**EPISODE 471**

[INTRODUCTION]

**[0:00:00.0] JM:** Are we a media company or a technology company? Facebook and The New York Times are both asking themselves this question. Facebook originally intended to focus only on building technology to be a neutral arbiter of information and this is turned up to be impossible. The Facebook News Feed is defined by algorithms that are only as neutral as the input data. Even if we could agree on a neutral data set to build a neutral News Feed the algorithms that generate this News Feed are not public so we have no way to vet their neutrality.

Facebook is such a powerful engine for distribution it has allowed for the rise in the number of publishers who can get their voice heard, and as a result of this, large media companies have lost some market share because Facebook has replaced their distribution advantage. The New York Times has always been a media company, but the standards for media consumption have shot up. Millions of people produce content for free, and that content is distributed through high quality experiences like, Twitter, YouTube, Medium, and Facebook. And, when a page takes too long to load on newyorktimes.com it doesn't matter how good the reporting is. The user is going to navigate away before they read anything.

Today, the New York Times has built out an experienced engineering team. In a previous episode we reported how The Times uses Kafka to make its old content more accessible. In today's show, we talk about how The Times uses React and GraphQL to improve the performance and the developer experience of engineers who are building software at the New York Times. Scott Taylor and James Lawrie are software engineers at the New York Times and in this episode they explained how the New York Times looks at technology.

The user experience on New York Times rivals that of a platform company like Facebook and this is, funny enough, assisted by technologies originally built at Facebook; React, Relay, and GraphQL. If you like this episode we have done many other shows about React and GraphQL. In fact back in the day we did an entire week of shows about React. and you can find all of our old episodes by downloading the Software Engineering Daily app for iOS and for Android. In the other podcast players you can only access the most recent 100 episodes but in

 

our app you can download all 600 episodes and you can also get recommendations based on what you've listened too so far and we're working on some other very cool stuff. For example, if you listen to an episode about React, maybe we can recommend you some great content about React.

So, we're building a recommendation system, we're building a new way to consume content about software engineering and these apps are open sourced at github.com/softwareengineeringdaily. If you're looking for an open source project to get involved with, we would love to get your help.

Now, let's get on this episode.

[SPONSOR MESSAGE]

**[0:03:08.3] JM:** G2i is a talent platform built for engineers by engineers. React Native, React, and mobile. The developers on G2i had expertise in the best tools to build your applications. When I need engineers to help me out with my apps, G2i is the first place I go, especially when I'm building with React or React Native. Contract a G2i developer to help you on a short-term basis or hire a G2i developer full-time and if you're looking to build cross-platform applications in React Native, definitely check out G2i.

The G2i platform is a community of React Native, React, and mobile developers, and these engineers can become part of your team. If, you're looking for developers to build your product checkout g2i.co. That's the letter g, the number 2, i, dot co. You can also send me an email and I'll be happy to tell you more about my experience with G2i. Find your React Native, React, and mobile talent by going to g2i.co and thanks to G2i for helping me ship my products and thanks for becoming a sponsor of Software Engineering Daily.

[INTERVIEW]

**[0:04:32.4] JM:** James Lawrie and Scott Taylor are Software Engineers who work at the New York Times. Scott and James, welcome to Software Engineering Daily.

**[0:04:39.6] ST/JL:** Thanks for having us.

**[0:04:41.2] JM:** Today, we are going to talk about a refactoring that the New York Times went through, mostly on the front-end. Let's start with just giving an outline for where we are media-wise and how the New York Times is changing.

It's 2017 there are up and coming media companies like BuzzFeed, there are social media companies like Facebook and Twitter, there are podcasts and YouTube and media is changing fast. But in this maelstrom of changing media the New York Times is holding strong as a durable media institution in fact it's growing aggressively. But how is the New York Times changing, internally? What's changing and what is staying the same?

**[0:05:27.3] ST:** Yes, so what's interesting, and James is going to talk about, you know, some were back-end systems. But actually a lot is changing. Pretty much everything is changing internally. You know, I work in a team called, Web Frameworks. Web Frameworks was kind of a group that was assembled to lead a migration to whatever our new platform was going to be. That team is about six people. You know, I work on a team with the stuff we're going to talk about today with, you know, Jeremy Gayed, he is one of our lead engineers. Olov Sundstöm, and then Matt DeLambo and Carrie Price helped out in different areas for our front-end architecture.

Yes, so the Times traditionally, you know, the Times had a website for a long time now and we've done different migrations over the years but I think this is one of the more massive ones, you know, especially on the front-end. The Times is traditionally been a PHP side actually for a lot of the pages our readers read. So like, you know, the home page and article page and some collections. We're doing a pretty radical shift right now to some other open source technologies like, React and Node.js, and GraphQL, and with that a member of different GraphQL clients we've been looking at like Relay and Apollo.

At the same time there is a lot of change happening, you know, we're moving stuff from data centers to the Cloud, you know, in a lot of instances and then James can tell you a little bit more about how we're actually migrating how we publish data.

**[0:07:05.7] JM:** Yeah, and we actually had a show recently about that with the, I believe the Director of Engineering, Boerge Svingen, he was talking about how Kafka is moving a lot of its heterogeneous data sources to Kafka to unify them, and people can go back and listen to that. James, do you have any perspective on how the New York Times front-end has evolved since the company moved online because I believe it move – started moving online in the 90's and I imagine there have been many evolutions and refactoring since that time. Do you have perspective on that?

