Get unlimited access to the best of Medium for less than $1/week. Become a member

Multi-Tenant Strategy for SaaS using MySQL5

Tu Chu · Follow
9 min read · Oct 23, 2018

Listen Share More

Source: https://think360studio.com/saas-application-development-service/

> *This is a re-post from ReachCRM blog , I found it when I was spending 3 months research on a "perfect" solution on building SaaS applications. Such a good read IMHO.*
>
> *Sadly the post was gone so I re-post it on my Medium for record. All credits are for author* ***ReachCRM***

I've recently had the greatest fun exploring options for implementing multi-tenant architectures for one of my upcoming projects. If you know anything about me, you

know I'm a data guy. So it's all about the fundamental database schema. This is a walk through the thought process I went through, so you can either come to the same conclusions, or you can disagree and perhaps point out some alternative solutions.

So let's begin! Rather than just give the final conclusion — which turns out to be

Open in app ↗

Search Medium

mingled that you can have.

This time, I thought I'd take a step back, do some research on other folks approaches, and then decide my strategy. It's quite surprising how little has been written about this topic. I hit some IBM approaches that depended on some unique DB2 things; I found some scholastic readings that opined it wasn't easy to do; I found the usual separate database per tenant, or separate schema per tenant with shared database discussions; I also wondered if by choosing MySQL as my implementation platform I was boxing myself in corner. Spoiler alert: it all ends happily ever after!

What not to do and why

First I discounted using shared schema and different databases. Too much management overhead, and instinctively, I wanted to eliminate the need to have multiple database instances. Having to do a separate database instance for each customer just sounds hideous when it comes to upgrade time.

And although some database vendors can create multiple schemas in a shared database, again, I'd be looking at having to maintain multiple definitions of the schema mapping into the database instance. If I had half a dozen customers… maybe it would fly. But for scaling to hundreds or thousands of customers, forget it.

Which means, everyone has to have co-mingled records in the one database, in the one schema. This has benefits of easing maintenance. But it also raises the big question of how do you ensure that one client can *never* see another clients' records?

How to separate each client's data when it is commingled

The first obvious step is that in every table, you'd have to create a "client identifier" of some sort. Let's call it a "Tenant ID". Excellent! Now all your client applications code need to ensure that whatever other querying they are doing, they must also remember to include a Tenant ID = some token value. Let's assume that when

Joanne Schmoe signed in, you can return the Client Account that Joanne belongs to, and that the account data contains the Tenant ID for this customer. So stash that somewhere and your application should be able to augment the queries and limit the records being returned to just that Tenant ID (client information).

Let's back up just a mo… recall that the application — typically the middle tier — is accessing the database using a super-user login that the front end knows nothing about. Which makes sense. You don't want to have a new database user account for every single 'real world' user accessing your application. Can you imagine how many user accounts Salesforce.com would have?

We also know that our real world user logs in to the web tier, is authenticated as being a member of a valid and current client account. So we know the application username, and we know the client account.

Our end goal is to somehow turn that information into a Tenant ID that can be automatically included in *every* SQL Insert statement in *every* table in the database. Leaving it up to the application code in the middle or web tier, is really not very satisfactory. It's certainly possible, that the user logs in, determine the client account, get the Tenant ID, stash it in the session somewhere, and make sure it gets included in all inserts. This means that every application has to get it explicitly correct every time. Do I trust developers to remember to do this, and then remember to test it, every time? No. It's too weak a link. I'm sure I could construct some object classes that wrap the insert, but I still have to remember to use those every time, and the code is now starting to get complex. So I sought a different and simpler approach.

