White Paper

Tenant - Sub Tenant Hierarchy and Tenant Aware applications

- Ramkumar R S

techcello

Contents

Introduction

In Multi-tenant SaaS applications, the word tenant is often assumed to mean a customer organization. But it need not be the only use case.

For a BPO's internal application, used by different verticals, each tenant can be one vertical, for which the application is configured differently.

For a BFSI (Banking / Finance / Securities / Investment) organization offering a market facing application, the tenant could mean one of the various channels (internal and external) that sell their products.

For an ISV with a global network of distributors and resellers, providing solutions to "Retail chains" the tenant could mean any and all of the following: A National Distributor, A Reseller, A Retail chain owner (Corporate office) and a particular retail store.

Another interesting case that we came across was this:

"…business users can use the application to set country specific rules. This application is expected to be deployed at least in 25 countries and we are considering each country as separate tenant with its own users, own authorization rules, own subset of customized fields, own set of process work flows, business rules and master list values"

So a even a country can be treated as a tenant, the channel partners within them sub-tenants, and the actual customers / user groups sub-sub-tenants.

Summary : Users belong to tenants and tenants belong to their parent tenants. Parent tenants can have their own users as well as further parent tenants above them.

So the concept of "tenant" refers to "user groups" with parent – child relationships between each group and unlimited nesting.

Why does this matter?

A broader definition and understanding of the word tenant, automatically expands the meaning and scope of "Multi-tenancy at the Application Layer"

Multi-tenancy at the infrastructure layer is about sharing the hardware across various tenants.

Multi-tenancy at the database layer is about grouping one (dedicated) or more tenants in a database.

Multi-tenancy at the Application layer is about configuring and customizing the application to suit different tenants while still maintaining a single code base.

To this definition we apply the broader context of the word tenant and parent-child relationships between tenants.

Two extremes:

A Multi-tenant Application (at the application layer) could still use a dedicated database and dedicated hardware box for a large customer.

A Cloud or PaaS hosted application, might share the infrastructure and database across multiple tenants, but still might be “Single tenant in its thinking”.

This means the application cannot be configured or customized for different tenants, without changing the underlying code. Or the requirements of all user groups are so homogenous that the application need not be tenant aware, except while storing and retrieving data.

So Multi-tenant SaaS is not a business model but a “way of thinking” and a “way of life” for architecting, engineering and building custom applications.

Tenant aware applications

Multi-tenancy has to flow through the veins and nerves of an application. It cannot be an afterthought or a bandage applied on the surface of an application.

Some questions to explain this:

Can a custom field added specifically for a tenant, be looked up and used in a business rule screen or a report building screen?

Can the tenant specific data scope policies and privileges be enforced across all parts of the application during run time, based on user context?

Can the licensing and subscription module talk to the authentication system and metering system to stop / limit access to a tenant’s users once the metered quota exceeds the specified limit?

Can the business logic and process flows of the application change dynamically depending on the context of the user and tenant during run time?

Can business users (non IT staff), setup, configure and customize the application at the tenant level using GUI administration screens accessed through an internet browser?

Tenant aware applications bring significant business benefits and competitive advantages to an organization owning or using the application. If you are an architect or developer thinking and talking about this, it is one sure shot way to get noticed by your CEO / CTO/CIO.

The Power of Tenant Hierarchy

When you add Tenant hierarchy to tenant aware applications, it has a very disrupting and non linear effect on the flexibility and maintainability of the application. Now complex things can be done with much more ease and on the fly.

For example:

Parent tenants control the access rights of downstream tenants – including what further customization they can do, and what they cannot. Configurations done at a parent tenant level, automatically role down to all the downstream child tenants. Child tenants can create tenant specific roles, assign privileges to these roles and ensure that even their parent tenants do not have access to these privileges or data.

How Techcello addresses these challenges

CelloSaaS has the following features to handle the Tenant – Sub Tenant Hierarchy

Tenant relationship management: Manage the relationship between tenants including the tenant fallback.

Tenant fallback: When a relationship chain is broken there needs to be a provision to assign a different tenant.

Tenant scope: Set the read, update and delete access of data at an entity level across tenants based on the relationship between the tenants. This access rights is flown across all the other features within the framework.

Tenant data isolation: Though some data needs to be shared between tenants there is a clear data isolation boundary set at an entity level by the tenant beyond which data cannot cross.

Data sharing between tenants: There could be few data that are owned by multiple tenants which are related.

Tenant Settings: Tenant settings are created in the form of templates and can be assigned to tenants easily. Who could control the tenant settings are again driven by tenant scope.

Configuration inheritance: This allows the configurations set at a parent level to be inherited automatically to the child tenants. If the child tenants need separate configuration they could copy the parent settings and override their configurations.

All the things that we have discussed in this paper are addressed at the celloSaaS framework level.

Leaving these crucial engineering aspects to individual developers is highly error prone and time consuming from both development and testing perspective. It may also lead to operational inefficiencies.

Conclusion

Multi-tenancy is not only about sharing infrastructure and database. Designing or building tenant aware applications, is a “way of thinking” and “a way of life”. Whether we are designing a new application or migrating an existing application to a tenant aware architecture, Multi-tenancy has to flow through the veins and nerves of an application. It should not be applied as a bandage on the surface.

When we combine Tenant Sub-tenant hierarchy along with a broader definition of who is a tenant (even countries / verticals / divisions can be tenants), then such tenant aware applications could bring significant business benefits and competitive advantages to the organization owning or using the application.

“Multi-tenant SaaS” is not a subject matter for Architects and Developers alone. It is entering the mainstream consciousness of CEOs, CTOs and CIOs.

About Techcello

Techcello’s Multi-tenant Application Development Platform and the celloSaaS Framework, is useful for building any .NET application: Small or Big, Simple or Complex, Cloud or On-premise, Enterprise or BPO, established ISV or a promising startup.

All the capabilities are available as ready to use APIs and Services. Some of the capabilities are shipped with default GUI screens that can be further extended or modified.

It is delivered as a package that can be installed on the developer’s machine and used from within Visual Studio. The binaries of the framework can be deployed along with the application anywhere. The run-time remains .NET (Windows / SQL / IIS).

For more information www.techcello.com

About the Author

Ramkumar is the Director of Product Management at Asteor Software. He was instrumental in incubating and bringing to market two new products in the Cloud / SaaS space: Techcello and Synergita.

Ramkumar is also the founder Director of Mango DVM an angel funded company in the Digital Music space, that has just completed a third round of funding.

Before becoming an entrepreneur Ramkumar, had spent over 2 decades in various corporate functions such as Automation Engineering, Project Management, Marketing, Sales , General Management, HR and Leadership Development.

He can be reached at rsr@rsrinnovations.com or ram.k@techcello.com