**[0:07:41.2] JL:** I do have a bit of perspective, I've been at The Times for about six years now. So, they're currently working on me sixth iteration of the website. When I arrived it was the fourth iteration and they're actually still pages here and there that are driven by third and fourth and fifth iterations of the website. So, I think – I think things are changing more rapidly now.

**[0:08:08.5] JM:** Are there any usability or performance issues that have come because of the engineering decisions of the past? Like are – because this refactoring that you made with GraphQL and Relay and React, what were some of the problems and the performance issues that you were seeking to improve upon?

**[0:08:30.5] ST:** Well, I think there's – there's always concerns around, are we doing enough to, you know, as we add new features to the site and as the code bases evolve, keeping performance up, and also, you know, what's our strategy if we're always maintain accessibility? And, you know, I'm pretty proud of the near times for actually, you know, our last iteration taking a pause at one point and really focusing on accessibility in a lot of different ways and we have –

**[0:08:57.5] JM:** What's that term mean; accessibility?

**[0:09:01.0] ST:** Accessibility means for, I guess, disabled persons or people that need assistance via, you know, maybe a screen reader, you know, for people who are eventually impaired, people who can't use a mouse, you know, just basically meaning that people who may need the use of assistive technology, there is a little bit of work in to do to ensure that you're not just writing free willing HTML it doesn't support the proper attributes we need to make the site accessible. So we have people that regularly like listen to the site with a screen reader or make sure that like as we're tabbing through it makes sense for somebody who's not viewing the

website, just your site. And, I think like what we kind of view – the way we build websites I think in general from company to company and, you know, really in this era there's a lot of focus, you know, on JavaScript to JavaScript frameworks and I think there are some advantages we can get with that.

I think, you know, an ideal scenario is that we end up with what we call a single page app. So, if I'm on the home page I can quickly navigate to an article and I don't have this paradigm of like loading the entire home page than doing a full request to the entire article page and doing full request for the entire something else and there's a wait to that and there's also a way after designed to either fit that paradigm or fit the paradigm with things and more modular and things are a little more elastic in the way they move back and forth.

You know, we're still figuring out a lot of stuff but we really liked — some of the reasons we're replatforming are for that reason. But, another reason is the way we maintain our code internally, the way we write code internally, the way we build user interfaces internally. We're — we've been just really excited about using React and it's kind of already paying dividends for us to be part of that ecosystem and just the way we can move and the way we can modularize things and we're also looking at the way we can share those things. So teams across the company, how do we take components and, you know, what happened before is that because the main website, you know, was in PHP a lot of it. The thing is to The Times it's not one modal with the code base. There's actually, you know, maybe several code basis that deal with different paths on the site.

One of the problems we had is we didn't have a lowest common denominator language. We were all writing these apps in and so it was very hard to share code across projects and I think, you know, around the building just Node.js and React has become pretty popular for a lot of these front-ends. So it's made it easier for us to plan for the future and say, "You know, if we build these components this way with somebody down on the newsroom for a side project wants to use it, you know, in their project, it's actually very easy to share the exact same code."

**[0:11:54.9] JM:** Yeah, she did a show recently with Netflix and Netflix has a team entirely devoted to making it easy for engineers to spin up new front-end clients and this makes sense because Netflix is on so many different surfaces. But you think of The Times, the Times is a

media company as well, theoretically it should be on all of the same surfaces as Netflix. So, you know, it makes sense that The Times would have the same desires to make it easy for any engineer at The Times who wants to spin up a new application, a new front-end to have a buffet of tools and components to reach for and build that new surface.

**[0:12:37.5] ST:** Yeah, definitely I think like that's one of our charges as the web platforms team is – or web frameworks team is that we need to pick solutions that are like portable that will hopefully survive, you know, the next migration and hopefully make it easy that if somebody wants to build things with, you know, wants to build a New York Times shaped thing it's very easy to get started and we actually have an open-source project called, "kyt", k-y-t, which is available from the NY Times GitHub account. It makes it really easy to spin up what we call, "Isomorphic JavaScript apps". So, that's where you have to deal with the server and the client.

I found it so useful, I actually use it in all of my side projects or like hobbyist projects. I think it's one of the best ways to get started with any project with doing isomorphic rendering but that's something that we're building The New York Times on as well and so like as we evolve it for The New York Times we're also open-sourcing that. I think another huge component about this is the data in that the switch to GraphQL is also going to democratize this a little bit more around the company. Because as we know in the past, if you have web service driven apps or websites, what ends up happening is your code base, you need to create an HTTP client and then you need to connect to several different rest services a lot of the time and with GraphQL, you know, we kind of one place to go to to get data. And so, I think like the goal, you know, the Utopia I see in my mind is that this GraphQL becomes the one place if you're going to build a New York Times app, you can possibly go and get all this data from and that's the stuff that James' team is thinking about too, is like really working on what is this API gateway look like for The Times and how does –

**[0:14:30.1] JM:** Yeah. So, before we jump into that, I want to get in to the nitty gritty of GraphQL and Relay and React. But let's just set the table a little bit more. James as the engineering lead of this project and particularly somebody who's been there for six years, give me a description for how this project to refactor the front-end got started and how it ended up being the focus of kind of a bigger team that's, I think, you know, six plus people? Was this a gradual process to start with just one person or was to say, concerted process where, you know,

top down management who was like, “All right, we’re going to do this big refactoring and we’re throwing a big team on it.”

