FullStack Developer

FuelEU Maritime — Full-Stack Developer Assignment

This assignment assesses your **engineering ability**, **architectural clarity**, and **use of AI agents** (such as *GitHub Copilot*, *Claude Code*, *Cursor Agent*, *OpenAI Codex*, etc.).

You will implement parts of a **Fuel EU Maritime compliance platform**: the frontend dashboard and backend APIs handling route data, compliance balance (CB), banking, and pooling.

🔍 General Objective

Build a minimal yet structured implementation of the Fuel EU Maritime compliance module with:

- **Frontend:** React + TypeScript + TailwindCSS
- **Backend:** Node.js + TypeScript + PostgreSQL
- **Architecture:** Hexagonal (Ports & Adapters / Clean Architecture)
- **Documentation:** AI-agent usage markdowns (mandatory)
- **Focus:**
  - Quality of domain modeling and separation of concerns
  - Use of AI-agents efficiently (generation, refactoring, testing)
  - Proper explanation of how those tools were used

🧠 AI Agent Usage & Documentation (Mandatory)

You **must** include the following markdown files:

1. `AGENT_WORKFLOW.md`

Document your agent usage, with the following sections:

```
# AI Agent Workflow Log ## Agents Used List all agents (Copilot, Cla
ude Code, Cursor Agent, etc.) ## Prompts & Outputs - Example 1: incl
ude the exact prompt and the generated snippet - Example 2: show how
you refined or corrected an output ## Validation / Corrections Descr
ibe how you verified and modified the agent output. ## Observations
- Where agent saved time - Where it failed or hallucinated - How you
combined tools effectively ## Best Practices Followed (e.g., used Cu
rsor's `tasks.md` for code generation, Copilot inline completions fo
r boilerplate, Claude Code for refactoring)
```

2. `README.md`

Include:

- Overview
- Architecture summary (hexagonal structure)
- Setup & run instructions
- How to execute tests
- Screenshots or sample requests/responses

3. `REFLECTION.md`

Short essay (max 1 page):

- What you learned using AI agents
- Efficiency gains vs manual coding
- Improvements you'd make next time

🔷 FRONTEND TASK — React + Tailwind

🎯 Objective

Create a **Fuel EU Compliance Dashboard** with four tabs:

1. **Routes**
2. **Compare**
3. **Banking**
4. **Pooling**

All values and API responses originate from the backend service described below.

🧩 Architecture (Hexagonal pattern)

```
src/ core/ domain/ application/ ports/ adapters/ ui/ infrastructure/
shared/
```

- Core = domain entities, use-cases, and ports (no React dependencies)
- UI adapters = React components and hooks implementing inbound ports
- Infrastructure adapters = API clients implementing outbound ports
- Styling via TailwindCSS

🧱 Functional Requirements

(1) Routes Tab

- Display table of all routes fetched from `/routes`
- Columns: routeId, vesselType, fuelType, year, ghgIntensity (gCO_2e/MJ), fuelConsumption (t), distance (km), totalEmissions (t)
- “Set Baseline” button → calls `POST /routes/:routeId/baseline`
- Filters: vesselType, fuelType, year

(2) Compare Tab

- Fetch baseline + comparison data from `/routes/comparison`
- Use target = **89.3368 gCO_2e/MJ** (2 % below 91.16)
- Display:
  - Table with baseline vs comparison routes
  - Columns: ghgIntensity, % difference, compliant (✅ / ❌)
  - Chart (bar/line) comparing ghgIntensity values
- Formula:

  `percentDiff = ((comparison / baseline) − 1) × 100`

(3) Banking Tab

Implements Fuel EU **Article 20 – Banking**.

- `GET /compliance/cb?year=YYYY` → shows current CB
- `POST /banking/bank` → banks positive CB
- `POST /banking/apply` → applies banked surplus to a deficit
- KPIs:
  - `cb_before`, `applied`, `cb_after`
- Disable actions if CB ≤ 0; show errors from API

(4) Pooling Tab

Implements Fuel EU **Article 21 – Pooling**.

