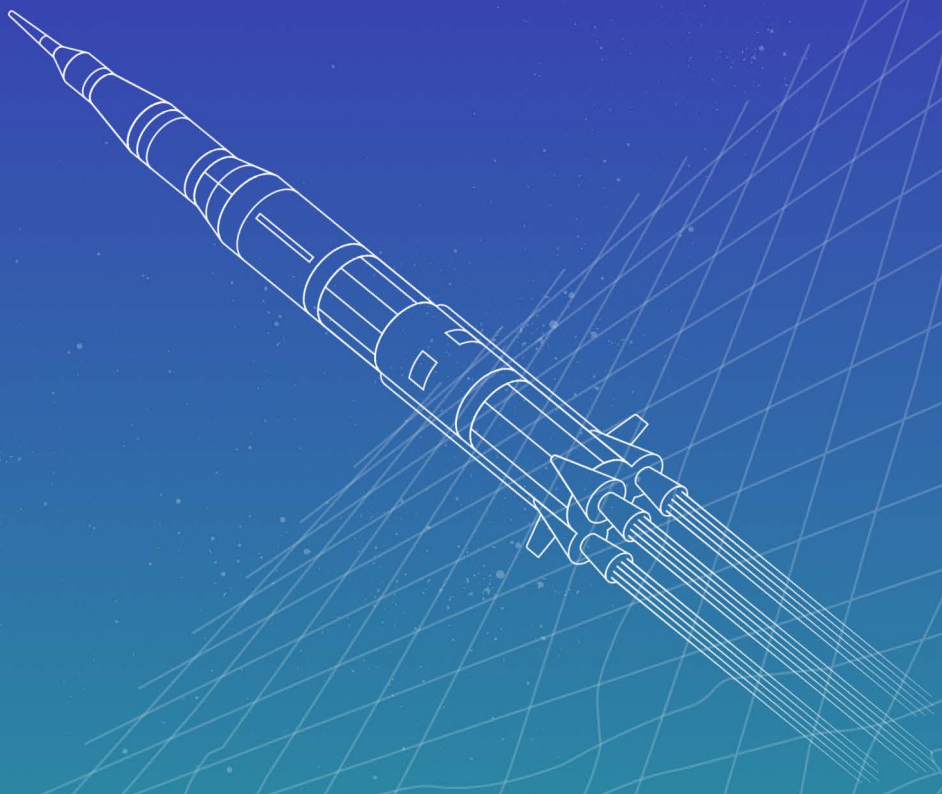
APOLLO

# NerdWallet Builds Better Apps, Faster with a Data Graph



nerdwallet™ | Case Study

# NerdWallet Key Stats

**>140M** Visitors to their tools and content

**27** Major mobile features added last year

**>40%** Year over year company growth

There are few things in life more important than personal financial security, yet there are few things more confusing and complicated than personal finances. That's a challenge that NerdWallet tackles by providing objective advice, expert information, and tools to help people better understand and manage their finances.

With over 140 million visits to their tools and content, NerdWallet is constantly looking for ways to improve the user experience and give more value to their customers. Last year, they shipped 27 major new features to their mobile app alone. Next year they want to move even faster, adding new functionality and increasing speed and reliability of their apps.

To hit these goals, the engineering team launched a code overhaul focused on implementing the Apollo GraphQL stack.


Nerdwallet provides advice, expert info and helpful tools that help people make smart money moves.

### CHALLENGE

Legacy REST infrastructure became increasingly complex, slowing down development and making it difficult to enforce consistency across the user experience.

### SOLUTION

A central data graph, implemented and managed by the Apollo platform.