**[00:15:11] JL:** Well, I think it was a kind of a combination of those. So, I started out working with the Native applications and I was involved in several of the projects of getting Native applications going on BlackBerry and on Windows and as, you know, Scott mentioned, you know, you have 15 different REST API's that you have to get those teams to integrate with and it can definitely be painful. So, we actually started our project for the Android and iOS teams. It wasn't GraphQL but it was — ended up with a similar idea where we could take the existing API's and produce like templates of the data that they required for producing their front-end applications.

So, as we are doing this, the web frameworks team was also thinking about what they wanted to do next and Olov and I — Olov Sundstöm and I had several conversations about various demand driven architectures, including GraphQL. We looked at Falcor for Netflix as well.

[SPONSOR MESSAGE]

**[00:16:20] JM:** This episode of Software Engineering Daily is sponsored by Datadog, a Cloud monitoring platform built by engineers for engineers. Datadog enables full stack observability for modern applications. See across all of your servers, your containers, your apps and your services in one place to monitor performance and to make data driven decisions.

Datadog integrates seamlessly to get our metrics and events from more than 200 technologies, including Cloud providers, databases, and we servers. With built in dashboards, algorithmic alerts, and end to end request tracing, Datadog helps teams monitor every layer of their stack in one place.

But don't take our word for it start a free trial today and Datadog will send you a free t-shirt. Go to softwareengineeringdaily.com/datadog to get started and get that free fluffy T-shirt.

[INTERVIEW CONTINUED]

**[00:17:29.0] JM:** For listeners who don't know Falcor — this is so interesting. Falcor was created independently of GraphQL, almost at exactly at the same time. The project looks extremely similar. Both of these projects, as I understand, you guys can correct me if I'm wrong. The functionality is essentially, you set up a GraphQL server or a Falcor server and from the front-end you can make requests that are all formatted at the same way and it looks basically like a JavaScript — sorry like a JSONObject without the, you know, it's like the JSONObject of like what are you requesting.

Like you say, "Here, I want to request a user and the user ID and the user's preferences and the users top four articles," and it just, it looks like a very general query and it goes to a GraphQL server or a Falcor server and that request gets made more specific. It gets processed on the Falcor or the GraphQL server and then it gets federated to all these different data sources that might have REST API's associated with them and what that it gives you this middleware that federates one simple request in to a complex set of request to get disparate data sources and what was so interesting is that Facebook developed technology that does this very specific thing at exact same time that Falcor developed this tech — or, I'm sorry; that Netflix developed this technology and called it Falcor.

And it is just funny that like the New York Times had the same problem. You had the exact same problem as Facebook and Netflix. We have a bunch of disparate legacy and new data sources — maybe it's Bigtable, maybe it's a S3, maybe it's a random API out there like Stripe or something, and you need to get data from all of these different sources. You want to get them in one single query. Am I describing the problem statement correctly?

**[00:19:22] JL:** Yeah, that's pretty much I think.

**[00:19:23] JM:** Yeah.

**[00:19:23] JL:** There was even — there's another group, Datomic, that was also looking at this problem. So yeah, it all kind of happen at the same time.

**[00:19:32] JM:** Yeah. Why is that so troublesome to have to request data from so many heterogenous data sources?

**[00:19:40] JL:** Well, I think at the times one of the things that we've struggled with that sort of like a, sort of having conventions around API's. So, part of the problem was that, you know, a certain group that was producing API's would have a certain type of response format and another group would have a different format and a different authentication mechanism. So, it was even more important for us, I think, to figure out a way to make that all sane for someone who wants to get data from multiple sources.

**[00:20:12] JM:** Yeah. James, when I land on a page on The New York Times and there is a GraphQL query somewhere in that page request and page loading process. Where is that GraphQL query, where is it executing in the process of page request and the page loading?

**[00:20:30] JL:** So Scott, maybe you'll answer that better than I? But I know that so there will be a request from the server where they reproduce the original HTMLs. So that would make a GraphQL request and then the client may make several requests. Certainly initially one to fill out perhaps the below the full part of the page. Another one potentially to fill in sort of user specific data, whether it's username or let's say items and they're reading list or that sort of thing.

**[00:20:59] JM:** Scott, do you want to provide more color on that?

**[00:21:00] ST:** Sure. This is where that concept of isomorphism comes into play, is that there is actually different data requested. Not totally different, but it's the different strategy on the server and the client and the reason is that we actually want to do a lot of caching on the server to where we can render a server response, let's say for the home page or for an article, and we can actually have that serve by Fastly our CDN.

