🎯 Programa Take-Home MLE Challenge

👋 Welcome

Thank you for your interest in joining the Data & AI team at Programa! We're excited to see what you can build.

This take-home challenge is designed to give you an opportunity to demonstrate your technical skills in a realistic scenario that reflects the kind of problems we solve at Programa. There are no trick questions—we genuinely want to see how you approach problem-solving, structure your code, and handle real-world data challenges.

⏱️ Timeline & Expectations

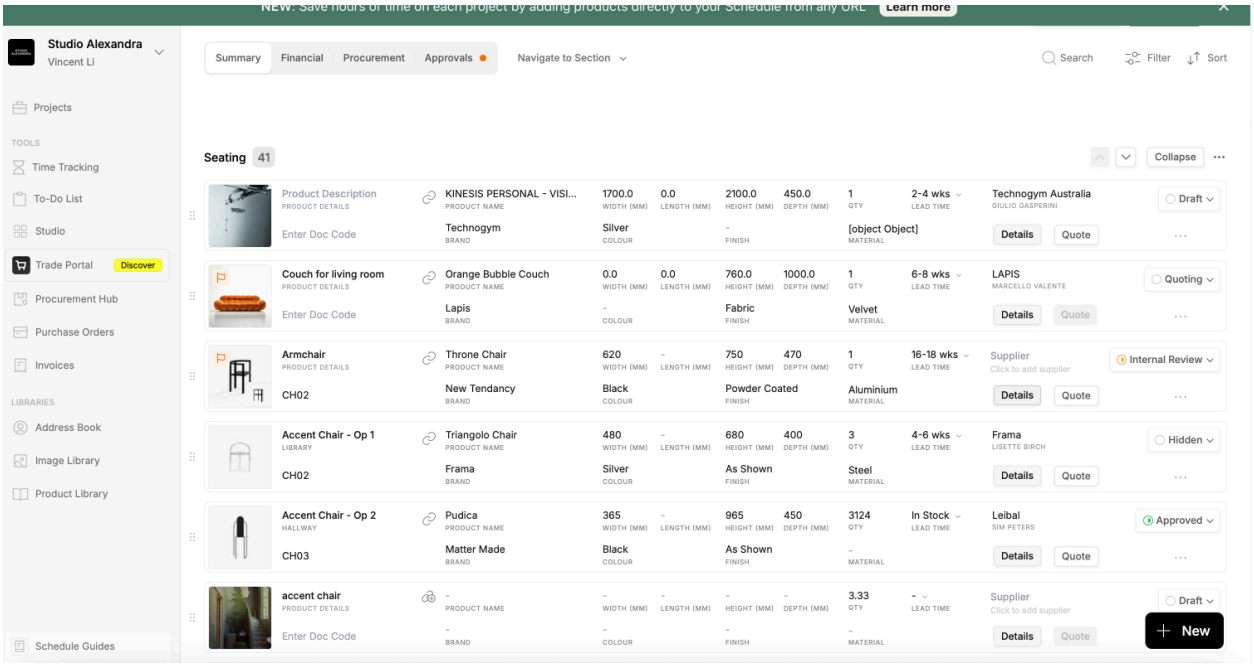
- **Duration:** 5 calendar days from when you receive this challenge
- **Estimated effort:** 8–12 hours (you don't need to spend all 5 days on this!)

Interior Schedule Extraction API

🧠 Objective

Interior designers often create Excel-based product schedules containing product names, materials, finishes, prices, dimensions—and sometimes images.

Your task is to **build a Python-based REST API** that accepts `.xlsx` file uploads and converts them into structured JSON format so that we can import the schedules into programa platform.



The primary challenge lies in the **variability of schedule formats** from different designers—including inconsistent column names, merged cells, multiple sheets, and diverse layouts. The goal is to **accurately extract and standardize** product data to support seamless schedule imports into our platform.

🛠️ Requirements

1. Build an API

- Use **FastAPI** or **Flask**
- Implement the endpoint:
  - `POST /parse`
  - Accepts an Excel `.xlsx` file upload (you will be provided with a sample of schedules)
  - Returns structured JSON in the format specified below
- Ensure the implementation is **modular, clean, and production-ready**

2. Extract Data

Expected fields to extract:

Field	Description	Type
`doc_code`	Drawing or reference code	string
`product_name`	Product display name	string
`brand`	Product brand	string
`colour`	Colour name	string
`finish`	Surface finish	string
`material`	Main material	string
`width`	Width in mm	integer
`length`	Length in mm	integer
`height`	Height in mm	integer
`qty`	Quantity	integer
`rrp`	Recommended retail price	decimal
`feature_image`	Image filename (optional)	string
`product_description`	Short description	string
`product_details`	Additional specifications	string

- Input data may span **multiple sheets**
- Expect **merged cells**, inconsistent headers, and missing values

---

📩 Expected JSON Output Format

```
{
  "schedule_name": "Lighting Schedule",
  "products": [
    {
      "doc_code": "L1",
      "product_name": "Minimalist Pendant Light",
      "brand": "Lighting Co",
      "colour": "Black",
      "finish": "Brushed Brass",
      "material": "Metal",
      "width": 300,
```

```
      "length": 300,
      "height": 600,
      "qty": 1,
      "rrp": 489,
      "feature_image": "blackpendantlight.jpg",
      "product_description": "A simple pendant light ideal for modern interiors.",
      "product_details": "Install at 2.4m height; supplied with dimmable bulb."
    }
  ]
}
```

⚙️ Suggested Libraries

You may use any Python libraries you see fit. For Excel parsing, you might find libraries like `openpyxl` or `pandas` useful.

🥾 Bonus Points

These features are **not required**, but may earn you bonus credit:

- Extract embedded images or image URLs (if available)
- Handle merged cells gracefully
- Handle inconsistent headers and edge cases robustly
- Include a `Dockerfile` for containerization
- Include basic test cases
- Include a **system architecture diagram** (optional but encouraged) showing how you'd deploy this API in a production environment

📁 Project Structure

- Place any sample `.xlsx` files under `/data`
- Keep code modular (e.g., `/app/parser.py`, `/app/models.py`)
- Include a `README.md` with:

- Setup instructions
- Example request/response
- Optional architecture diagram

🧪 Evaluation Criteria

- **Code quality**: Clean, modular, well-documented
- **Robustness**: Handles diverse real-world input formats
- **Accuracy**: JSON output structure and correctness
- **API design**: RESTful and resilient
- **Bonus**: Docker, tests, image support, architecture diagram

---

📤 Submission

- Upload your solution to GitHub
- Include a detailed system architecture diagram showing your proposed production deployment setup
- Invite: `joshiain` , `foozieakbar` and `vinceliorg` as collaborators
- Share the repository link with us via email

✨ **Note:** Image support is a nice-to-have, not a requirement. Focus on accuracy, robustness, and thoughtful code design. Good luck, and have fun!