**TaskFlow API Documentation**

**Table of Contents**

1. [Introduction](https://claude.ai/chat/8c38c60d-cfa1-4dac-86a4-d46ed2429b3b#1-introduction)
2. [Getting Started](https://claude.ai/chat/8c38c60d-cfa1-4dac-86a4-d46ed2429b3b#2-getting-started)
3. [Authentication](https://claude.ai/chat/8c38c60d-cfa1-4dac-86a4-d46ed2429b3b#3-authentication)
4. [User Management](https://claude.ai/chat/8c38c60d-cfa1-4dac-86a4-d46ed2429b3b#4-user-management)
5. [Task Management](https://claude.ai/chat/8c38c60d-cfa1-4dac-86a4-d46ed2429b3b#5-task-management)
6. [Category Management](https://claude.ai/chat/8c38c60d-cfa1-4dac-86a4-d46ed2429b3b#6-category-management)
7. [Dashboard & Analytics](https://claude.ai/chat/8c38c60d-cfa1-4dac-86a4-d46ed2429b3b#7-dashboard--analytics)
8. [Export Functionality](https://claude.ai/chat/8c38c60d-cfa1-4dac-86a4-d46ed2429b3b#8-export-functionality)
9. [Weather Integration](https://claude.ai/chat/8c38c60d-cfa1-4dac-86a4-d46ed2429b3b#9-weather-integration)
10. [Error Handling](https://claude.ai/chat/8c38c60d-cfa1-4dac-86a4-d46ed2429b3b#10-error-handling)

**1. Introduction**

**Overview**

TaskFlow API is a RESTful service built with Golang, Gin framework, and GORM for database operations. It provides comprehensive task management functionality with Firebase authentication integration.

**API Information**

* **Base URL**: http://localhost:8080/api
* **Version**: 1.0.0
* **Protocol**: HTTP/HTTPS
* **Response Format**: JSON
* **Authentication**: Firebase Auth

**Key Features**

* User authentication via Firebase
* CRUD operations for tasks and categories
* Real-time dashboard statistics
* Export functionality (CSV/JSON)
* Weather data integration
* Push notifications via FCM

**2. Getting Started**

**Prerequisites**

Go 1.19+

PostgreSQL 15+

Firebase Project

OpenWeatherMap API Key

**Installation**

**Step 1: Clone Repository**

git clone https://github.com/your-repo/taskflow-api.git

cd taskflow-api

**Step 2: Install Dependencies**

go mod init taskflow-api

go mod tidy

**Step 3: Environment Setup**

cp .env.example .env

Configure .env file:

# Database Configuration

DB\_HOST=localhost

DB\_PORT=5432

DB\_USER=postgres

DB\_PASSWORD=your\_password

DB\_NAME=taskflowdb

DB\_SSLMODE=disable

# Firebase Configuration

FIREBASE\_PROJECT\_ID=taskflow-96528

FIREBASE\_CREDENTIALS\_PATH=./firebase-credentials.json

# Weather API Configuration

WEATHER\_API\_KEY=c095bf7882565056a21c585f9ec1be7b

WEATHER\_BASE\_URL=https://api.openweathermap.org/data/2.5

# Server Configuration

PORT=8080

GIN\_MODE=release

**Step 4: Database Setup**

-- Create PostgreSQL database

CREATE DATABASE taskflowdb;

-- Create user (optional)

CREATE USER taskflow\_user WITH PASSWORD 'your\_password';

GRANT ALL PRIVILEGES ON DATABASE taskflowdb TO taskflow\_user;

**Step 5: Firebase Configuration**

1. Download firebase-credentials.json from Firebase Console
2. Place file in project root directory
3. Update FIREBASE\_PROJECT\_ID in .env

**Step 6: Run Application**

# Development mode

go run main.go

# Production build

go build -o taskflow-api

./taskflow-api

**Step 7: Verify Installation**

# Health check

curl http://localhost:8080/health

# Expected response:

{

"status": "ok",

"message": "TaskFlow API is running",

"timestamp": "2024-01-20T10:00:00Z"

}

**Database Schema**

The API will automatically create the following tables:

* users - User information
* categories - Task categories
* tasks - User tasks
* user\_categories - User-category relationships
* sync\_status - External data sync tracking

**3. Authentication**

**Overview**

TaskFlow API uses Firebase Authentication for user management. All protected endpoints require a valid Firebase ID token.