So because of that, we can't have user specific data on the server because we're not going to be passing cookies to a server on every request. We're actually going to hopefully serve most of that from the cache. So in the client, what happens is that there's different mechanisms for doing this within the client's themselves, whether it's Relay or Apollo. But we do have sort of "client only" data. So when you go to it in your browser then we're going to make what are essentially AJAX calls or Fetch calls. That's all kind of abstracted away into these frameworks

for more data. So an actual user and there is a specific data that you're logged in. You know, there is not a ton of user specific data right now but definitely around whether you're subscribed, you know, how many articles you've read, what stuff you may have saved to your reading list, stuff like that. That's the stuff that is specific two a user.

**[00:22:23] JM:** So, we're talking about GraphQL specifically right now, but there are other technologies that from the Facebook stack that you adopted. There is a Relay and React of course. Explain how these technologies fit together, Scott?

**[00:22:39] ST:** So what's nice, you know, React is one of those things that you could just lace into your app in a small way. Like you could have a PHP app that's using React kind of in the corner. But, you know, we're using React really as our rendering engine for the site. With that there's a couple different ways you can associate GraphQL data. The two we've looked at really are we going Relay and Apollo and Relay sort of decorates your React components to where if I have a component for an image and I know that I need the URL for it the width and height and perhaps some other crops if I want to do maybe responsive image rendering or something, I can specify in this decorator really just — and it's kind of like this very specific GraphQL notation. So it looks like a GraphQL what we call fragments and, you know, GraphQL queries are really like queries and fragments and these thing called mutations.

We deal mostly with queries and fragments, and fragments are what get associated with a specific React component. But the idea is that by co-locating your React component with the data you're requesting is that you never over fetching the amount of data you need. And then as your app evolves, you can be sure you're not breaking it. So the way these queries get assembled by this frameworks is that, you know, if I have a fragment for a React component that's including other Reach components then I'm going to include like a reference in this fragment within the component I am currently in.

So, the idea is that — I think one of the problems that happens with REST API's over time and over the years is that we never know when we can remove fields and we never know who's using them. I mean there are ways to find out perhaps, but it's hard. Another problem we ran into is that with that REST API's, you know, the economical representation of an article is actually a lot of data and the economical representation of what might be an image or video.

Because image may have 25 different crops associated with it. And, with REST you're always getting all the data a lot of the times. It's really hard to filter down to just the data that you actually need. And so, we would actually have some rest APR responses that were close to one megabyte. Especially if you're trying to request it on the client. That's a pretty big deal from making a user download data, especially if it's on their phone.

So, with GraphQL and with these React components, you know, as you are assembling your React sort of tree like building blocks, these frameworks are also assembling what a total query would look like as you combine components you are using and you may end up with a query that gets a lot of data but the data is actually like 15% of the size of your actual full response from REST. So that is something we really like. Apollo is very similar, but Apollo doesn't have as strict requirements around where you specify data and they allow you to actually separate and specify your data in different files.

**[00:25:53] JM:** What is Apollo? I know it's a technology that is developed at Meteor and I always hear it associated with the GraphQL conversation. What does Apollo do?

**[00:26:02] ST:** So, let me just give a quick overview of like where the state of kind of Relay and Apollo and actually last week was GraphQL Summit in San Francisco. I actually gave one keynote talks there and Facebook developed Relay internally but it was actually part of the Routing framework and by Routing we talk about single page apps, it means kind of how you declaratively say, "Here's my app and then within it are these routes, the routes may nest from there." But what happens is if you click a link you can kind of go from page to page and only the parts that change in the middle will change, basically.

What Facebook did with Relay is they created a way to have routes and also specify new GraphQL queries on those routes. Facebook obviously learned a lot over time and also with scaling their React footprint and scaling their GraphQL footprint, especially on things like React Native and their mobile apps. They realized they need a lot of changes to make this scale for the future and be fast. So, Relay became what they call now Relay Classic and Facebook released Relay Modern. Relay Modern is kind of the next generation version of Relay and we evaluated it a lot over the summer, but there are some key components missing from Relay Modern that is going to make for us impossible to upgrade to it right now.

So, our number one requirement is isomorphism. Meaning I need to be able to render on the server and the client. That doesn't come out of the box with Relay, Relay Modern. With Relay Classic there are some pieces in the ecosystem that fill those holes, but they don't exist yet for Relay Modern. So, we are actually looking at moving to Apollo for our GraphQL client. And Apollo, which is associated with Meteor but it is still a open source project. Apollo does have all these ecosystem pieces filled out.

There are some differences between the way Relay and Apollo work, but all the developers that work on this kind of full time are in contacting with the Facebook people a lot and they're very active in the GraphQL spec and the GraphQL space. They are all kind of, you know, in the same area. So what was kind of strange is that I assumed because Facebook, you know, conceived of GraphQL and conceived of Relay that they were by default the frameworks you were supposed to be using. When I was at GraphQL Summit that was not the case. Apollo, it feels to me and it felt like this to a lot of people that were there, that Apollo is kind of like the winner right now in the framework race.

**[00:28:44] JM:** Well, so let's help disambiguate for people what exactly a GraphQL client does. Because we've explained, and by the way we've done some other shows about GraphQL, so maybe listeners want to go check those episodes out. But for the sake of the listeners of this episode, we've talked about how I land on a page and, you know, if we are on the New York Times five years ago, if we went to thenewyorktimes.com, my page fires off requests to several different RESTful resources. It gets those RESTful resources back and has, you know, the client itself, your client device, your poor iPhone has to figure out which of these resources it actually needs. It has to filter through all these, you know, giant RESTful requests that it made.