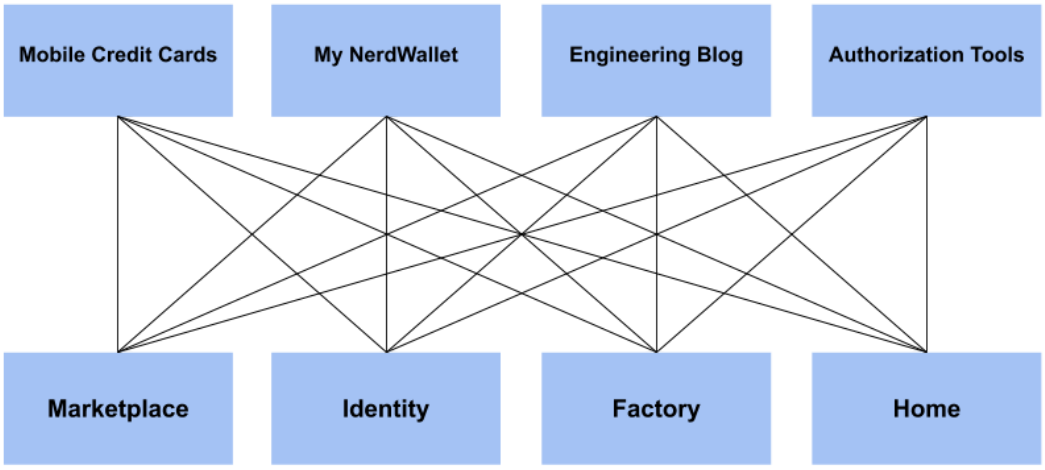
### RESULTS

- Bring new products to market faster and with fewer bugs
- More efficient, faster development
- More reliable user experience when internet speeds slow down
- Apps require less device CPU and memory footprint.

“During the early research of this initiative, it was becoming increasingly obvious that we were not moving fast enough where possible,” said Neil Lokare, Senior Software Engineer at NerdWallet.

“A good example was cookie-based authentication, which was built on complex code where the client was making far too many calls to different services--up to seven! And the infrastructure was inflexible, with back-end API concerns tightly coupled to the presentational concerns of the web pages. It was difficult to make changes to the code and, worst of all, it was error-prone for users.”This infrastructure was also making it difficult to enforce consistency across the user experience. Front end engineers leverage a wide range of services, from global authentication to specialized services such as fetching the latest mortgage rates. These integrations lived in libraries that made dependency management across codebases challenging. This model served the purpose to an extent, but it involved a significant amount of redundancy, middleware, and state management across platforms.

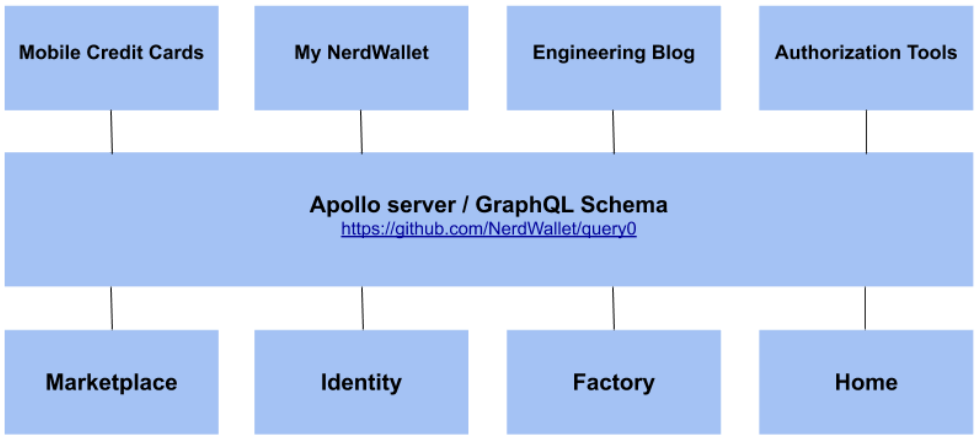
BEFORE ADOPTING A DATA GRAPH: REDUNDANCY AND COMPLEXITY



## TURNING TO A DATA GRAPH

A data graph that overlays existing services infrastructure, and is accessible to clients via GraphQL, was a natural fit for what NerdWallet wanted to achieve. By building a central menu of data across many services, they could anchor how front-end teams and back-end teams worked together to build features.

AFTER ADOPTING A DATA GRAPH: SIMPLIFIED, PERFORMANT INFRASTRUCTURE




As Lokare explained, “a data graph provided a well documented, cross-functional API that could be easily used or extended by teams throughout engineering. And not only would it allow us to work faster, but it would directly improve the app experience— working more reliably when internet speeds slow and taking up less device CPU and memory footprint.”