**Authentication Flow**

1. User authenticates via Firebase (frontend)
2. Frontend receives Firebase ID token
3. Token sent in API requests
4. API validates token with Firebase
5. User information retrieved/created in database

**Headers**

Authorization: Bearer {firebase\_id\_token}

Content-Type: application/json

**Error Responses**

// Invalid token

{

"success": false,

"error": "Invalid authentication token",

"code": "AUTH\_INVALID\_TOKEN"

}

// Missing token

{

"success": false,

"error": "Authentication token required",

"code": "AUTH\_TOKEN\_MISSING"

}

**4. User Management**

**4.1 Create User**

Creates a new user in the system.

**Endpoint:** POST /api/users

**Request Body:**

{

"name": "John Doe",

"email": "john.doe@example.com",

"firebase\_uid": "firebase\_user\_uid\_string",

"fcm\_token": "firebase\_fcm\_token\_string"

}

**Response:**

// 201 Created

{

"success": true,

"message": "User created successfully",

"data": {

"id": 1,

"name": "John Doe",

"email": "john.doe@example.com",

"firebase\_uid": "firebase\_user\_uid\_string",

"fcm\_token": "firebase\_fcm\_token\_string",

"created\_at": "2024-01-20T10:00:00Z",

"updated\_at": "2024-01-20T10:00:00Z"

}

}

**Error Responses:**

// 400 Bad Request - Invalid data

{

"success": false,

"error": "Invalid request format",

"details": "validation error details"

}

// 409 Conflict - User already exists

{

"success": false,

"error": "User with this email already exists"

}

**4.2 Get User by ID**

Retrieves user information by user ID.

**Endpoint:** GET /api/users/{id}

**Path Parameters:**

* id (integer, required) - User ID

**Response:**

// 200 OK

{

"success": true,

"data": {

"id": 1,

"name": "John Doe",

"email": "john.doe@example.com",

"created\_at": "2024-01-20T10:00:00Z",

"tasks": [

{

"id": 1,

"title": "Complete project",

"status": "in\_progress"

}

],

"categories": [

{

"id": 1,

"name": "Work",

"color": "#3B82F6"

}

]

}

}

**4.3 Get User by Firebase UID**

Retrieves user information by Firebase UID.

**Endpoint:** GET /api/users/firebase/{firebase\_uid}

**Path Parameters:**

* firebase\_uid (string, required) - Firebase User UID

**Response:**

// 200 OK

{

"success": true,

"data": {

"id": 1,

"name": "John Doe",

"email": "john.doe@example.com",

"firebase\_uid": "firebase\_user\_uid\_string"

}

}

**4.4 Update User Profile**

Updates user profile information.

**Endpoint:** PUT /api/users/{id}

**Path Parameters:**

* id (integer, required) - User ID

**Request Body:**

{

"name": "John Smith",

"email": "john.smith@example.com"

}

**Response:**

// 200 OK

{

"success": true,

"message": "Profile updated successfully",

"data": {

"id": 1,

"name": "John Smith",

"email": "john.smith@example.com",

"updated\_at": "2024-01-20T11:00:00Z"

}

}

**4.5 Update FCM Token**

Updates user's Firebase Cloud Messaging token.

**Endpoint:** PUT /api/users/{id}/fcm-token

**Request Body:**

{

"fcm\_token": "new\_firebase\_fcm\_token\_string"

}

**Response:**

// 200 OK

{

"success": true,

"message": "FCM token updated successfully",

"data": {

"id": 1,

"fcm\_token": "new\_firebase\_fcm\_token\_string"

}

}

**5. Task Management**

**5.1 Get User Tasks**

Retrieves tasks for a specific user with optional filtering.

**Endpoint:** GET /api/users/{id}/tasks

**Path Parameters:**

* id (integer, required) - User ID

**Query Parameters:**

* status (string, optional) - Filter by status: todo, in\_progress, done
* category\_id (integer, optional) - Filter by category ID
* priority (string, optional) - Filter by priority: low, medium, high
* limit (integer, optional) - Limit number of results (default: 50)
* offset (integer, optional) - Offset for pagination (default: 0)

**Example Request:**

GET /api/users/1/tasks?status=in\_progress&category\_id=1&limit=10&offset=0

**Response:**

// 200 OK

{

"success": true,

"data": [

{

"id": 1,

"title": "Complete project documentation",

"description": "Write comprehensive API documentation",

"status": "in\_progress",

"priority": "high",

"user\_id": 1,

"category\_id": 1,

"deadline": "2024-01-25T17:00:00Z",

"reminder\_sent\_at": null,

"created\_at": "2024-01-20T10:00:00Z",

"updated\_at": "2024-01-20T15:30:00Z",

"user": {

"id": 1,

"name": "John Doe",

"email": "john.doe@example.com"

},

"category": {

"id": 1,

"name": "Work",

"color": "#3B82F6"

}

}

],

"pagination": {

"total": 15,

"limit": 10,

"offset": 0,

"has\_more": true

}

}

**5.2 Create Task**

Creates a new task for a user.