The advantage of a GraphQL server is you make this one query to the GraphQL server and the GraphQL server federates the requests to the different RESTful resources and then it parses through, "Okay, what resources do I actually need to return to the client?" You give it back to the iPhone and this is much less burden on the iPhone itself. You've off-loaded a lot of the demand for request processing and filtering to that GraphQL server. Maybe I've glossed over some things but -

**[00:30:01] ST:** That's mostly right. I think like the GraphQL client, I think the easiest way to think about it, especially in the React space, is that it is what binds the data to your Reach components. But GraphQLis a little more nebulous than that because you can make GraphQL queries outside the framework. You can make them only on the server. I think for the sake of Relay and Apollo, they're definitely are meant to be front-end frameworks for hooking up your UI to your data.

Well because for instance the PHP version of the site, a lot of that data fetching happened on the server and then we had our client JS was kind of a bunch of different backbone modules that also requested data via AJAX but it was very free willing and it was hard to know like what they are getting where and where we are getting it from because you might hit a user service, you might request for the navigation, you might make a request for infinite scroll for what we call collections and those API's may have been all different generated in different places. Some need off, some don't. Some need cookies, it was hard to tell.

What happens now is that because GraphQL is kind of one of the clearing house for data, yeah we may make GraphQL queries in several places in the UI or in response to user actions, but I feel like it's a much tight integration with the data. You actually have to think way less about the specifics how you are getting the data, the specifics about what's resolving the data. It's really this one query language when you need a data for UI and the rest is somewhat magical.

But the other piece that I wanted to say about GraphQL Summit is, and James can talk a little bit about this is, you know, Facebook open sourced a reference implementation like a Node implementation for GraphQL server but the trend I saw at GraphQL Summit was actually, there was a lot of big companies there. You know, Twitter, Twitch, Coursera, Facebook obviously and like people are building GraphQL servers. There's a lot of open source projects in multiple languages. So if you use Python it's Graphene. If you use Java, there's GraphQL Java and then the implementation we use is Sangria, it's written in Scala.

I was actually surprised that Scala — the Scala implementation seemed like the most popular among the presenters and the attendees at the GraphQL Summit, and so James can maybe —

**[00:32:26] JM:** That's random.

**[00:32:28] ST:** Yeah. So James can maybe give you some background about why they picked that. But it was weird though because it has been — a year ago at GraphQL Summit, I wasn't there but I talked to a lot of attendees about it, when they asked who was using GraphQL, not a lot of people raised their hands. There was a lot of like interest in it and a lot of curiosity and now it has become very mainstream. Twitter is talking about “here's how we use GraphQL”, not if. You know, there's a lot of different — you know IBM, there's a lot of different companies that are like, "Yes, we're definitely using GraphQL. This makes so much sense and here's how we are doing it.”

**[00:32:58] JM:** Yeah, James so give us a little bit more color on that. So that was — sorry, you were saying that was the reference implementation? Facebook released a reference implementation for the GraphQL server?

**[00:33:07] ST:** Yeah, in node. Because there is two pieces. I mean, like even if you are running a Node.js app you are going to have two servers. One is your GraphQL server and one is your, you know, Express App that's running your application.

**[00:33:19] JM:** Right.

**[00:33:19] ST:** But at The Times, we have, you know, we have like 300 engineers or so and so we're going to have big teams and like big problems and so James is leading the data side of things and they've been implementing our GraphQL stuff.

**[00:33:35] JL:** Yeah, so back to that decision point when we're looking at Falcor versus GraphQL. I think the big appealing thing about GraphQL is that it was a spec. Yes there was these reference of implementation but they were also lots of other implementations out there, including Sangria. And our project that was the previous iteration before GraphQL, it was already written in Scala and it was using Twitter's Finagle framework, which I think fits in very well with this sort of federated data model and it was actually pretty easy to just put Sangria in the middle of all that.

**[00:34:11] JM:** So, what about the Scala implementation of the GraphQL server made it so popular? Why was Scala a good fit?

**[00:34:22] JL:** I'm not sure that specifically. I think, at least from my perspective, one of the advantages is that Sangria allows you to construct your GraphQL schema in many different ways. So there's this sort of simple standard model of creating it via code. But there's also ways where you can take a static schema files and then materialize your schema at run time. You could have a schema per client if you so desire. It makes it very, very flexible.

**[00:35:02] ST:** I think too, Sangria being an open source project, a lot of the vibes I got at GraphQL Summit were just that [Oleg], the guy who leads the project, has built a lot of features for it. It's very active. A lot of people, we're just kind of very appreciative of like what he has already done with them. There's also this notion, I don't know how much you've heard of this, of schema stitching. So, this is kind of new, in the past six months or so, mainstream idea of like if I'm in IBM, you know, my back-end services don't probably looks like an API gateway but then there is one GraphQL server. Or this GraphQL server actually federating schema's from across the company, and what does that look like, and how does that implementation work? So for the New York Times —

**[00:35:46] JM:** So meaning that there would be many people in the company who would express a GraphQL schema as the way to fetch data from their API?

