

Ask Your Data — System Description

Oct 12, 2026 · @Edwin Muchalanje

Ask Your Data is a Django web application that lets anyone upload a spreadsheet and ask questions about it in plain English; Claude writes and runs read-only SQL to answer, and every figure shown is computed by the app from a real query.

1. Overview

Users: business users with data in spreadsheets (sales, HR, marketing) who need answers without writing SQL; analysts who want to verify the method.

Key capabilities

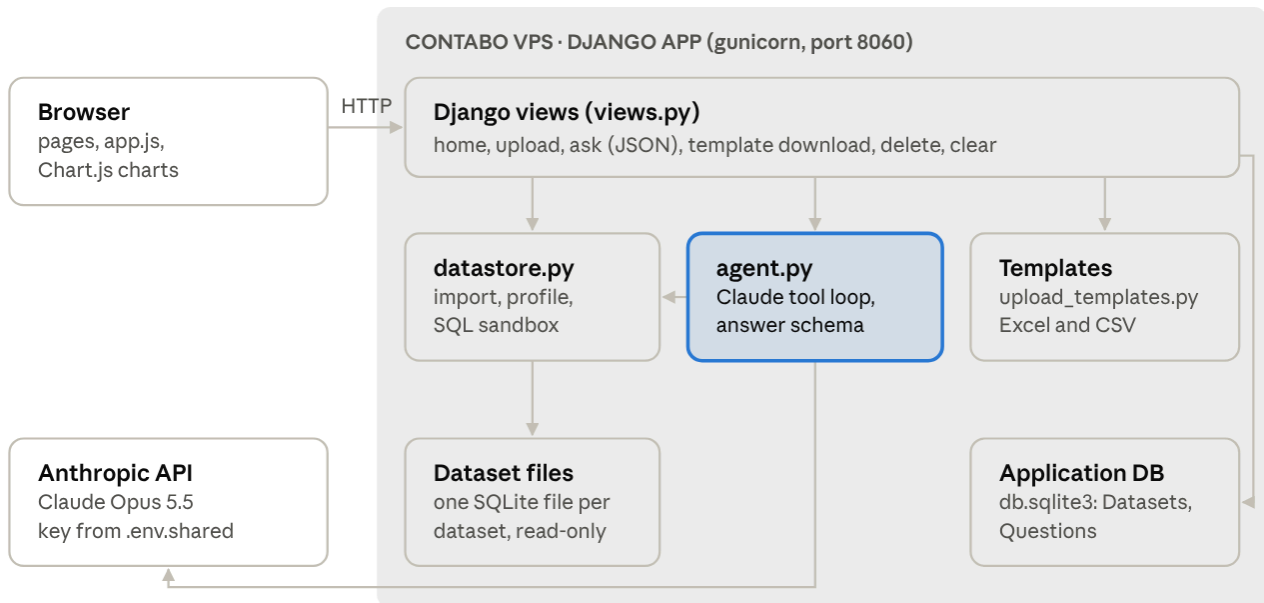
Capability	What the system does
Data upload	Accepts CSV and Excel up to 25 MB; cleans column names, detects numbers and dates, profiles every column
Upload templates	Generates Excel (with an Instructions sheet) and CSV starter files for sales, employees and marketing
Natural-language questions	Claude explores the data with up to 8 read-only SQL queries, then answers
Verifiable answers	Summary, chart (line, bar or pie), result table, every query run, copyable final SQL
Conversations	Follow-up suggestions; the last 3 answers are used as context
Sample datasets	Retail Sales (9,000 orders), Employee Records (840 people), Marketing Campaigns (624 weekly rows)

Companion document: *Business Analysis & Functional Specification* (process maps, data model detail, requirements).

2. Architecture

The system is a single Django application: the browser talks only to Django views, `agent.py` is the only module that calls Claude, and `datastore.py` is the only one that touches dataset files.

Claude only proposes SQL; the Django app runs it against per-dataset files



System architecture · browser, Django app modules, storage and the Anthropic API

Claude's SQL reaches the data only through `datastore.py`'s sandbox, and the app database holds metadata and answers, never the uploaded rows.