**Endpoint:** POST /api/tasks

**Request Body:**

{

"title": "Complete project documentation",

"description": "Write comprehensive API documentation for TaskFlow",

"status": "todo",

"priority": "high",

"user\_id": 1,

"category\_id": 1,

"deadline": "2024-01-25T17:00:00Z"

}

**Field Validation:**

* title (string, required, max: 255) - Task title
* description (string, optional, max: 1000) - Task description
* status (string, required) - One of: todo, in\_progress, done
* priority (string, required) - One of: low, medium, high
* user\_id (integer, required) - Valid user ID
* category\_id (integer, required) - Valid category ID
* deadline (datetime, optional) - Task deadline in ISO 8601 format

**Response:**

// 201 Created

{

"success": true,

"message": "Task created successfully",

"data": {

"id": 2,

"title": "Complete project documentation",

"description": "Write comprehensive API documentation for TaskFlow",

"status": "todo",

"priority": "high",

"user\_id": 1,

"category\_id": 1,

"deadline": "2024-01-25T17:00:00Z",

"created\_at": "2024-01-20T16:00:00Z",

"updated\_at": "2024-01-20T16:00:00Z"

}

}

**5.3 Get Task by ID**

Retrieves a specific task by ID.

**Endpoint:** GET /api/tasks/{id}

**Path Parameters:**

* id (integer, required) - Task ID

**Response:**

// 200 OK

{

"success": true,

"data": {

"id": 1,

"title": "Complete project documentation",

"description": "Write comprehensive API documentation",

"status": "in\_progress",

"priority": "high",

"deadline": "2024-01-25T17:00:00Z",

"created\_at": "2024-01-20T10:00:00Z",

"updated\_at": "2024-01-20T15:30:00Z",

"user": {

"id": 1,

"name": "John Doe"

},

"category": {

"id": 1,

"name": "Work",

"color": "#3B82F6"

}

}

}

**5.4 Update Task**

Updates an existing task.

**Endpoint:** PUT /api/tasks/{id}

**Path Parameters:**

* id (integer, required) - Task ID

**Request Body:**

{

"title": "Updated task title",

"description": "Updated description",

"status": "done",

"priority": "medium",

"category\_id": 2,

"deadline": "2024-01-26T17:00:00Z"

}

**Note:** All fields are optional. Only provided fields will be updated.

**Response:**

// 200 OK

{

"success": true,

"message": "Task updated successfully",

"data": {

"id": 1,

"title": "Updated task title",

"description": "Updated description",

"status": "done",

"priority": "medium",

"category\_id": 2,

"deadline": "2024-01-26T17:00:00Z",

"updated\_at": "2024-01-20T17:00:00Z"

}

}

**5.5 Delete Task**

Deletes a task (soft delete).

**Endpoint:** DELETE /api/tasks/{id}

**Path Parameters:**

* id (integer, required) - Task ID

**Response:**

// 200 OK

{

"success": true,

"message": "Task deleted successfully"

}

**Error Response:**

// 404 Not Found

{

"success": false,

"error": "Task not found"

}

**6. Category Management**

**6.1 Get All Categories**

Retrieves all available task categories.

**Endpoint:** GET /api/categories

**Response:**

// 200 OK

{

"success": true,

"data": [

{

"id": 1,

"name": "Work",

"color": "#3B82F6",

"created\_at": "2024-01-15T10:00:00Z",

"updated\_at": "2024-01-15T10:00:00Z"

},

{

"id": 2,

"name": "Personal",

"color": "#10B981",

"created\_at": "2024-01-15T10:00:00Z",

"updated\_at": "2024-01-15T10:00:00Z"

},

{

"id": 3,

"name": "Shopping",

"color": "#F59E0B",

"created\_at": "2024-01-15T10:00:00Z",

"updated\_at": "2024-01-15T10:00:00Z"

}

]

}

**6.2 Get Category by ID**

Retrieves a specific category by ID.