**[00:35:55] ST:** Yeah, so the schema really, it comes out of this tight definitions and they use resolvers. So if I want to get a recipe, how do I resolve the recipe from The New York Times, you know, data universe, and how does the recipe fit into an article about cooking that might want to include a recipe? So, you know, James' team works with what we call publishing pipeline, which is pumping out The New York Times as we know it. Which is articles and, you know, images and videos and you just kind of like, hundreds a day and it's like huge over time and how we deal with you know that system?

And there are a lot of kind of side projects to where we have a games team. What if the games team wants to become part of this big, you know, kind of GraphQL clearing house for data? You know, it's that something that James is going to do or is that something that the games team

can probably do? But then James team can provide a way to stitch that data in to our main schema and, I think Sangria was actually hinting at this for awhile about how do we take different GraphQL schema put in together? But people in Apollo have released this concept of schema stitching to where you actually multiple remote schemas and your GraphQL server can kind of be a proxy to all of them and there is this kind of configuration to link certain parts together.

GraphQL Summit, one of the guys from IBM released his project called Gramps, you know, like G-R-A-M-P-S. It does kind of similar thing. You know, you can pull in multiple schemas and it acts as kind of like this gateway to the schemas. So we haven't figured it out yet but I was — this is the kind of thing what conferences do for you is like, this sounded avant garde to us, but when we are the conference a lot of these different companies were talking about it like, "Yes, we are going to figure this out. This is how we want to do things."

[SPONSOR MESSAGE]

**[00:37:58] JM:** VividCortex is the best way to improve your database performance, efficiency and uptime. It's a Cloud hosted monitoring platform that eliminates your most critical visibility gap, providing insights at one second granularity into production database workload and query performance. It measures the execution and the resource consumption of every statement and transaction so you can proactively fix future database issues before they impact customers.

To learn more visit vividcortex.com/sedaily and find out why companies like GitHub, Digital Ocean and Yelp all use VividCortex to see deeper in to their database performance. Learn more at vividcortex.com/sedaily and get started today with Vivid Cortex.

[INTERVIEW CONTINUED]

**[00:39:04] JM:** So, I want to get people a top down view into this as we are touching a lot of disparate ideas and some people might be a little bit confused. So, you're describing a world in which at The New York Times there's a bunch of different teams. There's maybe a team working on recipes and cooking information. There's a team working on games. There's a team that is working on old, old articles, maybe the back catalogue of The New York Times and you

want to make it easy for a front-facing client to easily request data from all of these sources and pull it together in a web page, a single web page.

Maybe you've got a web page with a game and a recipe and you know something about the JFK assassination. So it's got to delve into the back catalogue. You've got to make all these different disparate data sources happy. Well, the front-end request for all these data sources happy, and so you want each of those disparate data sources, the recipes team and the game team in the back catalogue team, you want all of them to present a schema that decentralized GraphQL server can access. Is that - Am I painting the picture correctly? The schema idea?

**[00:40:16] ST:** That sounds good to me. I think like a more concrete example might be something like real estate. Some of these things aren't REST API's, some of these things might be protobufs, some of these things might be RPC calls, some of these things might be flat files, we don't know. But, the idea is that if I'm a New York Times developer, wouldn't it be nice if can sit down and start hacking on an app that just uses the New York Times wealth of data sources?

So, like why can't I when I'm creating an article and I happen to know something about the location of the article. Let's say it's in a certain neighborhood or it's in San Francisco, why can't I also show you real estate data in that same response, right? Right now, that's the stuff that's hard when you have disparate services. Because, how do I get the real estate data? Do I have to write an HTTP client for it? Is it JSONP on the client? Is there authentication? And a lot of times what happens is that there's different authentication for every service and so how do I manage that? How do I manage that on the client? It's just — there's a lot of different pieces to consider.

So GraphQL is kind of this magical thing. The devil is in the details and that's the stuff that James' team is working on is, you know, some of this data is kind of one to one resolution where you may have a flat structure of, you know, name value pairs. Those are pretty easy just to return. But what is it look like when one of those fields needs to resolve a different service that's not REST, that's something else? It's like, that's the magic of GraphQL kind of comes in.

**[00:41:50] ST:** So James, when you're talking to the recipes team or the real estate team or the back catalogue team, is the onus on them to present a way for the centralized GraphQL server to be able to access information from all of these different back-end data sources?

**[00:42:15] JL:** I think it's a process and we certainly not at the point where we are stitching several GraphQL API's together at this point. But, yeah, it's a conversation between the producer of the data and the client to figure out what exactly that schema should look like and how that, you know, schema fits into the already existing schema that we have. If you take the, like the back catalogue example, so these are articles that we're talking about so we can just probably just use the same article schema that we already developed for articles that are currently being produced on the site.

**[00:42:50] JM:** So Scott, you were talking earlier about the fact that you can't go to Relay Modern yet. So there's these different versions of Relay. And Relay, again, is what's going to bind the response from the GraphQL server. So you have this GraphQL server that goes out and federates request to all the different data sources, pulls them all together, returns it to the client and is going to bind that data your React components on the front-end and there's a Relay Classic and Relay Modern. Relay Classic did the query parsing logic at runtime. Relay Modern puts the query parsing logic at build time.

