Simplified Plan for GreenCode AI:

Your frontend looks clean and intuitive! You've done a great job making it user-friendly and visually appealing.

Simplified Plan for GreenCode AI:

Since your project is a **school project** and needs to be done quickly (2-3 days), here's what I recommend based on your setup and needs:

---

1. Backend Framework: Flask (Python)

- **Why Flask?**
  - Lightweight and quick to set up.
  - Perfect for creating REST APIs to process code input from your frontend.
  - Integrates easily with AI models and static analysis tools.

---

2. AI Model for Code Optimization: StarCoder (from Hugging Face)

- **Why StarCoder?**
  - Free, open-source, and specifically designed for coding tasks.
  - Handles multi-language input (Python, JS, etc.).
  - Easily integrated using Hugging Face's `transformers` library.
  - No need for fine-tuning or training from scratch.

Usage Example:

```
from transformers import pipeline
model = pipeline("text-generation", model="bigcode/starcoder")
output = model("Optimize this code for energy efficiency: " + user_code)
```

3. Code Analysis: Creedengo Rules (Manual Implementation)

- **Why Creedengo-like Rules?**
  - Targets energy-wasting patterns (like inefficient loops and memory usage).
  - We'll replicate some of these patterns in a simple way using **Python linters (like Pylint)**.
  - For example:
    - **Detecting nested loops.**
    - **Finding inefficient string concatenations.**
    - **Detecting unnecessary list copying.**

---

4. Energy Estimation: CodeCarbon

- **Why CodeCarbon?**
  - Free and easy to set up.
  - Gives an estimate of CO_2 emissions for Python code.
  - We'll use it to compare the **"before" and "after"** versions of the code.

Usage Example:

```
from codecarbon import EmissionsTracker
tracker = EmissionsTracker()
tracker.start()
# Run your code here
tracker.stop()
```

---

5. Database: SQLite (if needed)

- **Why SQLite?**
  - No server setup needed.

- Great for logging code inputs and suggested optimizations.
- Free and integrated within Python.

Backend Workflow:

1. **Frontend Submission:** User pastes code and clicks “Analyze.”
2. **Static Analysis (Creedengo-like):** Backend checks for common inefficiencies.
3. **AI Enhancement (StarCoder):** Model suggests improvements.
4. **Energy Estimation (CodeCarbon):** Calculates potential CO_2 savings.
5. **Response:** Returns analysis results and suggestions to the frontend.

Why This Setup Works:

- **Fast to build:** Flask and pre-trained models minimize setup time.
- **Free and reliable:** Uses free tools with good documentation.
- **Relevant to your idea:** Combines **AI-driven optimization** with **rule-based analysis** like Creedengo.
- **Sustainable focus:** Estimates CO_2 savings, giving your project real-world impact.

Let me know if you want help with the backend code or integrating any of these components!