SQL Injection issues

Another problem one has to guard against is that if you are using simple SQL Where clauses to select data and filter on the Tenant ID, you are again at the mercy of developers and testers to ensure it all works exactly as designed. Further, if your application interface allows users to specify queries, which your app formulates into SQL select statements, there is always the potential risk of some query injection that effectively makes the SQL Select return the complete record set overriding the Tenant ID where clause. How? You may be querying on a description field perhaps and fully expecting a text string which may not be in any standard format. And it may legitimately contain symbols such as an equals sign. All you need to do is insert the query value as " or where 0 = 0 " or any other legal expression that will ensure

the SQL Where clause will always evaluate to True, and a full record set will come back as the OR overrides the Tenant ID where clause. And for sure, you could write some code around all this to combat it, but that is what I would classify as non-productive code. It doesn't move the application functionality forward, it complicates the development, and it is still potentially imperfect. And in a multi-tenant implementation, anything short of a cast-iron guarantee of data privacy is a ticket to losing customers' confidence.

So what about implementing a user based security model? My first rough assessment of this option was that it wouldn't scale in the most general sense. I may have several thousand customer accounts and each customer with several hundred users. It just felt like it would be unworkable in the long run.

Update-able views to the Rescue

The key MySQL5 ingredient that suddenly did seem to offer a road forward was noting that views that are based on a single base table are fully update-able. The view has to have the primary key and conform to a few other minor rules, but essentially it is quite viable. Aha! So … create a base table with all required application columns, and add in one more column, the Tenant ID. The Tenant ID should *not* be any part of the primary key. Depending on your application, you decide if it needs indexing or not.

Now create a view which includes all your base table columns *excluding* the Tenant ID. Excellent! Now to tackle the problems with this. First you need a where clause in the view based on the Tenant ID. What should you declare against? There are very few system variables that you can use at runtime, and the best candidate is the userid. So if we make an assumption that the Tenant ID is logically equivalent to the userid, that would mean that to access the database, the application userid in the connection string must be the Tenant ID value. To do that, when we get a new client to our SaaS app, we'd have to create a single new user whose userid was the tenant ID — almost certainly not visible or known to the actual user — and associate that userid with the client account information. That way when a user logs in, authenticates in the login process (via the usual super-user db connection behind the scenes), we would do a lookup process, determine their Tenant ID userid, and now the application would use that as the db connection string userid for the duration of the session. What this achieves is that the select statements (and updates, inserts and deletes) are *guaranteed* to only retrieve (or update) record sets for that tenant ID. And we've limited the tenant ID being used in the application to

only one place, the authentication/login process. And even in you try to inject bizarre 0 = 0 where clauses into the view, it cannot retrieve records outside of its declared constraint. Try it. It's bullet proof.

How to populate the Tenant ID

But we still aren't done. How do we get the Tenant ID/userid into the database column in the base tables if it doesn't come through the view, and isn't in the application code. Again, the trick is that we are using the current userid as a surrogate for the tenant ID. MySQL unfortunately does not have a feature in the Create Table DDL to default a column value to the current userid on inserts as it does timestamps. Which is a pain, but easily overcome.

For every base table, create an insert trigger, whose function is to set the Tenant ID to the current userid system variable. Done! It is very simple code, very easy to validate, and cast iron guaranteed to make sure the correct tenant ID value goes into the row on insert. Because MySQL is getting the value being set. Not the application. And it's doing it after the application tiers have done all their data value manipulation so it's also bullet proof against web-tier attack.

You don't like triggers? For enterprise grade applications, a principal "must have" for me is having audit data in most tables, and usually comprising when a row was created, when it was last updated, and who did each. The "who" part has to be data passed by the application as our system userid is being coerced to be our Tenant ID. MySQL also is a little funky about timestamps. If you use the timestamp column type, you can construct the DDL to have a column auto-timestamp on insert, and the other column to auto-timestamp on update only. It's squirrelly though. And its been something that developers have wanted for ages — for MySQL to implement some obvious syntax to achieve this thing that we do to most tables on most applications. The other snag I have with using timestamps is that they are handled differently from datetime column types. And personally, I would rather have datetime formats doing the audit functions, to make it easier to integrate with my application and web tiers.

If I'm putting a trigger on insert for every single table to accommodate the Tenant ID, it is trivial to make that same trigger also do the On Insert datetime, and On update datetime stamping too, and do it with datetimes instead of timestamps. It's a side benefit to me for going this route.

