API Documentation - TaxSavvy

Introduction

The **TaxSavvy API** offers a suite of endpoints designed to help users manage their finances and taxes more efficiently. It provides functionality for:

- **User Authentication**: Register and log in users securely.
- **Tax Calculations**: Compute taxes for both the old and new tax regimes based on user input.
- **Budget Insights**: Generate personalised budget reports and provide financial tips to help users optimize their savings.
- **Scheme Finder**: Suggest government schemes based on user profiles and income.
- **AI Chatbot**: Provide an AI-driven chatbot for answering user queries related to taxes.

API ENDPOINTS

● LOGIN/SIGNUP

BASE URL : http://localhost:5000

1. Signup Endpoint

- **Endpoint:** POST /auth/signup
- **Description:** This endpoint allows users to register a new account by providing their name, email, and password. The password is hashed for secure storage.

Request Body:

json

```
{
  "name": "JohnDoe",
  "email": "john@example.com",
  "password": "password123"
}
```

Response:

Success:

json

```
{
 "success": true,
  "message": "User registered successfully"
}
```

Error (User already exists):

json

```
{
  "success": false,
  "message": "User already exists"
}
```

Error (Internal server issue):

json

```
{
  "success": false,
  "message": "Error registering user"
}
```

2. Login Endpoint

- **Endpoint:** POST /auth/login
- **Description:** This endpoint allows users to log in by providing their email and password. The server will check the email and compare the provided password with the stored hashed password.

Request Body:

json

```
{
  "email": "john@example.com",
  "password": "password123"
}
```

Response:
Success:

json
```
{

  "success": true,
  "message": "Login successful"
}
```
Error (User not found):
json
```
{
  "success": false,
  "message": "User not found"
}
```
Error (Incorrect password):
json
```
{
  "success": false,
  "message": "Incorrect password"
}
```
Error (Internal server issue):
json
```
{
  "success": false,
  "message": "Error logging in"
}
```

- ## Tax Calculation

  Databases: MongoDB (Tax Data)

  Service: Taxservice

Details:

- The **Tax Calculator** and **Comparison Tool** in the UI Layer interact with the **Tax Service** to calculate taxes based on user input (income, deductions, etc.).

- The **Tax Service** communicates with the **MONGO Tax DB** to store and retrieve tax-related data.

Request (POST to /tax/save):

- Body: Contains user details like email, age, salary, interestIncome, etc.

- **Example:**

Json
```
{
   "email": "user@example.com",
   "age": 30,
   "salary": 500000,
   "interestIncome": 10000,
   "rentalIncome": 20000,
   "digitalAssetsIncome": 5000;
   "deductions80C": 150000,
   "medicalInsurance80D": 25000,
   "npsEmployer80CCD2": 50000,
   "otherDeductions": 5000
}//just an example data original db has more inputs
```

Response:

- Status: Success or Failure (200 or 400/500)
- Body: Contains calculated oldRegime and newRegime tax data.
- Example:

json

```
{
    "success": true,
    "oldRegime": {
        "totalIncome": 550000,
        "taxableIncome": 350000,
        "taxPayable": 25000,
        "netIncomeAfterTax": 525000,
        "effectiveTaxRate": "4.55%"
    },
    "newRegime": {
        "totalIncome": 550000,
        "taxableIncome": 450000,
        "taxPayable": 35000,
        "netIncomeAfterTax": 515000,
        "effectiveTaxRate": "6.36%"
    }
}
```

Request (GET to /tax/:emai**l):**

- **URL:** GET /tax/{email}
- **Response: Tax data for the user with the given email.**

json

```
{
    "success": true,
    "data": {
        "email": "user@example.com",
        "age": 30,
        "oldRegime": {...},
      "newRegime": {...},
      "createdAt": "2025-03-30T15:00:00Z"
   }
}
```

Request (PUT to /tax/:id**):**

- **Body:** Updated tax data for the given id.

json

```
{
   "age": 31,
   "salary": 550000,
   "interestIncome": 15000,
   "otherIncome": 7000,
   "deductions80C": 160000
}
```

Response: Confirmation of the update.
json

```
{
   "success": true,
   "message": "Tax data updated successfully"
}
```

Endpoint:

- **URL:** http://localhost:5000/tax/{email}
- **Method:** GET
- **Authorization:** No authentication is required.

