# Technical Test — Order Integration (Backend Only)

You will have access to two existing endpoints and must consume them while applying solid architectural practices typical in Laravel. The goal is to evaluate the clarity of your domain design, code quality, tests, and documentation.

## 1. Scenario

Micros.Services exposes two different API contracts for order creation:

| **Version** | **Path** | **Payload Format** | **Style** |
| --- | --- | --- | --- |
| **v1** | POST https://dev.micros.services/api/v1/order | Flat (root fields) | Traditional REST |
| **v2** | POST https://dev.micros.services/api/v2/order | Nested in data.type / attributes | JSON:API 1.1 |

All monetary values are in **EUR** and tax regulations follow Portuguese rules (9-digit NIF). Authentication is disabled for testing purposes—focus on internal modelling and response/error handling.

### 1.1 Example – v1 (request ↔ response)

**Request**

{

"customer\_name": "João Almeida",

"customer\_nif": "504321123",

"total": "199.90",

"currency": "EUR",

"items": [

{ "sku": "PEN-16GB", "qty": 3, "unit\_price": "9.90" },

{ "sku": "NOTE-A5", "qty": 10, "unit\_price": "12.00" }

]

}

**Response**

{

"data": {

"uuid": "e3d4b1d2-97db-4e5d-a7f5-7c9f7b1c2e10",

"number": "ORD-2025-00001",

"status": "created",

"total": "199.90",

"currency": "EUR",

"created\_at": "2025-07-22T14:12:09Z"

}

}

### 1.2 Example – v2 (request ↔ response)

**Request**

{

"data": {

"type": "orders",

"attributes": {

"customer": {

"name": "João Almeida",

"nif": "504321123"

},

"summary": {

"currency": "EUR",

"total": "199.90"

},

"lines": [

{ "sku": "PEN-16GB", "qty": 3, "price": "9.90" },

{ "sku": "NOTE-A5", "qty": 10, "price": "12.00" }

]

}

}

}

**Response**

{

"links": {

"self": "https://micros.services/api/v2/order/ORD-2025-00001"

},

"data": {

"type": "orders",

"id": "ORD-2025-00001",

"attributes": {

"uuid": "e3d4b1d2-97db-4e5d-a7f5-7c9f7b1c2e10",

"status": "created",

"currency": "EUR",

"total": "199.90",

"created\_at": "2025-07-22T14:12:09Z"

}

}

}

## 2. Challenge

1. **Domain Modelling**
   * Create **Value Objects** for Money (amount + currency) and TaxId (NIF).
   * Implement an **Order** entity composed of lines (items) and the customer.
2. **Application Layer**
   * Develop input/output structures for each API version.
   * Implement a service that receives those structures, transforms them into Value Objects/entities, and calls the remote endpoint via Laravel HTTP Client.
3. **Version Selection & Serialization**
   * Provide a mechanism to choose the v1 or v2 connector at runtime.
   * Keep serialization/deserialization in one place; avoid scattered formatting logic.
4. **Tests**
   * Write unit tests (Pest) covering:  
     a) total calculations,  
     b) request construction,  
     c) response parsing.
   * Add integration tests using HTTP fakes.
5. **Documentation**
   * Supply a README explaining design decisions, short diagrams (if useful), and instructions to run the project.
   * Indicate which principles were applied and where.

## 3. Non-Functional Requirements

* PHP 8.2+ · Laravel 10^
* Code in English with concise comments.
* Follow PSR-12.
* Keep the project lean—avoid unnecessary dependencies.

**Note:** This task is **backend-only**; no frontend work is required.