3. Components and modules

The code is one Django project (`config`) with one app (`analyst`); three modules hold the core logic and the rest is thin web glue.

Module	Responsibility
<code>analyst/datastore.py</code>	Reads uploads (CSV encoding and delimiter detection, Excel), cleans column names, converts types, writes one SQLite file per dataset, builds the column profile, runs sandboxed read-only queries

Module	Responsibility
<code>analyst/agent.py</code>	The AI analyst: system prompt, <code>run_sql</code> tool, answer JSON schema, tool-use loop, error mapping, re-running the final SQL, chart validation
<code>analyst/upload_templates.py</code>	Builds template files from the sample generators; styles the Excel sheet and writes the Instructions sheet
<code>analyst/views.py</code>	HTTP handlers: home, upload, dataset workspace, ask (JSON), template download, delete, clear history
<code>analyst/models.py</code>	<code>Dataset</code> and <code>Question</code> models (application database)
<code>analyst/management/commands/load_samples.py</code>	Generates the three sample datasets deterministically (fixed random seeds)
<code>templates/</code>	Server-rendered pages: base layout, home, dataset workspace
<code>analyst/static/analyst/app.js</code>	Chat UI: sends questions, shows progress, renders answers, Chart.js charts, tables, audit trail, follow-up chips
<code>analyst/static/analyst/app.css</code>	Styling with light and dark themes, responsive down to phone width
<code>analyst/tests.py</code>	8 offline tests using a simulated Claude client
<code>config/settings.py</code>	Environment-driven settings; loads the project <code>.env</code> then the shared portfolio secrets file

4. Key workflows

Three workflows cover the whole system; the question workflow is the only one that calls the AI.

4.1 Upload and import

1. The browser posts the file to `/upload/` ; files over 25 MB are rejected.

2. CSV bytes are decoded (UTF-8, UTF-8 with BOM, then Latin-1) and the delimiter is sniffed (comma, semicolon, tab); Excel is read with openpyxl.
3. Column names are cleaned (lower case, `_` separators, unique).
4. Text columns that are at least 95% numeric become numbers (commas removed); at least 90% date-like become ISO dates.
5. A `Dataset` row is created and the data is written to `data/datasets/dataset_<id>.sqlite3` as table `data`, inside one transaction.
6. The column profile (types, ranges, distinct counts, examples, nulls, 8-row preview) is saved on the dataset, and the browser is sent to the workspace.

4.2 Answering a question

1. The browser posts `{"question": "..."} to /d/<id>/ask/` (CSRF-protected).
2. The app sends Claude the system prompt, the dataset profile (cached), the last 3 Q&As and the question, with one tool: `run_sql`.
3. For each `run_sql` call the app runs the query in the read-only sandbox and returns up to 60 rows as text, or the SQL error.
4. When Claude is done it returns JSON matching the answer schema; on the 8th step tools are switched off so it must answer.
5. The app re-runs `final_sql` (up to 200 rows) and keeps the chart only if its columns exist in the result.
6. The `Question` is saved with answer, SQL, chart, result, steps, tokens and duration, and returned to the browser, which renders it.

4.3 Template download

1. The browser requests `/templates/<kind>.<xlsx|csv>`; unknown kinds or formats return 404.
2. The app generates 15 example rows with the sample-data generator (fixed seed) and returns them as CSV (UTF-8 with BOM, so Excel opens it correctly) or as a styled Excel workbook with an Instructions sheet.

5. Interfaces

The system exposes seven HTTP endpoints and makes one kind of outbound call, to the Anthropic Messages API.

HTTP endpoints

Method	Path	Purpose	Response
GET	/	Home: datasets, upload card, templates	HTML
POST	/upload/	Upload a file (multipart, field <code>file</code> , optional <code>name</code>)	Redirect to workspace
GET	/templates/<kind>.<fmt>	Download template; <code>kind</code> $\in$ sales, employees, marketing; <code>fmt</code> $\in$ <code>xlsx</code> , <code>csv</code>	File
GET	/d/<id>/	Dataset workspace with conversation history	HTML
POST	/d/<id>/ask/	Ask a question, body <code>{"question": str}</code> (1-500 chars)	JSON answer, or <code>{"error"}</code> with 400/502
POST	/d/<id>/clear/	Delete the dataset's conversation	Redirect
POST	/d/<id>/delete/	Delete an uploaded dataset and its file	Redirect

The Django admin is also mounted at `/admin/`.

