TaxBot

Digital assistant that helps you do your taxes

What the assignment demands

- Python Web framework along with a frontend
- REST API
- SQLite integration
- OpenAI integration
- Docker
- Github workflow(and package build)

Step 1: Web framework and web template

Flask was chosen rather randomly. As I have no large experience with any python framework.

HTML page has embedded JS and CSS, It handles the data submission, updating the page on response, and loading previous content on reload from the database.
Fine for an assignment but not a real world App

Step 1: Difficulties faced

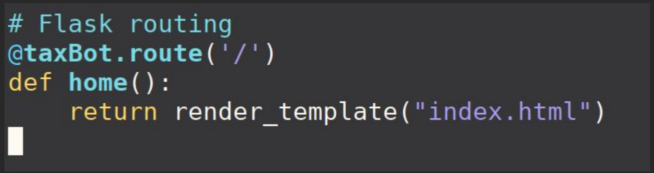
Little experience in this level of web dev coming from a mostly embedded space. I had to learn a lot.

Step 1: Code Review

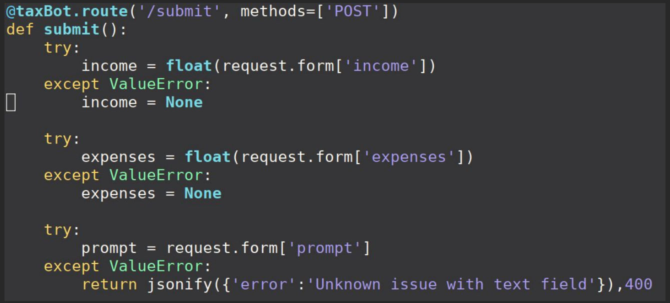
Initial implementation

- home() renders the HTML(or rather the homepage)
- submit() accepts data the user submits from the web page.

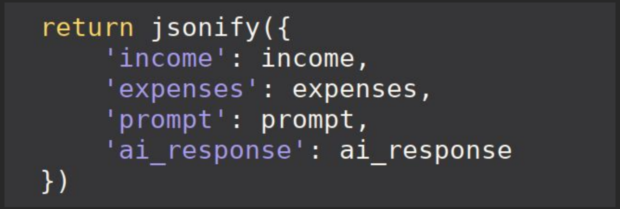
* Data is values are checked due to data integrity demands in Step 2



```python
# Flask routing
@taxBot.route('/')
def home():
    return render_template("index.html")
```



```python
@taxBot.route('/submit', methods=['POST'])
def submit():
    try:
        income = float(request.form['income'])
    except ValueError:
        income = None

    try:
        expenses = float(request.form['expenses'])
    except ValueError:
        expenses = None

    try:
        prompt = request.form['prompt']
    except ValueError:
        return jsonify({'error':'Unknown issue with text field'}),400
```

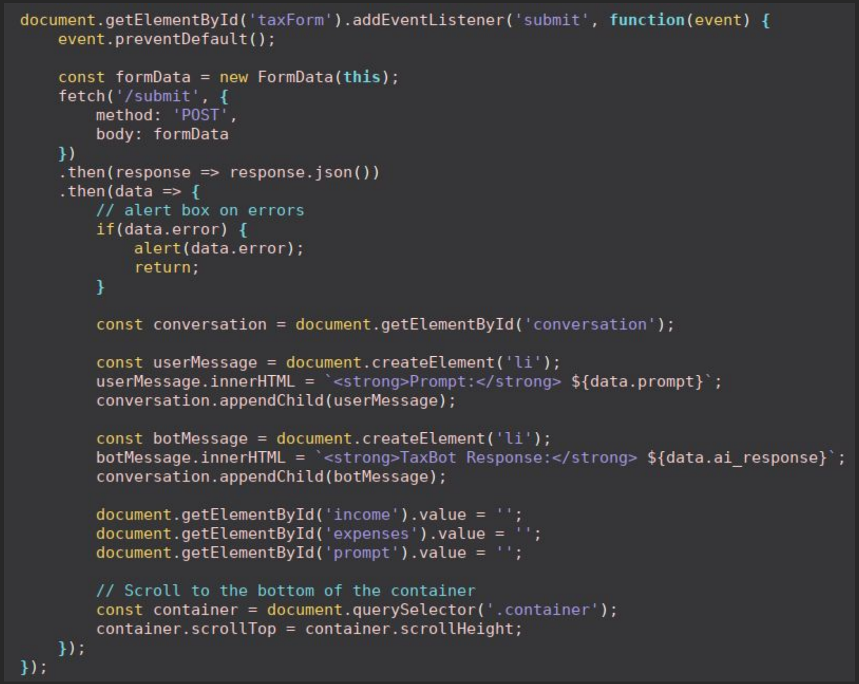


```python
return jsonify({
    'income': income,
    'expenses': expenses,
    'prompt': prompt,
    'ai_response': ai_response
})
```