- `GET /compliance/adjusted-cb?year=YYYY` → fetch adjusted CB per ship
- `POST /pools` → create pool with members
- Rules:
  - Sum(adjustedCB) ≥ 0
  - Deficit ship cannot exit worse
  - Surplus ship cannot exit negative
- UI:
  - List members with before/after CBs
  - Pool Sum indicator (red/green)
  - Disable “Create Pool” if invalid

📊 KPIs Dataset (for mock or seed data)

routeId	vesselType	fuelType	year	ghgIntensity	fuelConsumpti
R001	Container	HFO	2024	91.0	5000
R002	BulkCarrier	LNG	2024	88.0	4800
R003	Tanker	MGO	2024	93.5	5100
R004	RoRo	HFO	2025	89.2	4900
R005	Container	LNG	2025	90.5	4950

✅ Evaluation Checklist

Area	Criteria
Architecture	Proper hexagonal separation (core ↔ adapters)
Functionality	Routes, Compare, Banking, Pooling tabs work as specified
Code Quality	TS strict mode, ESLint/Prettier, clean naming
UI	Responsive, accessible, clear data visualization
AI-Agent Use	Quality and depth of AGENT_WORKFLOW.md + prompts
Testing	Unit tests for use-cases and components

🔶 BACKEND TASK — Node.js + TypeScript + PostgreSQL

🎯 Objective

Build APIs backing the Fuel EU Dashboard:

- Manage Routes & Comparisons
- Calculate Compliance Balance (CB)
- Handle Banking and Pooling logic

⚙️ Architecture (Hexagonal)

```
src/ core/ domain/ application/ ports/ adapters/ inbound/http/ outbo
und/postgres/ infrastructure/ db/ server/ shared/
```

Use dependency-inverted modules:

core → ports → adapters.

Frameworks (Express/Prisma/etc.) only in adapters/infrastructure.

🧱 Database Schema

Table	Key Columns	Purpose
routes	id, route_id, year, ghg_intensity, is_baseline	basic data
ship_compliance	id, ship_id, year, cb_gco2eq	computed CB records
bank_entries	id, ship_id, year, amount_gco2eq	banked surplus
pools	id, year, created_at	pool registry
pool_members	pool_id, ship_id, cb_before, cb_after	allocations

Seed the five routes above; set one baseline = true.

🧮 Core Formulas

- **Target Intensity (2025)** = 89.3368 gCO_2e/MJ
- **Energy in scope (MJ)** ≈ fuelConsumption × 41 000 MJ/t
- **Compliance Balance** = (Target − Actual) × Energy in scope
- Positive CB → Surplus ; Negative → Deficit

🔗 Endpoints

`/routes`

- `GET /routes` → all routes
- `POST /routes/:id/baseline` → set baseline
- `GET /routes/comparison` → baseline vs others
  - `percentDiff` and `compliant` flags

`/compliance`

- `GET /compliance/cb?shipId&year`
  - Compute and store CB snapshot
- `GET /compliance/adjusted-cb?shipId&year`
  - Return CB after bank applications

`/banking`

- `GET /banking/records?shipId&year`
- `POST /banking/bank` — bank positive CB
- `POST /banking/apply` — apply banked surplus
  - Validate amount ≤ available banked

`/pools`

- `POST /pools`
  - Validate ∑ CB ≥ 0
  - Enforce:
    - Deficit ship cannot exit worse
    - Surplus ship cannot exit negative
  - Greedy allocation:
    - Sort members desc by CB
    - Transfer surplus to deficits
  - Return `cb_after` per member

🧪 Testing Checklist

- **Unit** — ComputeComparison, ComputeCB, BankSurplus, ApplyBanked, CreatePool
- **Integration** — HTTP endpoints via Supertest
- **Data** — Migrations + Seeds load correctly
- **Edge cases** — Negative CB, over-apply bank, invalid pool

---

✅ Evaluation Checklist

Area	Criteria
Architecture	Ports & Adapters; no core ↔ framework coupling
Logic Correctness	CB, banking, pooling math matches spec
Code Quality	TypeScript strict, tests pass, ESLint clean
Docs	AGENT_WORKFLOW.md + README complete
AI Agent Use	Clarity of prompts, logs, and validation steps

📦 Submission Instructions

1. Create a **public GitHub repository** with two folders:
   - `/frontend`
   - `/backend`
2. Include:
   - `AGENT_WORKFLOW.md`
   - `README.md`
   - `REFLECTION.md`
3. Ensure `npm run test` and `npm run dev` both work.
4. Commit history must show incremental progress (not one single dump).
5. Deadline: *72 hours from assignment receipt*.

📘 Reference

All constants, CB formula, and banking/pooling rules follow

Fuel EU Maritime Regulation (EU) 2023/1805, Annex IV and Articles 20–21 (see pp. 27 & 104–107).

2025-May-ESSF-SAPS-WS1-FuelEU-calculation-methodologies.pdf 4 MiB

This is your full brief. Deliver clean code, structured repositories, and transparent documentation of AI-agent collaboration.