Answer JSON (from `/ask/`): `id`, `question`, `answer`, `final_sql`, `chart {type, x, y[], title}`, `result {columns[], rows[], truncated}`, `steps[]`, `follow_ups[]`, `error`, `duration_ms`, `tokens`.

Claude API integration

Setting	Value	Why
SDK	<code>anthropic</code> Python SDK 1.13	Official client; built-in retries on rate limits and server errors
Model	<code>claude-opus-5-5</code> (env <code>ANTHROPIC_MODEL</code>)	Strongest default for multi-step reasoning over data
Effort	<code>medium</code> (env <code>ANTHROPIC_EFFORT</code>)	Balance of speed and depth; <code>low</code> is cheaper
Tool	<code>run_sql</code> with <code>strict: true</code>	Tool inputs always match the schema

Setting	Value	Why
Output	<code>output_config.format</code> JSON schema	Answer always parses: answer, final SQL, chart, follow-ups
Prompt caching	<code>cache_control</code> on the dataset profile	Repeat questions reuse the cached system prompt
Refusal fallback	Beta <code>server-side-fallback-2026-07-01</code> , <code>fallbacks: "default"</code>	A declined request is retried on a fallback model server-side
Limits	<code>max_tokens</code> 16,000; up to 8 steps (env <code>MAX_AGENT_STEPS</code>)	Bounds cost and time per question

6. Security and data protection

The main risk is that AI-generated SQL could damage or leak data; five independent layers make every query read-only and limit it to one dataset, and the tests try to break each one.

SQL sandbox layers

#	Layer	Blocks	Tested
1	Statement check: one statement, must start with <code>SELECT</code> or <code>WITH</code>	<code>DELETE</code> , <code>INSERT</code> , <code>DROP</code> , <code>PRAGMA</code> , <code>ATTACH</code> , stacked statements	Yes
2	Read-only connection (<code>mode=ro</code>)	Any write at the SQLite level	Yes
3	SQLite authorizer: only read, select, function and recursive actions	Writes hidden inside a CTE, <code>load_extension</code> , file functions	Yes
4	One database file per dataset	Reading another user's dataset	By design
5	5-second progress-handler timeout and 500-row cap	Runaway or infinite recursive queries, huge results	Yes

Secrets

- `ANTHROPIC_API_KEY` is read from `/root/portfolio/.env.shared` (permissions 600, outside every git repository), shared by all portfolio projects.
- Django's `SECRET_KEY`, debug flag and allowed hosts come from the project `.env`, which is git-ignored; `.env.example` documents the variables with no values.

Web security

- CSRF protection on every POST, including the JSON `/ask/` endpoint.
- Debug mode off in the public deployment; `ALLOWED_HOSTS` and `CSRF_TRUSTED_ORIGINS` limited to the server's address.
- All user and AI text is HTML-escaped before rendering in the browser; only **`**bold**`** and ``code`` are formatted.
- Upload size capped at 25 MB; only CSV, TXT and Excel extensions accepted.

Known gaps

- No user accounts: every visitor sees every dataset, so the public demo must not hold confidential data.
- No per-visitor rate limit on questions yet; API spend should be capped with a monthly limit in the Anthropic Console.
- Served over HTTP until a domain and TLS certificate are set up.

7. Technology stack and configuration

The stack is deliberately small: Python, Django and SQLite on the server, plain JavaScript and Chart.js in the browser, no build step.