Step 1:Code Review

Initial implementation

- home() renders the HTML(or rather the homepage)
- submit() accepts data the user submits from the web page.
- Simple JS provides the data and updates the HTML document, clears the text boxes.



```
document.getElementById('taxForm').addEventListener('submit', function(event) {
    event.preventDefault();

    const formData = new FormData(this);
    fetch('/submit', {
        method: 'POST',
        body: formData
    })
    .then(response => response.json())
    .then(data => {
        // alert box on errors
        if(data.error) {
            alert(data.error);
            return;
        }

        const conversation = document.getElementById('conversation');

        const userMessage = document.createElement('li');
        userMessage.innerHTML = `<strong>Prompt:</strong> ${data.prompt}`;
        conversation.appendChild(userMessage);

        const botMessage = document.createElement('li');
        botMessage.innerHTML = `<strong>TaxBot Response:</strong> ${data.ai_response}`;
        conversation.appendChild(botMessage);

        document.getElementById('income').value = '';
        document.getElementById('expenses').value = '';
        document.getElementById('prompt').value = '';

        // Scroll to the bottom of the container
        const container = document.querySelector('.container');
        container.scrollTop = container.scrollHeight;
    });
});
```

Step 2: REST Api and SQLite

- Used Flask SQLAlchemy, seemed easier to handling raw SQLite commands in python. Also well integrated with Flask.
- Implemented more REST functions now that i have a DB.

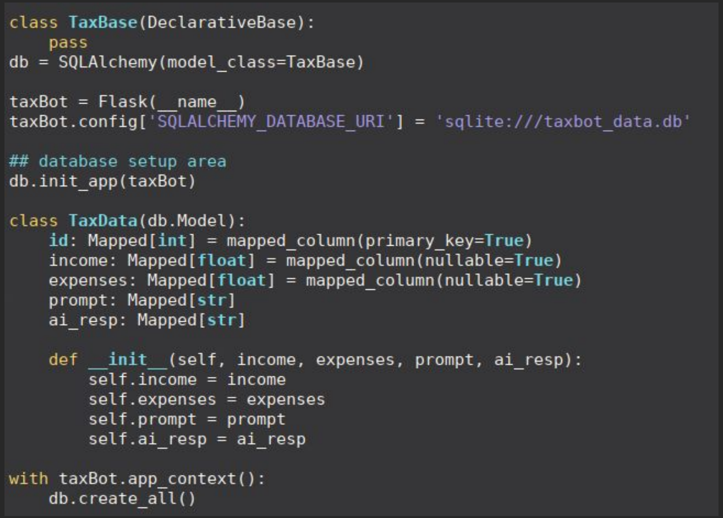
Step 2: Difficulties faced

- Issues appeared when trying to make the application more robust due to the way variable scope and typing works in python. (fixed)
- Data is not secure at all (not fixed)

