# White Paper

## Non Functional Requirements of Government SaaS

- Ramkumar R S

techcello

**Contents**

Abstract – Summary....4

Context....4

Government SaaS....4

Functional Vs Non Functional Requirements (NFRs)....4

Why NFRs are more important and critical in Government SaaS?....5

1.0 True Multi-tenancy at the Application level....5

1.1 Who is a Tenant in Government SaaS?....6

2.0 Scalability....6

2.1 Data Connection Abstraction....6

2.2 Data Isolation....6

2.3 Distributed Caching and Stateless Design....6

3.0 Customizability and Configurability by Non IT Personnel....6

3.1 Themes and Logos....7

3.2 Associating CSS files to Tenants....7

3.3 Forms and Grids....7

3.4 Data model extension....7

3.4.1 Custom fields on Forms and Grids....7

3.4.2 Custom fields on Queries and Reports....8

3.4.3 Custom fields on Business Rules and Workflows....8

3.5 Business Rules Customization....8

3.6 Workflow Customization....8

3.7 Report Customization....8

3.8 Notification Template Customization....9

4.0 Configurable Access Control System .....9

4.1 Tenant specific roles and role – privilege mapping.....9

4.2 Dynamic Data scope policies.....9

4.3 Single Sign on.....10

4.4 Tenant – Sub tenant Hierarchy.....10

4.5 Tenant Configuration Templates.....10

5.0 Other NFRs.....10

5.1 Notification.....10

5.2 Schedulers.....10

5.3 Auditing.....10

5.4 Metering .....11

5.5 Usage Auditing .....11

5.6 Module & Feature provisioning .....11

5.7 Pre and Post Processors and Policy Injection.....11

5.8 Logging and Exception Management.....11

5.9 Master data and Pick up lists Management.....11

6.0 NFR Stack Design .....11

A Typical NFR Stack / Framework for a Multi-tenant SaaS Application.....13

About the Author.....14

**Abstract - Summary**

A Government SaaS that addresses a diverse set of stakeholders and multiple levels of user groups is much more complicated and technically challenging than a Private SaaS application that is often focused on a niche homogenous user base. The Non functional requirements (NFRs) of such a Government SaaS are hence very critical and important.

Defining the NFRs of Government SaaS independent of the functional requirements has many benefits. It leads to a focused effort on NFRs, a robust architectural framework, engineering excellence and consistency. The NFR stack when implemented as a loosely coupled framework, could also become a common stack for multiple Government SaaS applications, resulting in significant reduction in cost of development, support and maintenance (TCO).

**Context**

SaaS refers to the delivery of Software (to its users) through a “Software as a Service” Model, wherein the various stakeholders using the application simply use it from an Internet Browser, without having to own and manage the hardware and software infrastructure required for it.

The SaaS provider owns the software and is responsible for its uptime and availability. The SaaS provider is also responsible for providing the necessary functional and non-functional features required in the application, and continuously enhance them as per the requirements of the users and stakeholders.

The application could be hosted on-premise in a hardware exclusively owned by the SaaS provider or it could be hosted on a secure Private or Government Cloud.

**Government SaaS :**

In this case, the SaaS provider is the Government. The actual responsibility of development and management of the application could have been assigned to either a government agency or an external private vendor.

The customers using this SaaS could be other government departments, implementation agencies, outsourced vendors and would include citizens who have to interact with the Government through the SaaS application.

**Functional Vs Non Functional Requirements (NFRs)**

While building any application, the focus is often on the Functional requirements. These are capabilities and features that the users expect from the application. The functional requirements could vary from one group of users to the other and from one domain to the other. There can be multiple applications and multiple modules within an application to cater to various usage scenarios.

However, there are non-functional requirements for any application, many of which are latent and implicit. Some of the NFRs could be so common sense, one might wonder, whether it is really required to explicitly articulate it.

In the context of SaaS and more importantly Government SaaS, the NFRs are not only important, but critical for its successful deployment, implementation and usage. Let us see why?

**Why NFRs are more important and critical in Government SaaS?**

Let us take a simple example of "Performance Management" of Government Staff. It is a horizontal application that is needed by almost all parts of the Government, but the functional requirements of each division, department, agency and vendor will have overlapping commonalities as well as unique variations. Trying to build, maintain and support a customized system for each group is costly. At the same time, a standard application cannot be forced on everyone.

So how to build and deliver a common software system in the SaaS mode, while still retaining the flexibility required to configure and customize to suit different user groups, is itself a Non functional Requirement. The need for this capability is more pronounced and critical in a Government SaaS than in Private SaaS. A private SaaS provider can focus on a niche consumer segment with homogenous requirements. A Government SaaS on the contrary has to work in a diverse environment but needs to be developed, deployed and maintained within tight budgets.