Let's recap the solution

Instead of using a master connection user to handle database access, you would use a discrete userid, one per tenant account, and that is what becomes your tenant ID.

The tenant ID is not known to the end user or client administrator.

When you get a new client (a brand new Tenant) sign up for an account, the super user creates the new database userid and sves that value with the client account info as the Tenant ID. It should generate a new Tenant ID and password; it should create that user in the application schema, and grant that new user privileges to a security scheme. Your application needs will determine the access rights. This should be a simple role based thing.

You declare all your base tables to have a Tenant ID column.

You declare an update-able view for every base table which *excludes* the Tenant ID, but uses a where clause to equate the Tenant ID to the current system userid.

You create a trigger for every base table that sets the Tenant ID on insert to the current system Userid.

All application code uses the views to access the database. They should never use the base tables.

Your login authentication process is the only component in your application that uses the super user connection to retrieve the user's account info (which is has to do anyway to ensure your SaaS account is active and the user is valid), and retrieves the Tenant ID information. It then saves the Tenant ID to an application session or state variable so that all future connections the application makes to the data tier are made using that Tenant ID and not the super user id. If you want to be really serious about it, I would recommend putting user, account, and tenant ID information in a separate schema and database that only the superuser account has access to, and then in the application database, make it so that the superuser account has absolutely no access at all. Then an application cannot inadvertently open a back door by making a connection using the superuser account.

The result is that we have a simple multi-tenant implementation in MySQL 5 that is easy to maintain, guarantees all rows inserted have the correct Tenant ID, guarantees that all rows retrieved or update can only be for the current Tenant ID,

and it doesn't make the application code any more complicated as it just perceives it is going against a table, even though it is a view.

It works for me!

Database SaaS Software Development Software Architecture Multitenancy

Follow

Written by Tu Chu

297 Followers

"Mind-reading" products/engineers leader, a lifelong learner and a go-giver.

More from Tu Chu

Tu Chu in Laravel—In right way

Laravel Form for Update / Delete

HTML doesn't support PUT/PATCH or DELETE method, so it's a bit tricky to make the form sent your data with those method:

1 min read · Oct 6, 2016

88

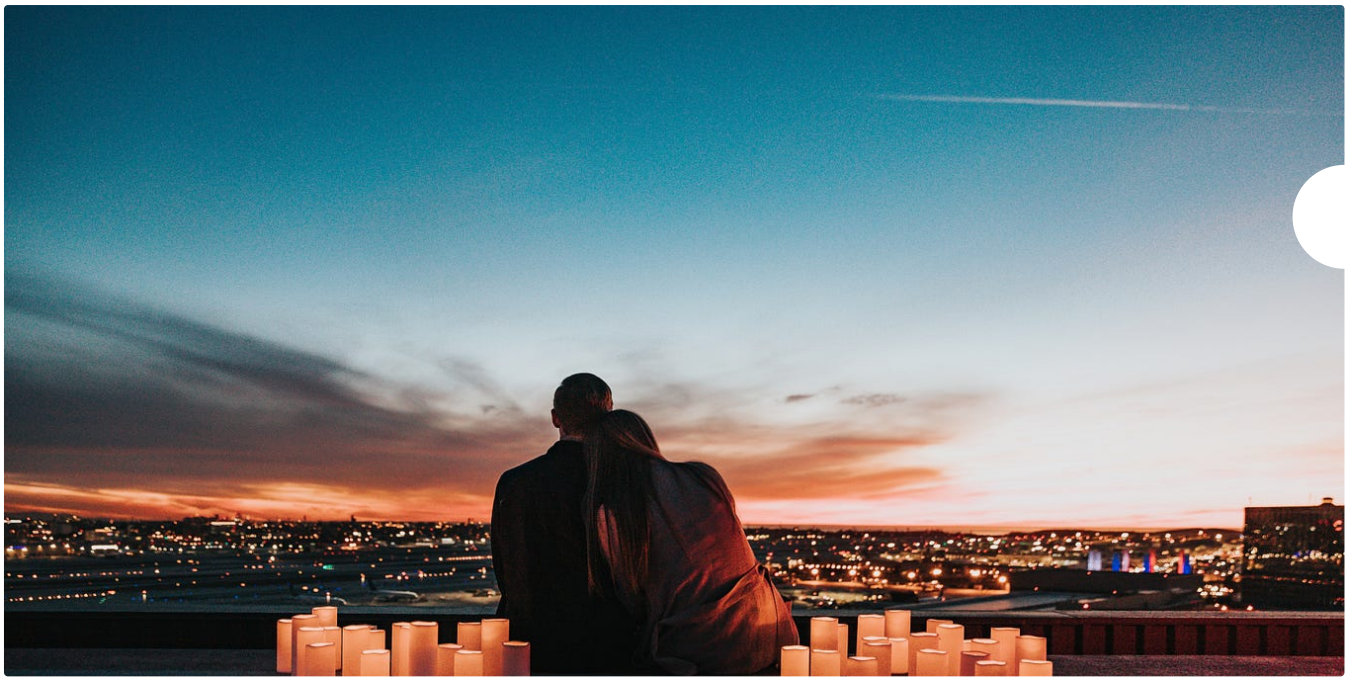
Tu Chu in Laravel—In right way

Using GuzzleHttp with Laravel

Guzzle is a PHP HTTP client that makes it easy to send HTTP requests and trivial to integrate with web services.