## PILOTING THE GRAPH

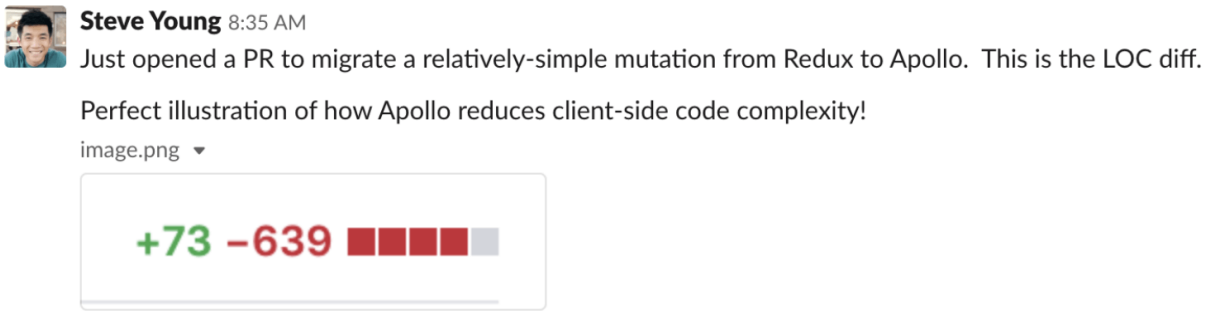
Early in 2019, NerdWallet identified its online shopping and rewards platform codebase as the first use case for a data graph. It also took the opportunity to start phasing out Redux and replacing the existing API integrations with GraphQL.

As a relatively young codebase, leveraging a single backend service written in Python, NerdWallet’s Rewards experience was an ideal pilot candidate, as outlined on the NerdWallet engineering blog.

Apollo’s GraphQL Server and Client were chosen for the implementation. As Lokare explains, “Apollo is the de facto standard GraphQL implementation, but we were also pleasantly

surprised to find built-in support for things like caching, deduplication, and error handling. You just write the code that's specific to interacting with your backend, and Apollo takes care of the rest."

The team saw benefits immediately. "The data graph was easily readable. There were fewer network round trips, far less client code, and fewer bugs," explains Lokare. "Plus, there's no Redux state management for the products that use GraphQL, which means even less code and no special considerations to navigate if someone else jumps in to work on the code."




Now, both web and native mobile codebases reference the same data graph.

## MANAGING THE IMPLEMENTATION

NerdWallet had initially planned for the back end team— the ones most familiar with the services themselves— to own the data graph layer. But as Lokare explains, "The more we recognized that a data graph enables a decoupled, product-first approach, we made the decision to have product engineering teams own this." In accordance with the Principled GraphQL guidance, this helped the team to ensure that it had a demand-oriented schema that was always built against real product needs and evolved with the product.

NerdWallet also chose to use Apollo's platform to manage the data graph layer. "Apollo gives us the tooling to operate the graph across teams without creating and maintaining another internal product," says Lokare, "Graph Manager, especially, allows us to leverage the Apollo Server infrastructure we built and enrich our schema with usage information, errors, and attribution. It's great for making sure we have no breaking changes rolling out to production."

## BUILDING A GRAPHQL CULTURE

Implementation time was a main driving force behind adoption and helped the team ship code expeditiously. Now the next step for NerdWallet is expanding the graph's use and creating a culture around GraphQL.

Sarah R., Senior Software Engineer at NerdWallet, is helping educate the rest of the engineering organization on GraphQL and on how to leverage the data graph.

"Right now, anyone can contribute to the data graph," said Sarah. "As teams hear what was done for the Rewards experience, they're excited to learn how to integrate and start implementing their fetching logic in the Apollo GraphQL layer. Teams are also finding that the graph benefits onboarding of new engineers, where it's much faster to get started and contribute to the code base."

Sarah added that one of the most effective ways to encourage company-wide education on the benefits of the data graph has been the internal working group that NerdWallet formed. The group meets weekly to discuss best practices, share tools, discuss concerns, and brainstorm strategies. As Sarah explains, "GraphQL is succeeding at NerdWallet, because it's a team effort."

Moving forward, there is an expectation for full GraphQL adoption across teams, said Sarah: "Internally, we've worked hard on evangelizing the technology in a way where managers, product managers, developers and QA all understand the value of the graph to their own roles and the needs of the business."

*NERDWALLET INTERNAL TRAINING BY NEIL LOKARE*

As the tangible benefits of the graph stack up, that value is getting easier to explain. For example, said Sarah, “Teams that are moving to the graph are bringing products and services to market sooner and dealing with bugs faster and less invasively. Engineers are stoked!”

“A data graph is so much more streamlined and straightforward. It’s just far easier and faster to use.”

> *“There’s no way we’d go back to the complexity we had before.”*
>
> Sarah R., Senior Software Engineer at NerdWallet