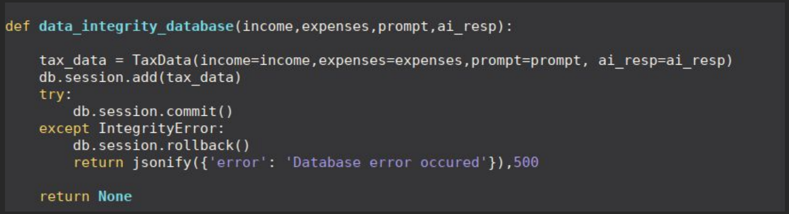
Step 2: Code review

- With SQLAlchemy creating a database was easy and painless.
- Basic data integrity check

* We checked the data on submission on Step 1



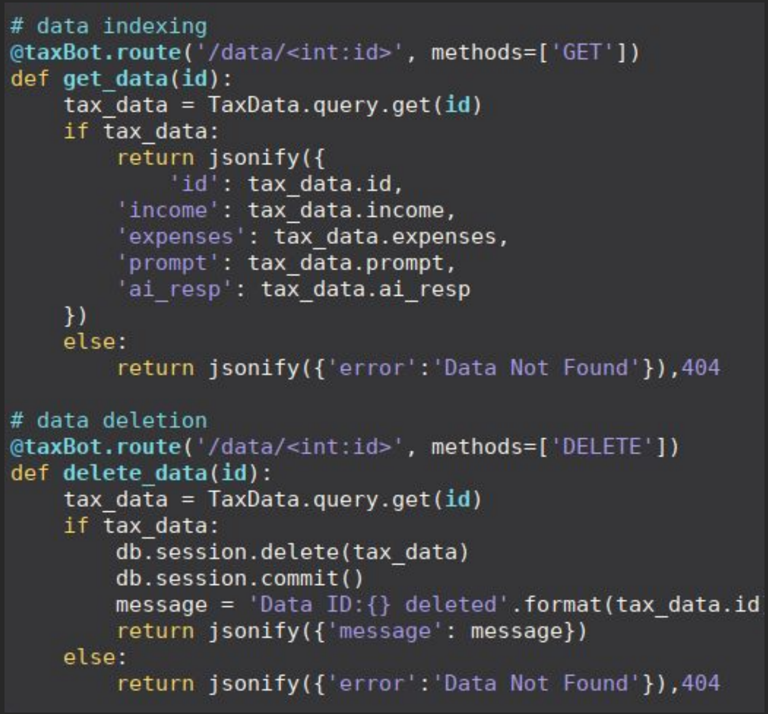
```python
class TaxBase(DeclarativeBase):
    pass
db = SQLAlchemy(model_class=TaxBase)

taxBot = Flask(__name__)
taxBot.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///taxbot_data.db'

## database setup area
db.init_app(taxBot)

class TaxData(db.Model):
    id: Mapped[int] = mapped_column(primary_key=True)
    income: Mapped[float] = mapped_column(nullable=True)
    expenses: Mapped[float] = mapped_column(nullable=True)
    prompt: Mapped[str]
    ai_resp: Mapped[str]

    def __init__(self, income, expenses, prompt, ai_resp):
        self.income = income
        self.expenses = expenses
        self.prompt = prompt
        self.ai_resp = ai_resp

with taxBot.app_context():
    db.create_all()
```



```python
def data_integrity_database(income,expenses,prompt,ai_resp):

    tax_data = TaxData(income=income,expenses=expenses,prompt=prompt, ai_resp=ai_resp)
    db.session.add(tax_data)
    try:
        db.session.commit()
    except IntegrityError:
        db.session.rollback()
        return jsonify({'error': 'Database error occured'}),500

    return None
```