Like Customizability, there are many other NFRs (such as Cloud ready architecture, Scalability and Security) which are more critical in Government SaaS than in a Private SaaS.

**1.0 True Multi-tenancy at the Application level**

The technologies associated with SaaS and Cloud provide substantial savings to the SaaS provider as well as its customers, only if it is designed and architected as a Multi-tenant application with a single code base. Using Single tenant architecture with a separate server instance running for each customer / tenant could be a quick way to go-to-market. However this approach might work for some niche scenarios or a stop-gap arrangement but cannot be a permanent solution for Government SaaS.

A multi-tenant application, does not become multi-tenant, just by sharing the underlying hardware infrastructure or database software. For example, creating separate schema for each customer on a single database server and customizing the data model to the unique requirements of each customer is an easy way out. But the objective is not just to save on hardware costs and licensing costs, but to ensure scalability and maintainability at low cost.

So multi-tenancy in the context of computing infrastructure (where multiple customers / tenants share the same infrastructure pool) is different from multi-tenancy at the application level. A multi-tenant application has a single code base that can be configured and customized for multiple tenants.

### 1.1 Who is a Tenant in Government SaaS?

A tenant in the context of Private SaaS, is usually a customer organization that pays and uses the SaaS provided by the SaaS provider.

A Tenant in the context of Government SaaS could be any set of users grouped together by common and homogenous needs. The tenant (User Group) could be set up and managed independent of other tenants. The application could also be customized and configured to suit the requirements of each tenant.

## 2.0 Scalability

### 2.1 Data Connection Abstraction

In today's world, tying your code to any specific database is no longer accepted. Databases could be scaled out – partitioned vertically or horizontally as they grow. The data partitioning should be configurable during run-time. So it is important to abstract the data connection and pass all the Data access through an intelligent Multi-tenant DAL layer. This layer would do the connection string management during run time, based on the context of the user / tenant and based on the tenant configuration templates.

### 2.2 Data Isolation

Each tenant's data could be stored in a separate database, or in a shared database with separate schema / tables or in a fully shared table with a tenant ID. In all these cases the data model should be same for all tenants. (Customization of data model is discussed separately in a forthcoming section)

Developers should write code and queries as if they are writing for a single tenant. The intelligent Multi-tenant DAL layer should insert the tenant context during run time. This ensures that one tenant's data does not get mixed up with another tenant's data because of a developer's mistake.

### 2.3 Distributed Caching and Stateless Design

The NFR stack should use Distributed Caching and Stateless design, so that depending on the usage load, the server instances can be scaled up or down, without affecting user experience. By using a Load balancer and theoretically unlimited no. of Server instances, the application should be scalable to any no. of concurrent users.

## 3.0 Customizability and Configurability by Non IT Personnel

Customizability does not mean, that scripts and dedicated code are written and plugged in by IT personnel to meet the needs for each tenant. The aim should be to allow Power Users (Tenant level Administrators) to

customize and configure the application to suit the requirements of their User Group, without having to depend on internal or external IT Staff. These admin screens should be accessible through a normal browser.

Customization and configuration should be supported for the following :

### 3.1 Themes and Logos

Each tenant should be configurable with their own Logos and Color Themes. All the UI screens in the application should reflect this.

### 3.2 Associating CSS files to Tenants

Many a times, the requirements of a particular tenant could be such that, changing the logo and the color scheme alone is not sufficient. The entire layout of the application pages and the graphical images used in them might have to be customized. The application should allow Power uses to create, or select from uploaded CSS files and associate them with individual tenants or for a set of tenants.

### 3.3 Forms and Grids

Display names / Column headings used in each form / grid, the order in which the fields / columns appear, the visibility / hide condition, whether the field is mandatory or not all these need to be customized by a power user. This feature may not be required for all forms and grids in the application, but will be required in at-least some of them.

### 3.4 Data model extension

When a tenant customizes the application to add one or more custom fields to an entity, the transaction database should not be disturbed. The extension fields should be stored separately in a separate table / database as key value pairs along with tenant ID and entity ID.

Data model extension should be possible through an administration GUI screen which will be used by Power users who have the necessary rights and privileges to do this.

While fetching data or saving data on an entity, the extension fields should automatically get appended depending on the context of the user and tenant.

Unlimited no. of custom fields should be supported for each entity. However certain entities that may not require customization can be excluded completely. (Data model extension feature will be disabled for those entities)

#### 3.4.1 Custom fields on Forms and Grids

The power user should be able to add the custom fields (created using data model extension) in to any form or grid that uses the entity.

### 3.4.2 Custom fields on Queries and Reports

While creating Queries and reports that can be customized by a tenant, the power user should be able to view and include custom fields associated with that tenant.

### 3.4.3 Custom fields on Business Rules and Workflows

While customizing business rules and workflows, the power user should be able to view and include custom fields associated with that tenant, as part of the rule / work flow set up.