Layer	Technology	Version
Language	Python	3.12
Web framework	Django	5.2
AI	Anthropic Python SDK, Claude Opus 5.5	1.13
Data processing	pandas, NumPy, openpyxl	3.0, 2.5, 3.1
Databases	SQLite (application DB + one file per dataset)	Bundled with Python
App server	gunicorn, 2 workers, 180 s timeout	26.2
Static files	WhiteNoise	6.12
Front end	Django templates, vanilla JavaScript, Chart.js	Chart.js 4.4.1 (cdnjs)
Fonts	Inter, JetBrains Mono (Google Fonts)	

Configuration (environment variables)

Variable	Default	Purpose
<code>ANTHROPIC_API_KEY</code>	(from shared secrets file)	Claude API key
<code>ANTHROPIC_MODEL</code>	<code>claude-opus-5-5</code>	Model for the analyst
<code>ANTHROPIC_EFFORT</code>	<code>medium</code>	<code>low</code> , <code>medium</code> or <code>high</code>
<code>MAX_AGENT_STEPS</code>	<code>8</code>	Max queries per question
<code>DJANGO_SECRET_KEY</code>	dev-only value	Must be set in production
<code>DJANGO_DEBUG</code>	<code>1</code>	<code>0</code> in production
<code>DJANGO_ALLOWED_HOSTS</code>	<code>localhost,127.0.0.1</code>	Comma-separated host names
<code>DJANGO_CSRF_TRUSTED_ORIGINS</code>	empty	Full origins, e.g. <code>http://host:8060</code>

Fixed limits in code: 25 MB uploads, 500-character questions, 5-second query timeout, 500 rows to the AI (60 shown to it per call), 200 rows in the result table.

8. Deployment, operations and testing

The demo runs on a Contabo VPS under gunicorn on port 8060 at `http://169.58.13.178:8060`; it is a manually started process today, and a systemd + nginx setup with HTTPS is the next step.

Hosting layout

Item	Location
Code (git repository, branch <code>main</code>)	<code>/root/portfolio/ask-your-data/</code>
Virtual environment	<code>/root/portfolio/ask-your-data/venv/</code>
Application database	<code>db.sqlite3</code> in the project folder
Dataset files	<code>data/datasets/dataset_<id>.sqlite3</code>
Project settings	<code>.env</code> (git-ignored)
Shared API key	<code>/root/portfolio/.env.shared</code>

Runbook

```

cd /root/portfolio/ask-your-data
./venv/bin/pip install -r requirements.txt          # dependencies
./venv/bin/python manage.py migrate                # database schema
./venv/bin/python manage.py load_samples           # (re)build sample
datasets
./venv/bin/python manage.py collectstatic --noinput # static files for
WhiteNoise
./venv/bin/gunicorn config.wsgi:application --bind 0.0.0.0:8060 --workers 2 -
-timeout 180

```

Testing

8 automated tests run offline with a simulated Claude client (`python manage.py test analyst`), so they cost nothing and need no network:

Area	What is checked
Import	Column cleaning; text numbers like "1,200" typed as numbers
CSV formats	Comma, semicolon, Latin-1 and BOM files parse
Templates	All 6 template files download and re-upload with 15 rows; bad kinds and formats return 404
Sandbox	Deletes, stacked statements, <code>PRAGMA</code> and writes inside a CTE are rejected
AI loop	Tool calls run, a failed query is reported back and recovered from, fallback is requested
Chart validation	A chart naming a missing column falls back to the table
Refusal	A declined request becomes a friendly error
Endpoint	<code>/ask/</code> returns the answer, saves it, and the page shows it in history

A live end-to-end check on 12 Oct 2026 answered "Which category has the best profit margin?" correctly (Apparel, 53.26%) in 11.9 s using 5,317 tokens.

Monitoring

Every question stores `duration_ms`, `input_tokens` and `output_tokens`, so speed and cost per question can be reviewed in the Django admin.

Limitations and roadmap

Item	Plan
Process stops on reboot or session end	Run under systemd behind nginx
HTTP only	Domain + TLS certificate
No per-visitor limits	Rate limit questions per IP
Shared visibility	User accounts with private datasets
Progress shown as rotating messages	Stream real progress with Server-Sent Events
Single table per dataset	Multi-table uploads with joins