Step 2: Code review

- With SQLAlchemy creating a database was easy and painless.
- Basic data integrity check
- Additional REST functions implemented



```
# data indexing
@taxBot.route('/data/<int:id>', methods=['GET'])
def get_data(id):
    tax_data = TaxData.query.get(id)
    if tax_data:
        return jsonify({
            'id': tax_data.id,
        'income': tax_data.income,
        'expenses': tax_data.expenses,
        'prompt': tax_data.prompt,
        'ai_resp': tax_data.ai_resp
    })
    else:
        return jsonify({'error':'Data Not Found'}),404

# data deletion
@taxBot.route('/data/<int:id>', methods=['DELETE'])
def delete_data(id):
    tax_data = TaxData.query.get(id)
    if tax_data:
        db.session.delete(tax_data)
        db.session.commit()
        message = 'Data ID:{} deleted'.format(tax_data.id
        return jsonify({'message': message})
    else:
        return jsonify({'error':'Data Not Found'}),404
```

Step 2: Showcase

Get the entire DB log – used when refreshing the page to populate it

```
curl -X GET http://localhost:5000/log
```

Get DB entry using the id number

```
curl -X GET http://localhost:5000/data/NUMBER
```

Delete DB entry using the id number

```
curl -X DELETE http://localhost:5000/data/NUMBER
```

Post to the db - Used when pressing the submit button

```
curl -X POST http://localhost:5000/submit -d "income=50000&expenses=10000&prompt=Help me with my taxes"
```

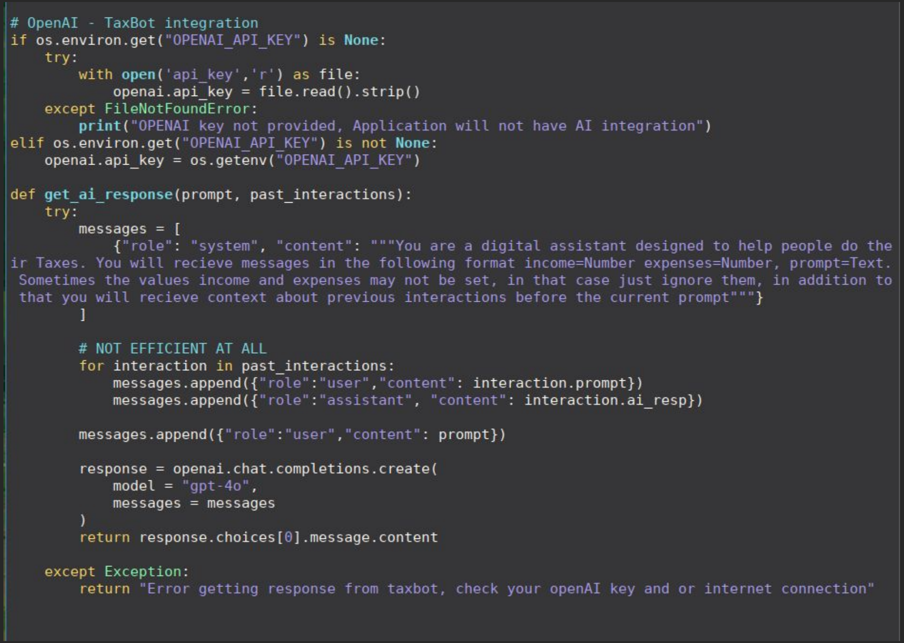
Step 3: OpenAI integration

- Used the openAI package, quite easy to set up.
- Can be used either with the OPEN_API_KEY env variable or a file called api-key which contains the key.

Step 3: Code Review

- Read environment variables.
- OpenAI uses “roles” a way for you to program GPT.

  System: Defines the conversation topic
  User: Represents the user who interacts with bot.
  Assistant: Responses to user inputs