Request Parameters:

- **Path Parameter:**
  - email (Required): The email ID of the logged-in user. It is used to fetch the tax data specific to that user.
  - **Example:** /tax/john.doe@example.com

Response:

Successful Response (200 OK**):**

- **Content-Type:** application/json

Body:

json

```
{
  "data": {
```

```
  "oldRegime": {
    "totalIncome": 1000000,
    "taxableIncome": 950000,
    "taxPayable": 120000,
    "chapterVIA": 50000,
    "exemptAllowances": 100000,
    "standardDeductions": 50000,
    "incomeTax": 100000,
    "surcharge": 5000,
    "cess": 5000,
    "netIncomeAfterTax": 850000,
    "effectiveTaxRate": "12%"
  },
  "newRegime": {
    "totalIncome": 1000000,
    "taxableIncome": 950000,
    "taxPayable": 100000,
    "chapterVIA": 50000,
    "exemptAllowances": 100000,
    "standardDeductions": 50000,
    "incomeTax": 80000,
    "surcharge": 4000,
    "cess": 4000,
    "netIncomeAfterTax": 900000,
    "effectiveTaxRate": "10%"
  }
}}
```

Error Response (4xx/5xx**):**

- **Status Code:** 404 Not Found or 500 Internal Server Error
- **Content-Type:** application/json

Body:
json

```
{
 "error": "Failed to fetch tax data",
 "message": "The tax data for the user could not be retrieved."

        }
```

Financial Tips API Documentation

Base URL : http://localhost:5000/financial-tips

Endpoints

1. Get Financial Tips by Email

Endpoint: GET /:email

Description: Fetches personalized financial tips for a user based on their tax data.

Request Parameters:

Email String The user's email to fetch financial tips
Response:

- Success Response:

```
{
  "success": true,
  "email": "user@gmail.com",
  "tips": [
    "Increase your investments under Section 80C like PPF, ELSS, and EPF up to ₹1.5 lakh.",
    "Consider increasing your health insurance premium under Section 80D to ₹25,000."
  ],
  "count": 2,
  "lastUpdated": "2025-03-30T12:45:00.000Z"
}
```

- Failure Response (No Data Found):

```
        {

  "success": false,
  "email": "user@example.com",
  "tips": ["No tax data found - please complete your tax profile"],
  "message": "User tax data not found"
}
```

- Error Response (Server Error):

```
{
  "success": false,
  "error": "Server error",
  "tips": ["System temporarily unavailable - please try again later"]
}
```

Authentication & Security

- This API currently does not require authentication but should be secured using token-based authentication in production.

- Ensure .env contains the valid GEMINI_API_KEY for AI-based responses.

- ## Budget Report

1. Overview

The Budget Report API provides insights into financial, social, and national impacts based on user-provided details like profession and age group. This API fetches tax data, generates AI-driven summaries, and returns a structured report.

2. Base URL

https://api.taxsavvy.com

3. Authentication

All requests must include an API key for authentication.

Authorization: Bearer YOUR_API_KEY

API Endpoints

1.Fetch Budget Report

Endpoint:

GET /api/budget-report

Description: Retrieves a budget impact report based on user-provided profession and age group.

Query Parameters:

Age

Profession

Example Request:

GET
/api/budget-report??profession=${profession}&ageGroup=${ageGroup}&email=${email}

Response:

```
{
  "status": "success",
  "data": {
    "financialImpact": {
      "taxSavings": 25000,
      “Tax payable”:30000,
    },
    "socialImpact": "Your tax savings can contribute to infrastructure projects like roads and
schools."
  }
}
```

Response Fields:

financialImpact (Object) : Financial impact details (tax savings, investment opportunities)
socialImpact(String): AI-generated insights on social impact
nationalImpact(String):Generalized national impact summary(same for all inputs)

Error Responses:

400:Unauthorized. Missing or invalid API key.
401:Internal Server Error. Please try again later
500:Invalid parameters. Please provide a profession and age group.