What's the different between those and why The New York Times unable to take advantage of Relay Modern? I think actually you said you don't use either of these. You actually use Apollo.

**[00:43:43] ST:** Well, that's interesting I think you are referring to like my Medium post that I made in the summer.

**[00:43:46] JM:** Yes.

**[00:43:47] ST:** That was a lot about this kind of like restating our mission. Just because, what are we really trying to accomplish with this replatform? At the time Relay Modern either hadn't been released or was on the horizon and we are going to try to move forward with it when we found the time and I ended up doing a lot of side investigation. Like I wrote some projects using Wordpress data instead of using The New York Times. Because one of the challenging

things is that especially as React app we may have 400, 500 React components that have all these GraphQL decorations and so if somebody needs to change across the board, it's very hard to try out the frameworks. So we'll typically try them out on a smaller scale on kind of an example project or side project.

And, the thing about stuff that comes out of Facebook is that Facebook has a very specific, you know, engineering system to where adding build time steps is not really a problem. So like, Facebook's code has to be transpiled in ten different ways, you know, everything you do. So, there is kind of culture around these build steps and that's not a huge problem for us but if we are going to introduce this build step it does add some complexity to our project. But the real reason that we couldn't move forward with Relay Modern is just the ecosystem around it, and I will explain a little bit.

So, when you do routing, which means going from page to page in a React app, there are only a handful of choices that people tend to go to. One of those is React Router and to make React Router work with Relay there's a library called React Router Relay. But then to make it work on the server and server rendering means like when I go to the page I see the whole page and then, you know, I made the client may come in and actually React may do a re-render on a client as well, but you won't notice.

A client only app typically means that you come to the site and there is a bunch of loading graphics that makes you think there's a lot of loading happening and there is because we're making the request for the data. So, if we don't have a server-render if you were to view the source of the web page, there wouldn't be a web page there and so we do need that and want that. So with the Relay Modern, a lot of the libraries that were needed to make server-rendering possible in Relay Classic just don't exist and those projects don't have any plans to upgrade to Relay Modern. Relay Modern is wildly different than Relay Classic. Relay Classic did a lot of magic in the heat of things. So like, as — when we say run time we're meaning as their app is running it's doing a lot of this parsing logic.

The problem with that, that Facebook discovered is that over time as your apps get huge that costs increase as well for the run time parsing and they realized that a lot of stuff that we are doing is statically knowable at build time. So, you can actually do that parsing ahead of time and

your app can run and already have that parsing done when you're making your queries or parsing your fragments. So that idea still exists and it's great but so, the problem was that the tools to do isomorphic rendering there was only one and that project was very small and only had one maintainer and it becomes the kind of thing where, do we want to pick a project that's for The New York Times for the future that has that little support especially the community around it? Because we want to rely around open source when we can.

And so, when we look at Apollo we realized we can actually get the same thing from Apollo. However we have a lot more documentation, a lot more community effort around it. It's really become the defacto open source library and I thought when I went to GraphQL Summit it might have been mixed like half Relay, half Apollo and when I gave my talk I was actually one of the only people that talked deeply about Relay. A lot of people didn't mention it. It was just the Apollo became the defacto standard.

So, we realized that going from Relay Classic to Apollo wasn't actually that huge of a lift and then the Apollo has a lot of plug and play pieces to where you have a lot more options for how you do things or how you customize things and lastly, like if I had somebody on my team and I really want them to get some Apollo knowledge I can point them at a lot of documentation. Relay has some documentation but it's extremely — it's elegant framework but it's also extremely complicated. And so, I think we were kind of afraid that a small number of people in the building would really understand Relay and they become single points of failure for this information and for this architecture.

**[00:48:35] JM:** Yeah, very interesting. Okay, let's take a step back. I want to understand more about how engineering works at The New York Times as, I know were up against time but like I said we did the show with Boerge Svingen about Kafka at The New York Times and that gave some insight in to how the legacy of different data sources at The New York Times created some different engineering challenges. But it created a hot bed of interesting engineering solutions as well.

James, since you've been in The New York Times for a while, can you talk about the interaction between the back-end and the front-end engineering teams or just the different engineering teams that — the different technical teams at The New York Times?

**[00:49:19] JL:** Sure, I think for a long time The New York Times there was definitely silos that popped up, fiefdoms. You know, people were developing a certain way and they wouldn't really talk to other teams that were doing similar things and the whole time that I've been there that has been a big challenge and hard to get communication happening throughout the entire engineering team.

I think we are in a much better place now than we were. We have a whole RFC process where every team is expected to share with everyone else changes, new architecture that they're introducing. We still don't have, I would say, you know, standards around languages to write in or frameworks to use. But we're slowly getting there.

**[00:50:12] JM:** And, what's been your perspective on interaction between engineering and the journalism team?

**[00:50:17] JL:** So, the teams that I've been involved with have largely not been interacting with the journalism. We have a specific group that does interactive news and I know they are very much embedded with the journalists and producing these sort of presentation of the stories alongside the people who are writing the stories.

**[00:50:40] ST:** It was about four years ago the times that this innovation report too, if people don't remember that. I think like we've been slowly moving closer to the journalists in a lot of ways and the journalists have moved — are moving faster kind of like away from print and more towards the web first, which sounds like strange to us because, you know, we're all engineers and we've lived in the web for a long time. But I think journalism, especially in The New York Times, I mean used to mainly be a printed paper that people got on their doorstep and a lot of the revenue came from print advertising and so this huge paradigm shift.

