Coding Test — Junior PHP Developer (Symfony)

1) Business Context

A company called **ZOO** currently integrates with two shipping carriers:

- `transcompany`
- `packgroup`

Each carrier has its own shipping price formula (currency is always **EUR**):

- **TransCompany**:
  - if parcel weight is **<= 10 kg** → **20 EUR**
  - if parcel weight is **> 10 kg** → **100 EUR**
- **PackGroup**:
  - **1 EUR per 1 kg**

In the future, additional carriers will be added, each with a different pricing formula.

The solution must be extensible.

---

2) Task Objective

Design and implement an OOP-based backend architecture in **PHP (Symfony)** to calculate shipping costs for a selected carrier according to its pricing rules.

Your solution should include:

- domain/service classes,
- API controller(s),
- validation layer,
- an extensible mechanism for adding new carriers without rewriting core logic.

---

3) Functional Requirements

3.1 Backend API

Implement an endpoint that accepts:

- `carrier` (slug or ID),
- `weightKg` (parcel weight),

and returns calculated price in EUR.

Suggested contract:

`POST /api/shipping/calculate`

Request:

```
{
 "carrier": "transcompany",
 "weightKg": 12.5
}
```

Response (success):

```
{
 "carrier": "transcompany",
 "weightKg": 12.5,
 "currency": "EUR",
 "price": 100
}
```

Response (validation/business error example):

```
{
 "error": "Unsupported carrier"
}
```

You will likely need additional endpoints to implement the UI. Therefore, the final list of endpoints depends entirely on your implementation.

3.2 Frontend (simple UI)

Create a minimal web page with:

- numeric input for parcel weight,
- select input for carrier slug/ID,
- button: **Calculate price**,
- result/error output.

`Vue.js` is preferred for the UI (basic implementation is enough).

---

4) Technical Expectations

Use OOP and clean architecture principles (SOLID-friendly design).

Adding a new carrier should require only adding a new strategy class + registration, without modifying existing strategy logic.

---

5) Infrastructure Requirements

Containerize the solution with Docker (preferred):

- `nginx` container
- `php` container (Symfony app)
- `ui` container (Vue app)
- orchestration via `docker-compose.yml`

---

6) Testing

`PHPUnit` tests are welcomed (and strongly recommended):

- unit tests for each pricing strategy,
- unit/integration tests for calculator service and API endpoint behavior.

---

7) Deliverables

Provide:

1. Source code repository
2. Run instructions (`README.md`)
3. Docker setup (`docker-compose.yml`)