● AI Chatbot API Documentation

Base URL

http://127.0.0.1:5000

Endpoints

1. Chat with AI

Endpoint:

POST /chat

Description:

Sends a message to the chatbot and receives an AI-generated response.

Request Body:

```
{
  "message": "<User's input message>"
}
```

Response:

```
{
  "response": "<AI-generated response>"
}
```

Success Response:

- **Status Code:** 200 OK
- **Example:**

```
{
  "response": "The standard deduction for salaried employees is Rs. 50,000."
}
```

Error Responses:

- **Empty Message:**
    - **Status Code:** 400 Bad Request
    - **Example:**

```
{
  "error": "Empty message"
}
```

- **Server Error:**
    - **Status Code:** 500 Internal Server Error
    - **Example:**

```
{
  "error": "Internal server error message"
}
```

2. Get Chat History

Endpoint:
GET /history

Description:

Fetches the last 10 messages exchanged between the user and the chatbot.

Response:
```
{
  "chat_history": [
    {
      "_id": "<MongoDB ObjectId>",
      "text": "<Message text>",
      "sender": "<user | bot>"
    },
    ...
  ]
}
```

Success Response:

- **Status Code:** 200 OK
- **Example:**

```
{
  "chat_history": [
    {
      "_id": "65a7c92f7d1c4e7c3c2a1a3b",
      "text": "What is the standard deduction for salaried employees?",
      "sender": "user"
    },
    {
      "_id": "65a7c92f7d1c4e7c3c2a1a3c",
      "text": "The standard deduction for salaried employees is Rs. 50,000.",
      "sender": "bot"
    }
  ]
}
```

Error Response:

- **Server Error:**
    - **Status Code:** 500 Internal Server Error
    - **Example:**

    ```
    {
        "error": "Internal server error message"
     }
    ```

- ## Budget Features

Base URL

http://localhost:5000/budget-features

Endpoints

1. Fetch Financial Tips

Endpoint:

GET /api/tips

Description: Fetches financial tips from a local JSON file.

Response:

```
[
  {
    "id": 1,
    "title": "Save Money",
    "description": "Cut unnecessary expenses to save more."
  },
  {
    "id": 2,
    "title": "Invest Wisely",
    "description": "Diversify investments to reduce risks."
  }
]
```

Errors:

- 500 - Failed to load or parse tips.json

2. Fetch Filter Options

Endpoint:

GET /filters/:type

Description: Fetches unique values for different filters (location, age, profession, category).

Path Parameter:

- type (string) - The filter type (location, age, profession, category)

Response Example (for /filters/location):

```
["Bihar", "Madhya Pradesh"]
```

Errors:

- 400 - Invalid filter type
- 500 - Error fetching filters from the database

3. Fetch Features Based on Filters

Endpoint:

GET /features

Description: Fetches budget features based on selected filters.

Query Parameters:

- age (string) - Age group (optional)
- location (string) - Location (optional)
- profession (string) - Profession (optional)
- category (string) - Feature category (optional)

Request Example:

```
GET /features?location=New%20York&profession=Doctor
```

Response Example:

```
[
  {
    "id": 1,
    "name": "Tax Savings Plan",
    "description": "A government-backed tax-saving scheme for professionals.",
    "category": "Tax Benefits",
    "feature_detailed_explanation": "This plan allows you to save up to 20% on taxes by
investing in government-approved funds."
  }
]
```

Errors:

- **500 - Error fetching features from the database**

4. Handle Unknown API Endpoints

Fallback Route:

Any undefined route

Response:

```
{
  "error": "API endpoint not found"
}
```

Errors:

- 404 - The requested API route does not exist

Notes

- Ensure the MySQL database is properly initialized before making requests.
- Query parameters for /features are optional, but filtering will be more effective if at least one is provided.
- If a category contains multiple values, it is parsed and split for filtering.