget_size": "medium",
    "welcome_message": "Hello! How can I help you today?",
    "placeholder_text": "Type your message..."
}
```

Security Considerations

1. **Change Default Credentials**: Always change the default admin password
2. **Use HTTPS**: Deploy with HTTPS in production
3. **Set JWT Secret**: Use a strong, random JWT secret key
4. **Domain Restrictions**: Configure `allowed_domains` to restrict widget embedding
5. **API Keys**: Keep your LLM API keys secure and never commit them

Development

Running Tests

```bash
pytest tests/
```

Code Structure Guidelines

- **Routers**: All API endpoints are organized in the `routers/` directory
- **Models**: Pydantic models define the data structures
- **Utils**: Reusable utilities are in the `utils/` directory
- **Database**: SQLite for simplicity, but easily replaceable with PostgreSQL

Adding a New LLM Provider

1. Add the provider function in `llm.py`:

```python
def _get_newprovider_response(messages, model, temperature):
    # Implementation
    return {"content": "response", "tokens_out": 100}
```

2. Update the `get_llm_response` function to include your provider

Adding New Document Types

Add a processor function in `utils/file_processors.py`:

```python
def _process_newtype(file_path: Path) -> str:
    # Extract text from the file
    return extracted_text
```

Troubleshooting

Common Issues

1. **Vector store missing error**
   - Run ingestion for the tenant/agent first
   - Check if the `vector_store/<tenant>/<agent>` directory exists
2. **Authentication errors**
   - Ensure JWT token is included in headers
   - Check if user has access to the requested tenant
3. **LLM provider errors**
   - Verify API keys are set correctly
   - Check provider-specific requirements

4. **File upload errors**
   - Ensure file types are supported
   - Check file size limits

Production Deployment

Using Docker

Create a `Dockerfile`:

```dockerfile
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["python", "-m", "rag_chatbot.cli", "serve", "--host", "0.0.0.0"]
```

Build and run:

```bash
docker build -t rag-chatbot .
docker run -p 8000:8000 -e OPENAI_API_KEY=your-key rag-chatbot
```

Environment Variables

- `OPENAI_API_KEY`: OpenAI API key
- `ANTHROPIC_API_KEY`: Anthropic API key (optional)
- `GOOGLE_APPLICATION_CREDENTIALS`: Path to Google credentials (optional)
- `JWT_SECRET_KEY`: Secret key for JWT tokens

Database Migration

For production, consider migrating from SQLite to PostgreSQL:

1. Update `database.py` to use SQLAlchemy
2. Set `DATABASE_URL` environment variable
3. Run database migrations

Contributing

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Add tests if applicable
5. Submit a pull request

License

[Your License Here]

Support

For issues and questions:

- GitHub Issues: [repository-url]/issues
- Documentation: [documentation-url]