CASE STUDY

LEADING DIGITAL BANK TREATS SYNTHETIC DATA AS CODE — EMBEDDING GENROCKET INTO THE FULL ENGINEERING LIFECYCLE

Executive Summary

A leading digital bank — new to market, with a large and rapidly evolving data estate — needed to build confidence in its data products from day one. Waiting until a data model was largely built before preparing test data was creating friction that a modern engineering organization couldn't absorb.

Working with a GenRocket channel partner, the bank adopted a Data-as-Code approach powered by GenRocket. Rather than treating synthetic data as a downstream testing task, the team used GenRocket's JSON Schema and Web APIs to build governed, UI-visible synthetic data projects programmatically — alongside each data model. **The result:** synthetic data now lives inside the engineering lifecycle, not bolted on at the end.

Data as Code synthetic data as an engineering asset	**Full SDLC** Dev → Build → Deploy → Test → Change	**API-driven** governed, UI-visible projects	**Repeatable** Rebuild projects as models evolve

Context & Overview

The client is a leading digital bank standing up its technology estate at pace. As a new digital-first institution, confidence in its data products has to be built in from day one — there is no legacy safety net to fall back on. The estate is large, complex, and changing frequently, with data models evolving alongside the systems they power.

The engagement was delivered in partnership with a GenRocket delivery service provider on the account. GenRocket was already in use inside the organization, but initial adoption had been concentrated in several independent use cases. The opportunity: move beyond point solutions and position synthetic data generation across the full engineering lifecycle — starting with the bank's Data Chapter, the team responsible for the data platform and data products.

The Validation Challenge

Traditional test data preparation didn't fit how this bank builds. The team was hitting the limits of a legacy pattern:

- **Late test data, late confidence:** Waiting until a model or application was largely built before preparing test data created unnecessary friction and delayed validation.
- **A rapidly changing data estate:** Every model change triggered rework in test data setup, slowing iteration and increasing manual overhead.
- **Fragmented use:** GenRocket adoption was already in place but concentrated in a few independent projects — the GenRocket platform's full lifecycle value wasn't being used.
- **Testing treated as a separate stage:** Synthetic data generation was bolted onto the tail end of development rather than embedded in how models were built and evolved.

The team's central question reframed the problem: could synthetic source data be built alongside each data model, so the same controlled inputs support development, QA, and automated validation from the very first commit?

The GenRocket Solution: Data as Code

Working with the GenRocket channel partner, the bank built a Data-as-Code approach on top of GenRocket's Design-Driven Synthetic Data platform. The core idea: treat each synthetic data project as a versioned engineering asset that is created and provisioned programmatically — not set up by using a UI.

How It Works

- **Declarative project definition:** The data model and synthetic-data requirements are combined into a structured JSON Schema definition — a single source of truth that describes what the GenRocket project should contain.
- **Programmatic project provisioning:** GenRocket's Web APIs consume the JSON Schema and build the full set of project assets automatically — Domains, Receivers, Generators, Scenarios, and Configurations.
- **Governed, UI-visible output:** Automation does not create a shadow artifact outside the platform. The result is a fully governed, auditable, UI-visible GenRocket project — indistinguishable from one built in the GUI, but reproducible on demand.
- **Lifecycle-wide reuse:** The same controlled synthetic inputs support Dev, QA, and automated validation across every environment the model touches — no reprovisioning, no drift.
- **Rebuild on model change:** When the underlying data model evolves, the JSON Schema is updated and the project is regenerated. No manual rework, no lost governance.

FROM	TO — with Data as Code on GenRocket
Synthetic data designed after the Model is created	Synthetic data built alongside every data model
Project configuration in the UI	Programmatic provisioning via JSON Schema and Web APIs
Project updates required whenever the data model changes	Regenerate the project from an updated schema
Independent, use-case-specific adoption	Repeatable pattern across the Data Chapter and Enterprise
Testing treated as a separate stage	Synthetic data embedded across Dev, Build, Deploy, Test, Change

Results & Benefits

The Data as Code approach shifted synthetic data from a testing utility to a core engineering capability. Early outcomes include:

- **Synthetic data across the full SDLC:** Available from Dev through Change, evolving alongside the systems it validates.
- **Repeatable infrastructure:** Project assets are created, versioned, deployed, and regenerated programmatically — no manual rebuilds.
- **Governance preserved:** Every automated project remains fully visible, manageable and auditable inside GenRocket's UI.
- **Reduced developer effort:** A structured, schema-driven model cuts repeated discovery and rework — for both engineers and the AI tools assisting them — reducing time and unnecessary token usage.
- **Faster confidence in data products:** Validation happens as models are built, not weeks after — critical for a digital bank building trust from the ground up.
- **Broader GenRocket adoption:** The engagement moved GenRocket from independent use cases to a repeatable pattern the Data Chapter can scale across the enterprise.

Why This Matters

This engagement demonstrates a shift in how mature engineering organizations think about test data. Synthetic data is no longer a downstream QA task — it is an upstream engineering asset, versioned and provisioned like any other piece of infrastructure. For a digital bank operating in a fast-moving, regulated environment, that shift matters:

- **Data products earn confidence earlier:** Because validation is designed in, not bolted on.
- **The estate can change without breaking testing:** Because the schema is the contract, and the project rebuilds from it.
- **Engineering effort compounds:** Because every project is a reusable pattern, not a one-off configuration.

What's Next

With the Data as Code foundation in place, the bank and delivery services partner are extending the pattern into new territory:

- **Deeper API coverage:** Expanding programmatic provisioning to cover more of the GenRocket project surface, driving further repeatability.
- **Enterprise adoption:** Scaling the pattern beyond the Data Chapter into adjacent engineering teams and business domains.
- **Model-driven scale:** Handling larger, more complex data model structures with the same declarative approach.
- **Continuous evolution:** Feeding learnings back into GenRocket's product roadmap — shaping how future customers adopt Design-Driven Synthetic Data at scale.

Conclusion

For this leading digital bank, synthetic data is no longer a testing artifact — it is code. Defined declaratively, provisioned programmatically, governed centrally, and reused across the full engineering lifecycle. In partnership with the GenRocket services delivery partner, GenRocket has moved from a helpful data solution to a foundational platform for how the bank builds and validates its data products.

The result is a modern engineering discipline: repeatable, governed, and built for a data estate that never stops changing. For any organization asking how synthetic data belongs inside — not alongside — the software lifecycle, this engagement is the answer.