ALT/FNDATA ACADEMY · MODULE 12

Data engineering and the API

This session moves students from querying data in a browser to understanding how a production data API is structured and how a small pipeline is built on top of it. Students start in the no-code sandbox, using its discovery views to see what a table catalog and a schema endpoint actually expose, then move to the documented shape of the production API itself:

90 minutes · sandbox.altfndata.com

SESSION AT A GLANCE

What today looks like

- Length: 90 minutes. A 60 minute and a 120 minute variant are noted under Teaching notes.
- Level: undergraduates or graduate students who have taken an introductory programming or data science course. The core session assumes no prior API experience; the optional extension assumes basic Python familiarity.
- Access: the no-code sandbox at sandbox.altfndata.com opens the session, and it carries the core 70 minutes on its own. The optional extension has a substantial API and code component and requires an instructor class API key requested from info@altfndata.com, plus the downloadable Python client or tutorials notebook at docs.altfndata.com.
- Goal of the session: students leave able to describe the shape of the production query endpoint, explain why it paginates on offset, and, if time allows, run a small client-side ETL that pulls one brand's rows and computes an aggregate themselves.

OBJECTIVES

By the end of class students can

- Use the sandbox's discovery views to identify which tables exist and which fields a given table exposes, and explain why discovery should precede any query design.
- Describe the shape of the production API's query endpoint, including the request method, the required header, and the four parts of the request body.
- Explain why the API paginates on an offset rather than a page number, and describe how a client would retrieve a full result set larger than one page.
- Distinguish a query that filters and sorts rows from a query that aggregates, and explain why this API returns rows and leaves aggregation to the client.
- (Extension) Use the reusable Python client to authenticate a request, retrieve paginated results for a single brand, and compute a simple aggregate, such as a count or an average, from the retrieved rows.
- Identify the operational limits of the API relevant to a data engineering context, including the page size ceiling and the fact that API keys are issued manually by the team rather than self-served.

AGENDA

How the session runs

- 0 to 10 min: Introduce the session's frame, understanding a data provider's contract before building on it, and preview the two discovery endpoints and the single query endpoint every table shares.
- 10 to 25 min: Sandbox orientation and discovery. Students use the sandbox's table list and schema tab to answer the question, what tables exist and what fields does each one expose.
- 25 to 40 min: Guided walkthrough of the production API's query endpoint shape, the request body's four parts, and the response envelope, using the documentation and worked examples rather than live code.
- 40 to 55 min: Guided walkthrough of pagination on offset, using a worked example that shows what a client must do to retrieve more rows than fit on one page.
- 55 to 70 min: Optional code extension, instructor demo using the class API key and the Python client to pull one brand's rows and compute a client-side aggregate.
- 70 to 85 min: Small-group exercise, students design (on paper or in the sandbox) the query body they would send to answer a specific question about a chosen brand, and if doing the code extension, run it.
- 85 to 90 min: Wrap-up and homework assignment.

LIVE DEMO

The in-class walkthrough

- Open sandbox.altfndata.com, sign in, and open the table list view. Ask students to identify how many production tables exist and to name three.
- Open the schema tab for one table, for example the handbags data, and have students list every documented field shown, noting that a query should only ever reference fields that appear here.
- Explain that the production API mirrors this same discovery pattern outside the sandbox, through GET /v1/tables, which lists every table, and GET /v1/tables/{name}/schema, which lists a given table's fields, and that a well-built client always calls these before constructing a query.
- Walk through the shape of the query endpoint itself, POST /v1/tables/{name}/query, sent with an X-API-Key header, and a JSON body containing fields, filters, sort, limit, and offset. Show the worked example request body below on the projector.
- Walk through the response envelope, table, result_count, and data, and point out that the API returns individual rows in data, not a pre-computed total or average.
- Introduce pagination on offset with a worked example, explain that a single request returns at most around 1,000 rows, and that a client retrieving a brand with 3,400 matching rows must send four requests, each advancing offset by the number of rows already retrieved, stopping once a response returns fewer rows than the limit requested.

DISCUSS

Questions for the room

- Why should a client always call the discovery endpoints before constructing a query, rather than guessing at table and field names?
- What would go wrong for a client that assumed a single request could return an unlimited number of rows, and how does the offset-based pagination pattern prevent that failure mode?
- Why might a data provider choose to return raw rows rather than server-side aggregates, and what tradeoff does that design place on the client?
- The API exposes 11 filter operators, including eq, like, in, and is_null. What kinds of questions can be answered with these operators alone, and what kinds would require something more expressive?
- Why does the API require a manually issued key rather than self-service key generation, and what does that imply about how a course would provision access for a class?

HOMEWORK

This week's assignment

- Each student writes a one-page data engineering brief describing a small ETL they would build on top of the API to answer a specific business question of their choosing, using only documented fields and any of the all_*_data tables.
- The brief must specify the exact discovery calls the client would make first, the exact query body or bodies it would send, how it would handle pagination if the result exceeds one page, and what aggregate or transformation it would compute client-side from the retrieved rows.
- Students who complete the optional code extension may submit their working script in place of the written query bodies, along with a short paragraph explaining the pagination logic.
- The brief must include an explicit limitations section addressing the manual key-issuance process and the fact that this dataset's newest quarters are still being ingested.
- Grading criteria:

NEXT

Wrapping up

- Restate the one-sentence takeaway before students leave.
- Point to the facilitator guide for discussion guidance and teaching notes.
- Questions or a class API key: info@altfndata.com