1 min read · Apr 21, 2016

1.1K 4



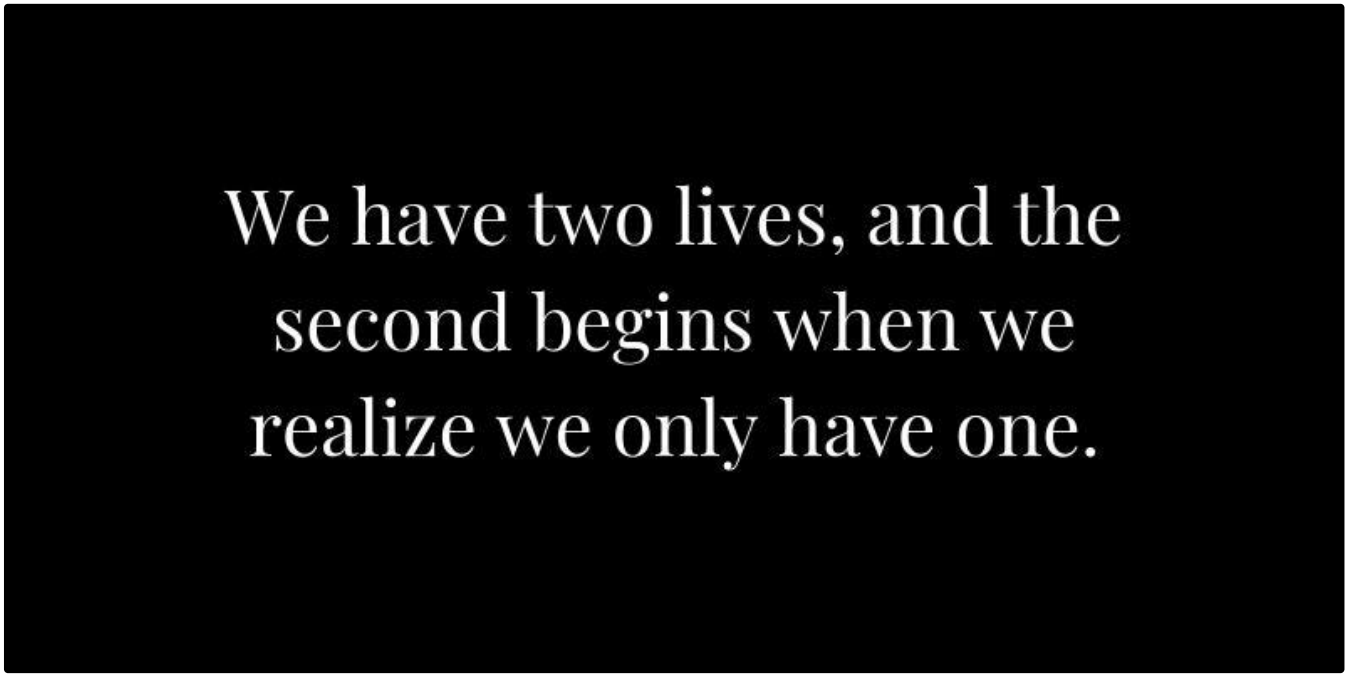
Tu Chu in Man in 30s

Gratitude vs Love

What are the differences?

2 min read · Mar 31, 2022

144



Tu Chu in Man in 30s

Crafting your own journey

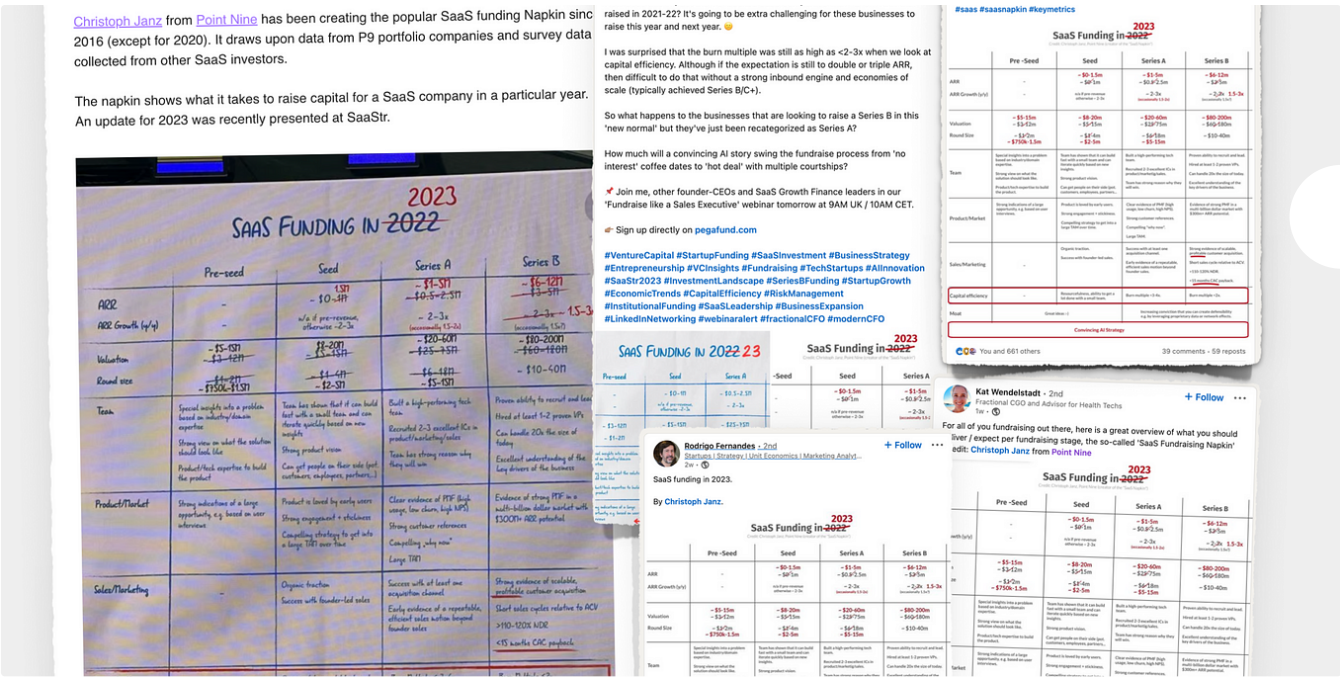
Each men has unique journey, and all stories are equally amazing! Here are some WHYs I started this publication as a way to keep noted with...

