Toolkit Integration Engine
Ten hour evaluation exercise for an experienced Python developer

1. Objective

Design and implement a small but real Toolkit Integration Engine in Python that

1. Accepts a named product, starting with at least one CRM system.
2. Locates and understands that product's public API description.
3. Identifies which endpoints and entities are likely to be useful for a generic toolkit.
4. Where there is ambiguity, presents choices to a user such as "select the elements you require".
5. Generates a working API integration layer that can be reused by other parts of the toolkit.

The first focus is CRM systems, for example HubSpot, Pipedrive, Salesforce, or similar. The architecture should make it straightforward to extend to other application types later, especially accounting systems such as Xero and QuickBooks.

2. Scope for a ten hour effort

The exercise should be achievable but stretching for roughly ten hours of work. The goal is not to finish a production ready product, but to show design thinking, code quality, and judgement.

Within that time, the candidate should aim to deliver:

1. A small service or library in Python that exposes a simple interface such as
   - `register_product(product_id, product_type)`
   - `discover_capabilities(product_id)`
   - `generate_integration(product_id, selected_capabilities)`
2. A working integration for at least one CRM product, with the ability to:
   - Authenticate against the CRM API using a sensible method for that product.
   - Retrieve metadata or an API specification, for example OpenAPI or similar, or a hand crafted description if discovery is difficult.
   - Identify core CRM entities such as contacts, organisations, deals, activities.
3. A simple way for a human user to resolve uncertainty, for example:
   - A small command line prompt flow, or
   - A minimal web interface with FastAPI or Flask, or
   - A configuration driven approach where the tool prints suggestions and accepts input.
4. A generated integration layer that:
   - Provides normalised operations such as `list_contacts`, `get_contact`, `create_contact`, `list_organisations`, and so on.
   - Hides product specific details behind a simple internal interface.
5. A short design note that explains how the same approach would extend to:

- Another CRM, even if not fully implemented.
- An accounting product such as Xero or QuickBooks, which may have different core entities and auth models.

3. Suggested functional requirements

Within the exercise, the candidate should attempt to cover the following.

3.1 Product registration and discovery

1. Represent a product with a minimal definition, for example
   - Name
   - Type, for example `crm` or `accounting`
   - API base URL
   - Auth method details
2. Implement discovery for the chosen CRM, for example:
   - Use an OpenAPI document if available.
   - If not easily discoverable, define a small internal schema that describes endpoints, HTTP methods, request and response shapes.
3. From the specification or schema, extract likely useful elements for a toolkit:
   - Entities, for example contacts, companies, deals.
   - Actions, for example list, get by id, create, update.

3.2 Usefulness ranking and user choice

1. Implement a simple ranking or selection of "useful" endpoints:
   - For example, prefer endpoints that handle core entities and standard CRUD operations.
   - Exclude clearly niche or low level endpoints.
2. When the engine is not sure, it should present choices such as:
   - "These endpoints look similar, please select the ones to include in the integration."
   - "These fields look like identifiers, please confirm which one should be treated as the primary key."
3. Capture the user's choices and store them in configuration for later reuse.

3.3 Integration generation

1. Generate a small integration module or class that:
   - Implements a stable internal interface for the toolkit, for example `ToolkitCRMClient`.
   - Maps internal method calls to the chosen product's API endpoints.
   - Handles authentication, basic error handling, and response mapping.
2. For the evaluation, it is acceptable if generation is:
   - Code written to disk, or
   - Dynamic class creation at runtime, or
   - Building configuration consumed by a generic CRM client.
3. Provide at least one concrete end to end use case, for example:

- Initialise client with credentials.
- List contacts.
- Create or update a contact.

4. Technical expectations

The exercise is intended for an experienced Python developer, so the evaluation will look for:

1. **Language use**
   - Modern Python, type hints, clear naming, sensible packaging.
2. **API and HTTP handling**
   - Use of a solid HTTP library, for example `httpx` or `requests`.
   - Sensible retry and timeout handling at a basic level.
3. **Structure and design**
   - Separation between product specific details and generic toolkit interfaces.
   - A clear extension point where new product types such as Xero and QuickBooks could be plugged in without rewriting core logic.
4. **Testing**
   - At least a small set of unit tests for core components.
   - Use of mocks or stubs for external API calls.
5. **Developer experience**
   - A short `README` that explains how to run the code and how the design is intended to grow.
   - Reasonable logging and error messages.

5. Evaluation criteria

When reviewing the ten hour submission, assess:

1. **Clarity of architecture**
   - Can you see how to add another CRM or move into accounting products without redesign.
2. **Correctness and robustness**
   - Does it actually talk to a real CRM API.
   - How does it behave on auth failure, bad input, or network issues.
3. **Usefulness of the “useful element” selection**
   - Does the engine reasonably identify core entities and actions.
   - Is the user choice flow practical and understandable.
4. **Quality of generated integration**
   - Is the resulting integration pleasant to use from the toolkit’s point of view.
   - Are abstractions sensible, for example no leaking of odd product specific details.
5. **Code quality**
   - Readability, structure, tests, and documentation.

6. Extension direction, not required but positive

If time allows, the candidate can:

1. Sketch a second product within the same type, for example a second CRM, and show how configuration or small adapters cover the differences.
2. Outline a mapping for one accounting product such as Xero, listing likely core entities and suggesting how the engine would approach discovery and selection there.