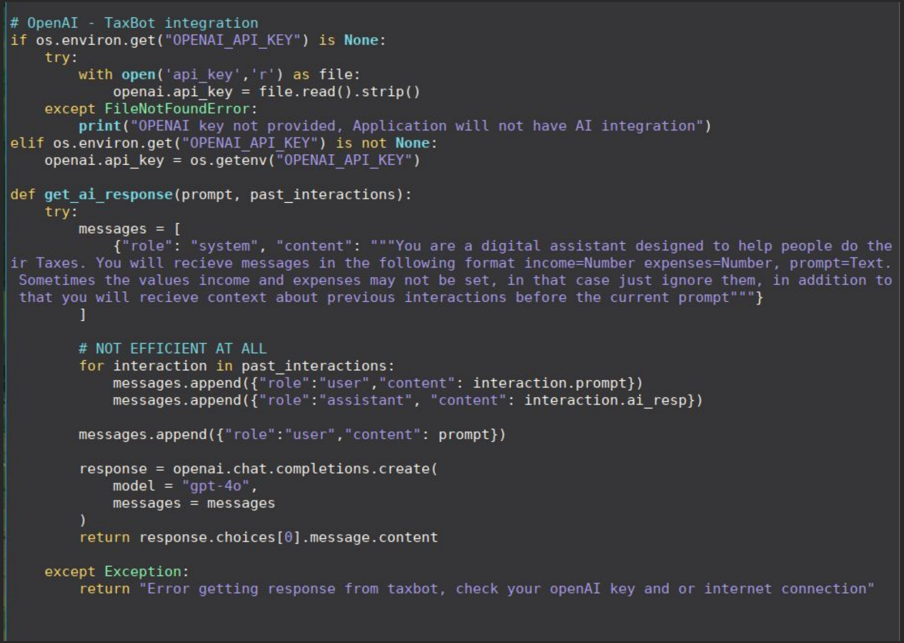


```
# OpenAI - TaxBot integration
if os.environ.get("OPENAI_API_KEY") is None:
    try:
        with open('api_key','r') as file:
            openai.api_key = file.read().strip()
    except FileNotFoundError:
        print("OPENAI key not provided, Application will not have AI integration")
elif os.environ.get("OPENAI_API_KEY") is not None:
    openai.api_key = os.getenv("OPENAI_API_KEY")

def get_ai_response(prompt, past_interactions):
    try:
        messages = [
            {"role": "system", "content": """You are a digital assistant designed to help people do the
ir Taxes. You will recieve messages in the following format income=Number expenses=Number, prompt=Text.
 Sometimes the values income and expenses may not be set, in that case just ignore them, in addition to
 that you will recieve context about previous interactions before the current prompt"""}
        ]

        # NOT EFFICIENT AT ALL
        for interaction in past_interactions:
            messages.append({"role":"user","content": interaction.prompt})
            messages.append({"role":"assistant", "content": interaction.ai_resp})

        messages.append({"role":"user","content": prompt})

        response = openai.chat.completions.create(
            model = "gpt-4o",
            messages = messages
        )
        return response.choices[0].message.content

    except Exception:
        return "Error getting response from taxbot, check your openAI key and or internet connection"
```

Step 3: Code Review

- Implementation is not that efficient since it needs to get all the past interactions and feed them to the bot.
- I assume eventually this will run into an issue due to the maximum amount of tokens is reached.

  Fine for an assignment, not fine a real world app.



```python
# OpenAI - TaxBot integration
if os.environ.get("OPENAI_API_KEY") is None:
    try:
        with open('api_key','r') as file:
            openai.api_key = file.read().strip()
    except FileNotFoundError:
        print("OPENAI key not provided, Application will not have AI integration")
elif os.environ.get("OPENAI_API_KEY") is not None:
    openai.api_key = os.getenv("OPENAI_API_KEY")

def get_ai_response(prompt, past_interactions):
    try:
        messages = [
            {"role": "system", "content": """You are a digital assistant designed to help people do the
ir Taxes. You will recieve messages in the following format income=Number expenses=Number, prompt=Text.
 Sometimes the values income and expenses may not be set, in that case just ignore them, in addition to
 that you will recieve context about previous interactions before the current prompt"""}
        ]

        # NOT EFFICIENT AT ALL
        for interaction in past_interactions:
            messages.append({"role":"user","content": interaction.prompt})
            messages.append({"role":"assistant", "content": interaction.ai_resp})

        messages.append({"role":"user","content": prompt})

        response = openai.chat.completions.create(
            model = "gpt-4o",
            messages = messages
        )
        return response.choices[0].message.content

    except Exception:
        return "Error getting response from taxbot, check your openAI key and or internet connection"
```