5 min read · Aug 25, 2021

150

See all from Tu Chu

Recommended from Medium



Christoph Janz in Point Nine Land

What does it take to raise capital, in SaaS, in 2023?

Unveiling the 2023 SaaS Funding Napkin

6 min read · 5 days ago

259 4

Avani Bataviya in Simform Engineering

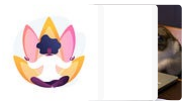
Office 365 login in Next.js using NextAuth.js

Secure user authentication using O365 (Azure Active Directory)

5 min read · May 2

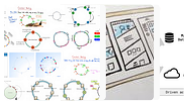
739 3

Lists



Stories to Help You Grow as a Software Developer

19 stories · 430 saves



General Coding Knowledge

20 stories · 406 saves

Coding & Development

11 stories · 201 saves

It's never too late or early to start something

15 stories · 150 saves

Wenqi Glantz in Better Programming

RDS Hybrid Multitenancy

How to implement hybrid multitenancy in one application with Spring Boot, Hibernate, PostgreSQL, and Liquibase

· 8 min read · Apr 13

166 1

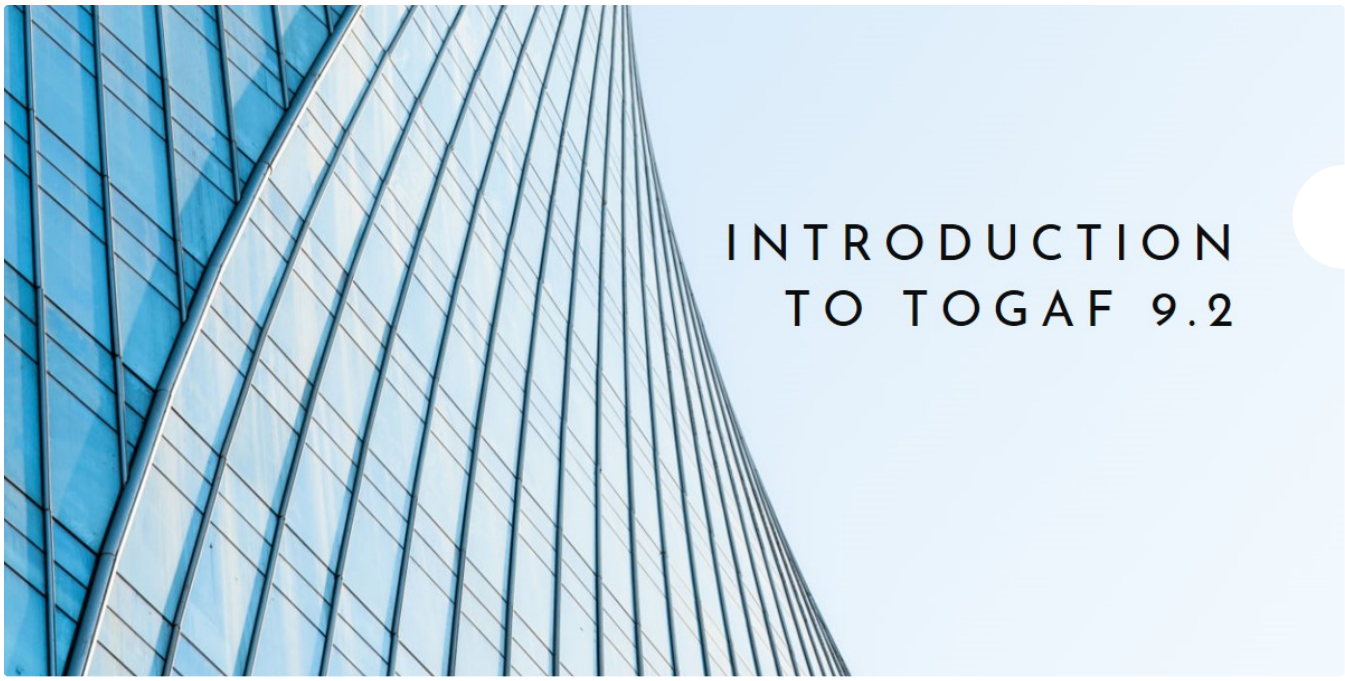
CCS Studios

Essential Services for SaaS Startups

Starting a Software-as-a-Service (SaaS) venture is an exciting journey filled with opportunities and challenges. Ensuring you have the...