**Endpoint:** GET /api/categories/{id}

**Path Parameters:**

* id (integer, required) - Category ID

**Response:**

// 200 OK

{

"success": true,

"data": {

"id": 1,

"name": "Work",

"color": "#3B82F6",

"task\_count": 15,

"created\_at": "2024-01-15T10:00:00Z"

}

}

**6.3 Create Category**

Creates a new task category.

**Endpoint:** POST /api/categories

**Request Body:**

{

"name": "Health",

"color": "#EF4444"

}

**Response:**

// 201 Created

{

"success": true,

"message": "Category created successfully",

"data": {

"id": 4,

"name": "Health",

"color": "#EF4444",

"created\_at": "2024-01-20T18:00:00Z"

}

}

**6.4 Update Category**

Updates an existing category.

**Endpoint:** PUT /api/categories/{id}

**Request Body:**

{

"name": "Health & Fitness",

"color": "#EC4899"

}

**Response:**

// 200 OK

{

"success": true,

"message": "Category updated successfully",

"data": {

"id": 4,

"name": "Health & Fitness",

"color": "#EC4899",

"updated\_at": "2024-01-20T18:30:00Z"

}

}

**6.5 Delete Category**

Deletes a category.

**Endpoint:** DELETE /api/categories/{id}

**Response:**

// 200 OK

{

"success": true,

"message": "Category deleted successfully"

}

**Note:** Category deletion will fail if there are tasks associated with it.

**7. Dashboard & Analytics**

**7.1 Get Dashboard Statistics**

Retrieves comprehensive dashboard statistics.

**Endpoint:** GET /api/dashboard/stats

**Query Parameters:**

* user\_id (integer, optional) - Filter stats for specific user
* start\_date (date, optional) - Start date for date range filter
* end\_date (date, optional) - End date for date range filter

**Response:**

// 200 OK

{

"success": true,

"data": {

"stats": {

"total\_users": 25,

"total\_tasks": 150,

"completed\_tasks": 95,

"in\_progress\_tasks": 35,

"pending\_tasks": 20,

"completion\_rate": 63.33,

"average\_completion\_time": 2.5

},

"tasks\_by\_status": [

{

"status": "todo",

"task\_count": 20

},

{

"status": "in\_progress",

"task\_count": 35

},

{

"status": "done",

"task\_count": 95

}

],

"tasks\_by\_category": [

{

"category\_name": "Work",

"task\_count": 80

},

{

"category\_name": "Personal",

"task\_count": 45

},

{

"category\_name": "Shopping",

"task\_count": 25

}

],

"tasks\_by\_priority": [

{

"priority": "high",

"task\_count": 30

},

{

"priority": "medium",

"task\_count": 80

},

{

"priority": "low",

"task\_count": 40

}

],

"recent\_activity": [

{

"id": 1,

"title": "Task completed",

"description": "John Doe completed 'Write documentation'",

"timestamp": "2024-01-20T17:30:00Z",

"user": "John Doe"

}

]

}

}

**8. Export Functionality**

**8.1 Export User Tasks (CSV)**

Exports user tasks in CSV format.

**Endpoint:** GET /api/users/{user\_id}/tasks/export/csv

**Path Parameters:**

* user\_id (integer, required) - User ID

**Query Parameters:**

* status (string, optional) - Filter by status
* category\_id (integer, optional) - Filter by category
* start\_date (date, optional) - Start date filter
* end\_date (date, optional) - End date filter

**Response:**

Content-Type: text/csv

Content-Disposition: attachment; filename=my\_tasks\_2024-01-20.csv

ID,Title,Description,Status,Priority,Category,Deadline,Created Date,Updated Date

1,"Complete project","Finish TaskFlow application","in\_progress","high","Work","2024-01-25 17:00:00","2024-01-20 10:00:00","2024-01-20 15:30:00"

2,"Buy groceries","Weekly grocery shopping","todo","medium","Personal","2024-01-21 18:00:00","2024-01-20 16:00:00","2024-01-20 16:00:00"

**8.2 Export User Tasks (JSON)**

Exports user tasks in JSON format.