There's a lot of people who, you know, may have worked at The Times 30, 40 years and then all of a sudden here we come with the React and GraphQL and stuff and they're like, "What's going on?" You know, so I think there's definitely always been a newsroom presence with technology. Like we have a graphics team that's insanely talented and a lot of cool projects have come out of the graphics team. Like, you know, Jeremy Ashkenas was a member for a long time and you

know Underscore, Backbone and CoffeeScript were written by him and perhaps others. Michael Bostack who is the author of d3, was on the graphics team and Rich Harris is there now who's the author of the Svelte framework and a lot of those developers are journalists/ developers, which is neat. So a lot of our graphics — our graphics team, which does produce a lot of these interactive pieces, they're also the journalists on a lot of these stories.

Yeah, I think there definitely was a divide for a long time, especially with technology around the business and then the newsroom and I think that those lines are getting blurred a lot now to where we're all kind of one ecosystem again.

**[00:52:26] JM:** Yeah, that's certainly what I'm seeing. Okay, given your vantage point at The New York Times, I want each of you to tell me something that I might not know about where media is going or what you're seeing at The New York Times that, you know, might surprise me. You know, as somebody else who is in media but you know I'm at a much smaller media company, Software Engineering Daily.

What's something that might surprise me about how things work at a large media organization that will give some perspective on where the media landscape is going?

**[00:53:01] ST:** It's always changing. I think that, you know, before I came to The Times, I came to The Times, you know, close to five years ago. I thought the Times was going to be this gold standard for technology and was going to be on this level that I had never seen and a lot of ways The Times is just a traditional big company with the same problems every other company has.

But, you know, what's inspiring me is that, you know, our technology team now has an addition like a video team, there's an audio team, there's a 360 video team, there is a VR team. What can be exciting about that is that as we're looking at these technologies like React, it's like how does React Native fit into these things? And like, how do we actually work across these teams to create like new journalistic experiences?

I think right now too, there's kind of this arms race that I'm seeing even from as an outside perspective sometimes is, you know, companies like the Washington Post and The New York

Times, which are becoming so crucial in this political environment and just kind of like this new universe that we're living in. You know, there's a lot of enthusiasm for these institutions and there's a lot of support. You know, our subscribers support us monetarily and it's like it gives us the freedom to experiment more and, you know, kind of delight our readers more and I think that like one of our main focuses as a company is how do we really engage this subscribers and give them the experience that's worth paying for? And so because of that, I think we are going to see a lot of experimentation here, the Washington Post, The Los Angeles Times, elsewhere.

So, I mean for me personally, it's actually a very exciting place to be. But, you know, when you go to these conferences we're all solving the same problems, we're all trying to figure out what this new universe is and with software, you know, React has kind of been the hot thing for a couple years now. Like it's — we're all these big companies that are trying to figure out, what is it mean to be an enterprise grade React app and how does that scale with software and how does that scale organizationally? And I think this GraphQL piece is a big part of that too.

**[00:55:09] JM:** James, anything you want to add?

**[00:55:10] JL:** I just want to add that alongside the new types of ways of telling stories, so the consistent thread for us from a data perspective is how do we capture that in a way that will make sense in the future? I mean we have 150 years worth of archives and what is that going to look like in another 50, 100 years?

**[00:55:36] JM:** Definitely, well that is one of the challenges that Boerge was talking about in my interview with him. So, it sounds like a great place to close. Well, James and Scott, thank you for coming on Software Engineering Daily. It's been great talking to you about GraphQL and, you know, I've been enjoying The Times' long form journalism for a long time. So thank you for putting your leg work in to making that institution tick.

**[00:55:58] ST:** Yeah, thank you so much for having us.

**[00:56:00] JL:** Thank you.

[SPONSOR MESSAGE]

**[00:56:04] JM:** Are you a Java developer, a full stack engineer, a product manager, or a data analyst? If so maybe you be a good fit at TransferWise. TransferWise makes it cheaper and easier to send money to other countries. It's a simple mission, but since it's about saving people their hard earned money, it's important. TransferWise is looking engineers to join their team. Check out transferwise.com/jobs to see their openings. We reported on TransferWise in past episodes and I love the company because they make international payments more efficient.

Last year, TransferWise's VP of engineering, Harsh Sinha, came on Software Engineering Daily to discuss how TransferWise works and it was a fascinating discussion. Every month customers send about $1 billion dollars in 45 currencies to 64 countries on TransferWise and along the way there are many engineering challenges. So there's plenty of opportunity for engineers to make their mark. TransferWise is built by self-sufficient autonomous teams and each team picks the problems that they want to solve. There's no micro management, no one telling you what to do.

You can find an autonomous challenging rewarding job by going to transferwise.com/jobs. TransferWise has several open roles in engineering and has offices in London, New York, Tampa, Tallinn, Cherkasy, Budapest, and Singapore, among other places. Find out more at transferwise.com/jobs. Thanks to TransferWise for being a new sponsor of software engineering daily and you can check it out by going to transferwise.com/jobs.

[END]