

# PDF RAG API - Frontend Integration Documentation



## Table of Contents

- [System Overview](#)
  - [API Base Configuration](#)
  - [Authentication Flow](#)
  - [Available Endpoints](#)
  - [Django Integration Guide](#)
  - [Error Handling](#)
  - [Best Practices](#)
  - [Testing Guide](#)
- 



## System Overview

### What This Backend Provides

Your FastAPI backend is a **PDF-based Question-Answering System** that allows users to:






- Upload PDF documents (Admin only)
- Ask questions about the content in those PDFs
- Get AI-powered answers from multiple LLM providers
- Manage user authentication and authorization

### Core Technologies

- **FastAPI**: REST API framework
- **LangChain**: RAG orchestration
- **FAISS**: Vector similarity search

- **Multiple LLMs:** OpenAI, Groq, Google Gemini
- **Reranking:** Cross-encoder for improved accuracy

## Key Features

-  **Role-based access** (Admin/User)
  -  **PDF document processing**
  -  **Multi-LLM support** with hot-swapping
  -  **Intelligent reranking** for better answers
  -  **Bilingual support** (English/Bangla)
- 

## API Base Configuration

### Backend Server Details

Base URL: `http://127.0.0.1:8000`

Default Port: 8000

Protocol: HTTP (HTTPS in production)

Content-Type: `application/json`

### Django Settings Configuration

```
# settings.py
```

```
BACKEND_API_URL = "http://127.0.0.1:8000"
```

```
BACKEND_TIMEOUT = 30 # seconds
```

---

## Authentication Flow

### How Authentication Works

1. **Registration** → 2. **Login** → 3. **Get Token** → 4. **Use Token in Headers**

## Token Usage Pattern

All protected endpoints require:

Authorization: Bearer <token>

## User Roles

- **Admin:** Can upload PDFs, configure LLMs, full access
  - **User:** Can ask questions, view documents
  - **Public:** Can only access login/register
- 



## Available Endpoints

### 1. Authentication Endpoints

#### POST /auth/register

Register new user account

Request:

```
{  
  "username": "john_doe",  
  "password": "secure123",  
  "email": "john@example.com", // optional  
  "full_name": "John Doe"     // optional  
}
```

Response:

```
{  
  "message": "Registration successful!",  
  "username": "john_doe",  
  "role": "user",  
  "status": "success"
```

```
}
```

## **POST /auth/login**

User login

Request:

```
{  
  "username": "john_doe",  
  "password": "secure123"  
}
```

Response:

```
{  
  "access_token": "john_doe_token_2024",  
  "user_role": "user",  
  "message": "User login successful"  
}
```

## **GET /auth/me**

Get current user info (Requires token)

Headers: Authorization: Bearer <token>

Response:

```
{  
  "role": "user",  
  "authenticated": true,  
  "token_valid": true  
}
```

## **GET /auth/check-username/{username}**

Check username availability

Response:

```
{
  "available": true,
  "message": "Username is available"
}
```

---

## 2. Core Functionality Endpoints

### POST /ask ★ Main Endpoint

Ask questions about PDF content

Request:

```
{
  "question": "What is the company policy on remote work?",
  "llm_provider": "openai",    // optional: "openai", "groq", "gemini"
  "model_name": "gpt-4o-mini", // optional
  "use_reranker": true,        // optional: improve accuracy
  "max_chunks": 10             // optional: context size
}
```

Response:

```
{
  "answer": "According to the documents, the company policy...",
  "status": "success",
  "llm_used": "OpenAI (gpt-4o-mini)",
  "chunks_used": 10,
  "reranker_used": true
}
```

---

### 3. Document Management Endpoints

#### GET /documents

List all PDFs in system (Requires token)

Response:

```
{
  "documents": [
    {
      "filename": "company_policy.pdf",
      "size_mb": 2.4,
      "uploaded_date": "2024-01-15T10:30:00"
    }
  ],
  "total": 5
}
```

#### POST /upload-pdf

Upload new PDF (Admin only)

Headers: Authorization: Bearer <admin\_token>

Content-Type: multipart/form-data

Body: file=<pdf\_file>

Response:

```
{
  "status": "success",
  "filename": "20240115_103000_document.pdf",
  "total_pdfs": 6,
  "total_chunks": 1250
}
```

---

## 4. System Status Endpoints

### GET /

Health check

Response:

```
{  
  "message": "PDF RAG API is running",  
  "status": "healthy",  
  "timestamp": "2024-01-15T10:30:00"  
}
```

### GET /status

System readiness

Response:

```
{  
  "status": "ready",  
  "current_llm": {"provider": "openai", "model": "gpt-4o-mini"},  
  "reranker_available": true,  
  "vector_store_ready": true  
}
```

### GET /llm-options

Available LLM configurations

Response:

```
{  
  "providers": {  
    "openai": {  
      "models": ["gpt-4o", "gpt-4o-mini", "gpt-4-turbo"],  
      "default": "gpt-4o-mini"  
    },  
  },  
}
```

```
"groq": {  
  "models": ["deepseek-r1-distill-llama-70b"],  
  "default": "deepseek-r1-distill-llama-70b"  
}  
}  
}
```