Step 4 & 5: Docker and Github workflows

- Wrote a Dockerfile that can be used to build the program.
- Dockerfile DOES NOT come with an api_key file the user needs to either provide his own or use the environment variable OPENAI_API_KEY

```
docker run -p 5000:5000 -e OPENAI_API_KEY=XXXXXXXXX taxbot
```

- Github workflows build and publish the application package on every push

```
docker pull ghcr.io/svin24/taxbot:latest
```

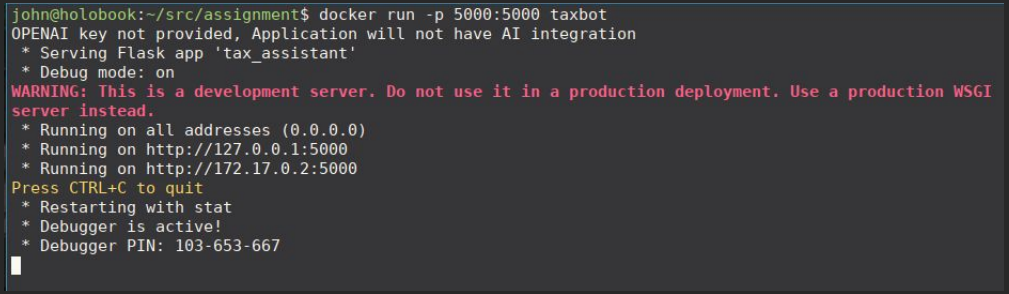
Step 4&5: Difficulties faced

- Github website is hard to navigate
- Documentation in my opinion is all over the place.
- Docker required specific options in flask for testing due to networking configuration issues I did not want to bother with on my machine.

Application showcase

This assumes one has followed the README to set the application up:

```
docker run -p 5000:5000 -e OPENAI_API_KEY=<API KEY HERE> taxbot
```

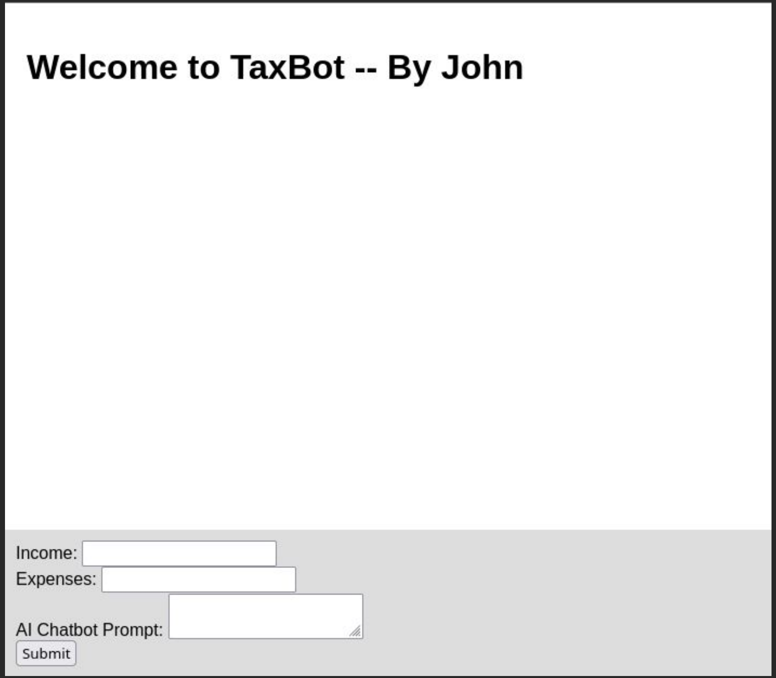


```
john@holobook:~/src/assignment$ docker run -p 5000:5000 taxbot
OPENAI key not provided, Application will not have AI integration
 * Serving Flask app 'tax_assistant'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 103-653-667
```

Open up the browser and write following in the URL: `127.0.0.1:5000`

Application showcase

There are three fields: income, expenses and AI prompt where the user asks the question.