2 min read · Sep 28

50



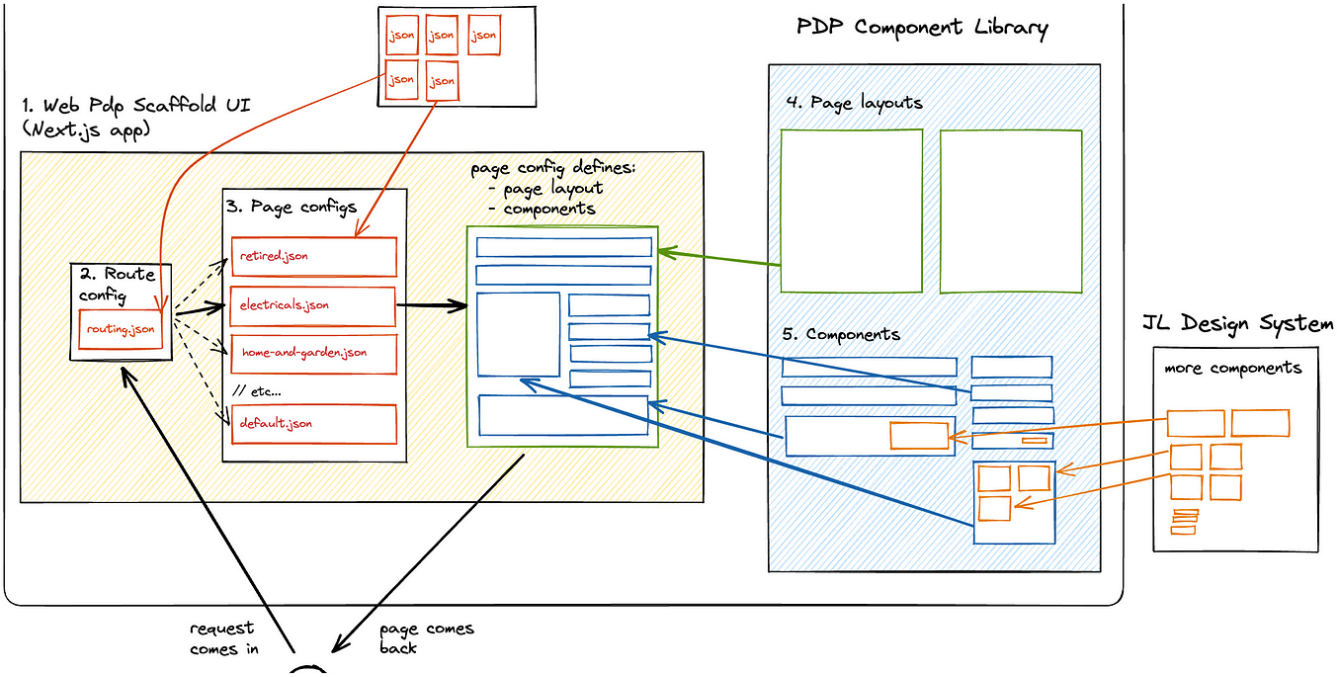
Wassim DRIRA

TOGAF 9.2: A Comprehensive Framework for Enterprise Architecture

TOGAF is a comprehensive framework for enterprise architecture (EA). It provides a set of best practices and guidelines for the...

14 min read · May 26

14



Owen Nicol in John Lewis Partnership Software Engineering

Config-driven layouts for faster experiments

Over the last few years, our John Lewis website (johnlewis.com) has been growing in ambition and scale, including allowing us to tailor our...

6 min read · Jul 5

100

See more recommendations