---

## Django Integration Guide

### 1. Project Structure

```
django_frontend/  
├── services/  
│   ├── rag_api.py    # API client service  
│   ├── views.py      # Django views  
│   └── templates/  
│       ├── login.html  
│       ├── chat.html  
│       └── documents.html  
└── static/  
    ├── js/  
    └── chat.js      # Frontend JavaScript
```

### 2. Create API Service Class

**services/rag\_api.py**

```
import requests  
from django.conf import settings  
  
class RAGAPIClient:  
    def __init__(self):  
        self.base_url = settings.BACKEND_API_URL
```



```
self.session = requests.Session()
```

### 3. Django Views Integration

#### Key Integration Points:

##### Login View

- Call `/auth/login` endpoint
- Store token in Django session
- Redirect to dashboard

##### Chat View

- Get token from session
- Call `/ask` endpoint with question
- Display formatted answer

##### Document View

- Call `/documents` with auth header
- Display document list
- Handle upload for admins

### 4. Session Management

# Store token after login

```
request.session['api_token'] = response['access_token']
```

```
request.session['user_role'] = response['user_role']
```

### 5. Frontend JavaScript Integration

#### For real-time chat interface:

```
async function askQuestion(question) {  
  const response = await fetch(`${API_URL}/ask`, {
```

```
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({question: question})
  });
  return await response.json();
}
```

---

## Error Handling

### Common Error Responses

Status Code	Meaning	Action Required
400	Bad Request	Check request format
401	Unauthorized	Token missing/expired
403	Forbidden	Admin access required
409	Conflict	Username already exists
422	Validation Error	Check field requirements
500	Server Error	Contact backend team
503	Service Unavailable	RAG system not initialized

### Error Response Format

```
{  
  "detail": "Specific error message here"  
}
```

---



## Best Practices

### 1. Token Management

- Store tokens securely (Django sessions or encrypted cookies)
- Never expose tokens in URLs
- Implement token refresh mechanism

### 2. API Calls

- Always use try-catch blocks
- Implement retry logic for network failures
- Show loading states during API calls

### 3. Question Handling

- Validate questions before sending
- Implement question history
- Cache frequent questions locally

### 4. File Uploads

- Validate file type (PDF only)
- Check file size (<50MB)
- Show upload progress

### 5. Performance

- Implement pagination for documents
- Use async/await for API calls

- Debounce search inputs
- 

## Testing Guide

### Quick Test Flow

#### 1. Test Registration:

```
curl -X POST http://127.0.0.1:8000/auth/register \  
-H "Content-Type: application/json" \  
-d '{"username":"test_user","password":"test123"}'
```

#### 2. Test Login:

```
curl -X POST http://127.0.0.1:8000/auth/login \  
-H "Content-Type: application/json" \  
-d '{"username":"test_user","password":"test123"}'
```

#### 3. Test Question (Main Feature):

```
curl -X POST http://127.0.0.1:8000/ask \  
-H "Content-Type: application/json" \  
-d '{"question":"What is this document about?"}'
```

### Default Test Accounts

- **Admin:** username: **admin**, password: **admin123**
  - **User:** username: **user**, password: **user123**
  - **Demo:** username: **demo**, password: **demo123**
-



## API Flow Diagram

User Journey:

Register → Login → Get Token → Ask Questions



View Documents



[Admin] Upload PDFs



---

## Quick Start for Django Developer

### 1. Start the FastAPI backend:

```
cd backend_folder
python main.py
# API runs on http://127.0.0.1:8000
```

### 2. Test the API is running:

```
curl http://127.0.0.1:8000/
```

### 3. Create Django API client service

### 4. Implement login/register views

### 5. Create chat interface for questions

### 6. Add document management views

---



## Important Notes

### What You DON'T Need to Handle

- PDF processing (backend handles it)
- Vector search (backend handles it)
- LLM integration (backend handles it)
- Reranking logic (backend handles it)

### What You NEED to Handle

- User interface and UX
- Token storage and management
- Error display to users
- Form validations (frontend)
- Loading states
- Response formatting/display

### Bilingual Support

The backend automatically detects and responds in Bangla if the question is in Bangla. No special configuration needed.

---



## Common Integration Issues

Issue	Solution
CORS errors	Backend should have CORS middleware configured
Token not working	Check "Bearer " prefix in Authorization header
503 errors	PDFs not loaded, add PDFs to backend's <a href="#">pdfs/</a> folder

Slow responses      Normal for first question (model loading), use loading indicators

Connection refused      Ensure backend is running on port 8000

---

## **Communication Protocol**

When your teammate needs changes:

1. Check if endpoint exists in this documentation
  2. Test with curl/Postman first
  3. If new endpoint needed, request with clear requirements
  4. Always maintain backward compatibility
- 

This documentation should give your Django teammate everything needed to integrate with your backend. The API is RESTful and straightforward - perfect for Django integration using either Django REST framework clients or simple requests library.