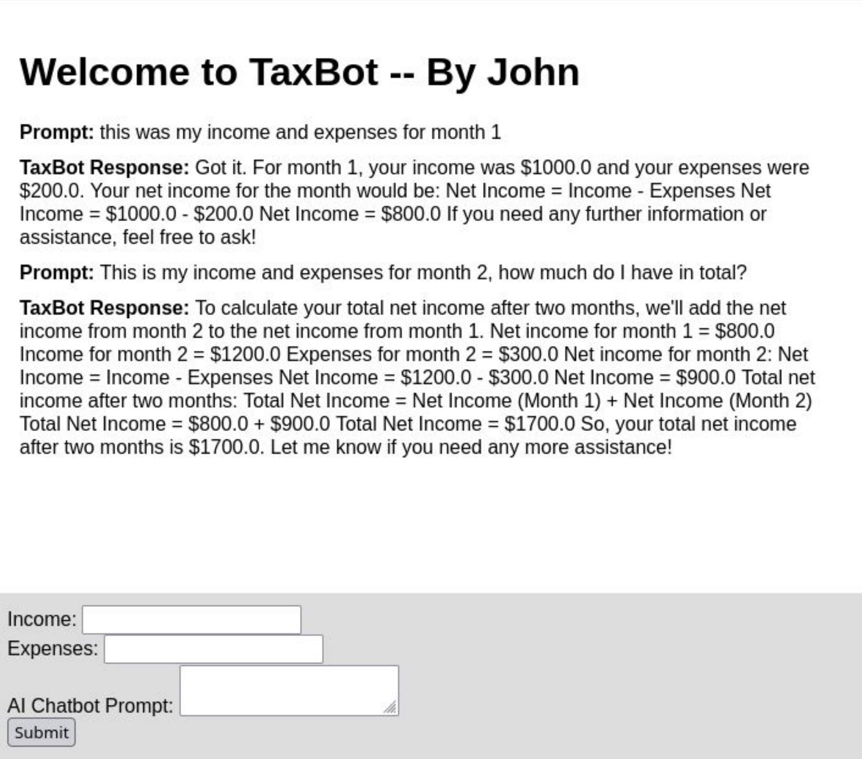


Welcome to TaxBot -- By John

Application showcase

There are three fields: income, expenses and AI prompt where the user asks the question.

The bot has conversation context and additionally, the log is restored on webpage reload.



Welcome to TaxBot -- By John

Prompt: this was my income and expenses for month 1

TaxBot Response: Got it. For month 1, your income was $1000.0 and your expenses were $200.0. Your net income for the month would be: Net Income = Income - Expenses Net Income = $1000.0 - $200.0 Net Income = $800.0 If you need any further information or assistance, feel free to ask!

Prompt: This is my income and expenses for month 2, how much do I have in total?

TaxBot Response: To calculate your total net income after two months, we'll add the net income from month 2 to the net income from month 1. Net income for month 1 = $800.0 Income for month 2 = $1200.0 Expenses for month 2 = $300.0 Net income for month 2: Net Income = Income - Expenses Net Income = $1200.0 - $300.0 Net Income = $900.0 Total net income after two months: Total Net Income = Net Income (Month 1) + Net Income (Month 2) Total Net Income = $800.0 + $900.0 Total Net Income = $1700.0 So, your total net income after two months is $1700.0. Let me know if you need any more assistance!

Future goals(or rewrite goals)

- Actual database security(anyone can delete, read and write to the db)
- Multiple instances
- Prettier webpage that uses external libraries. Also Markdown support.
- Shared chat: already works due to very relaxed security.
- Removal of income,expenses. Instead accept multiple file types like images.

Thank you

Github: https://github.com/svin24

Webpage: https://svin24.github.io/