**Endpoint:** GET /api/users/{user\_id}/tasks/export/json

**Response:**

Content-Type: application/json

Content-Disposition: attachment; filename=my\_tasks\_2024-01-20.json

{

"success": true,

"data": [

{

"id": 1,

"title": "Complete project",

"description": "Finish TaskFlow application",

"status": "in\_progress",

"priority": "high",

"category": {

"id": 1,

"name": "Work",

"color": "#3B82F6"

},

"deadline": "2024-01-25T17:00:00Z",

"created\_at": "2024-01-20T10:00:00Z",

"updated\_at": "2024-01-20T15:30:00Z"

}

],

"exported\_at": "2024-01-20T18:00:00Z",

"total\_tasks": 15,

"user\_id": 1,

"export\_type": "json"

}

**8.3 Admin Export (All Users Data)**

Exports all users data (Admin only).

**Endpoint:** GET /api/admin/export/users

**Response:**

Content-Type: application/json

Content-Disposition: attachment; filename=all\_users\_data\_2024-01-20.json

{

"success": true,

"data": [

{

"id": 1,

"name": "John Doe",

"email": "john@example.com",

"tasks": [...],

"categories": [...]

}

],

"exported\_at": "2024-01-20T18:00:00Z",

"total\_users": 25,

"export\_type": "admin\_export"

}

**9. Weather Integration**

**9.1 Get Weather Data**

Retrieves current weather data for a city.

**Endpoint:** GET /api/weather

**Query Parameters:**

* city (string, optional) - City name (default: Jakarta)

**Response:**

// 200 OK

{

"success": true,

"data": {

"location": "Jakarta",

"country": "ID",

"temperature": 28.5,

"feels\_like": 32.1,

"description": "partly cloudy",

"humidity": 75,

"pressure": 1013,

"wind\_speed": 3.2,

"wind\_deg": 180,

"visibility": 10000,

"clouds": 40,

"icon": "02d",

"timestamp": "2024-01-20T18:00:00Z",

"sunrise": "2024-01-20T06:00:00Z",

"sunset": "2024-01-20T18:30:00Z"

}

}

**9.2 Get Multiple Cities Weather**

Retrieves weather data for multiple cities.

**Endpoint:** GET /api/weather/multiple

**Response:**

// 200 OK

{

"success": true,

"data": {

"Jakarta": {

"temperature": 28.5,

"description": "partly cloudy"

},

"Bandung": {

"temperature": 24.2,

"description": "light rain"

},

"Surabaya": {

"temperature": 30.1,

"description": "clear sky"

}

},

"count": 3

}

**10. Error Handling**

**Standard Error Response Format**

All API errors follow this consistent format:

{

"success": false,

"error": "Human-readable error message",

"details": "Technical error details (optional)",

"code": "ERROR\_CODE (optional)",

"timestamp": "2024-01-20T18:00:00Z"

}

**HTTP Status Codes**

| **Status Code** | **Description** |
| --- | --- |
| 200 | OK - Request successful |
| 201 | Created - Resource created successfully |
| 400 | Bad Request - Invalid request data |
| 401 | Unauthorized - Invalid or missing authentication |
| 403 | Forbidden - Insufficient permissions |
| 404 | Not Found - Resource not found |
| 409 | Conflict - Resource already exists |
| 422 | Unprocessable Entity - Validation errors |
| 429 | Too Many Requests - Rate limit exceeded |
| 500 | Internal Server Error - Server error |

**Common Error Codes**

| **Error Code** | **Description** |
| --- | --- |
| VALIDATION\_ERROR | Request validation failed |
| AUTH\_TOKEN\_MISSING | Authentication token not provided |
| AUTH\_INVALID\_TOKEN | Invalid authentication token |
| RESOURCE\_NOT\_FOUND | Requested resource does not exist |
| PERMISSION\_DENIED | Insufficient permissions |
| RATE\_LIMIT\_EXCEEDED | API rate limit exceeded |
| DATABASE\_ERROR | Database operation failed |
| EXTERNAL\_API\_ERROR | External service unavailable |

**Example Error Responses**

**Validation Error**

{

"success": false,

"error": "Validation failed",

"code": "VALIDATION\_ERROR",

"details": {

"title": "Title is required",

"user\_id": "Invalid user ID"

}

}

**Authentication Error**

{

"success": false,

"error": "Invalid authentication token",

"code": "AUTH\_INVALID\_TOKEN"

}

**Resource Not Found**

{

"success": false,

"error": "Task not found",

"code": "RESOURCE\_NOT\_FOUND"

}