## 3.5 Business Rules Customization

Hard coding of business rules or using developer centric tools such as Microsoft Workflow Foundation is acceptable only for those scenarios where there is no end user level customization needed. But if there is a need to allow power users to customize certain business rules through the GUI, then we need to include an additional Business rule engine as part of the NFR. This engine will allow the developers to specify the business logic, variables, and other parameters during development and also expose them to the GUI for a power user to configure and customize. While such an engine should be used only for end user customizable scenarios, it should interoperate with other developer centric tools for more complex requirements.

## 3.6 Workflow Customization

This is similar to BR Customization. The developers should write the domain specific activities that are required in the application and expose these activities to a Workflow Designer. The GUI based workflow designer will be used by Power users to configure the sequence of activities with parallel and branching trees. The power users will also set up the start – stop and other conditions required for the execution of the workflow. The workflow steps could be manual or automatic. Manual steps will advance depending on the status returned from user driven activities. Automatic steps will advance once the business logic within them is executed.

The workflow customized by each tenant should be stored as meta data. During run time, the workflow engine, should load the appropriate version depending on the context of the tenant and execute the same.

Multiple workflows should be capable of running at the same time. Some of the workflow activities could be background jobs that can be scheduled and scaled out independent of the application.

## 3.7 Report Customization

3.7.1 Canned reports could be customized by tenants to change column headings, hide / show certain columns and add tenant specific custom fields if necessary.

3.7.2 Power users should be able to create Adhoc queries using a GUI based Query builder. They can drag and drop fields from entities that have been exposed to this tool during development. While doing this, tenant specific custom fields should be automatically shown in the respective entities.

3.7.3 The rights / privileges and data scope policies defined in the Access Control System should be enforced during Adhoc queries / reports, so that what users see is automatically filtered to match their data access privileges.

3.7.4 A Report / Chart designer could help power users to create their own reports and charts using queries created with the Adhoc Query builder.

**3.8 Notification Template Customization**

The notification templates used for each tenant should be customized through the UI so that the power user can set it up to suit the tenant organization.

**4.0 Configurable Access Control System**

Different tenants (User groups) might want to follow different policies on Access Control with respect to the data and features provided in the application. CRUD privileges related to database entities, named privileges associated to each entity, privileges defined at the field level, page level, service level and feature level – all these have to be configured for each tenant and user.

During development, the developer should focus on privileges and demand them when necessary. The roles and users could be set up during run time.

**4.1 Tenant specific roles and role – privilege mapping**

Tenants should be able to create custom roles for their organization and assign them to the users. They should be able to map the various privileges to the role and thus control "who can see what and who can do what" in the system.

These tenant specific access control policies should be remembered as meta data, which then should be enforced dynamically during run time, depending on the context of the user and tenant.

**4.2 Dynamic Data scope policies**

Often it will be needed to put a boundary on the data that can be accessed by a specific role. For example a Finance Manager should be able to see data only related to his division, or a HR manager should administer the application only for employees located in say 3 locations assigned to him.

In such scenarios the variable based on which the data scope is defined (Example division, location) could itself be a tenant specific master that the tenant could have created and customized for their organization. So the mapping of the Data scope policy should be done during run time by pulling up the tenant specific master data. The power user would be able to map them as one of the data scope conditions.

### 4.3 Single Sign on

Many tenants will have their own internal identity management and user authentication system. It should be possible to integrate the User authentication of the SaaS application with the authentication system of each tenant through federated authentication. This should follow the standards provided by the SAML protocol.

### 4.4 Tenant – Sub tenant Hierarchy

While each tenant is a group of users set up and managed independent of the other tenants, there needs to be a hierarchy among the tenants. For example a nodal agency could be a parent tenant and multiple agencies below it could be set up as child tenants. The child tenant in turn can have further smaller contractors who could be sub-tenants. And users could be created within these sub tenants.

All the customization, configuration and access control settings done by the Power Users should be aware of the Tenant – Sub tenant hierarchy, and these settings / configurations should automatically roll down for the child tenants.

Users can also be created with access to multiple tenant's data. This is generally required by members of service teams that look at data across Tenants for an unified view or for reporting. Permission from the respective Tenants should be obtained before activating such users.

### 4.5 Tenant Configuration Templates

Tenant specific configurations could be set up as a template and these have to be automatically rolled down to all the Child tenants. Child tenants could be allowed to selectively customize their own settings and these will be automatically rolled down to downstream tenants.

## 5.0 Other NFRs

Some of the other NFRs are mentioned below in brief.

### 5.1 Notification

Should support FTP and email notifications, both in instant and batch mode. The templates for notification (at tenant level) should be customized by the Power user through the UI.

### 5.2 Schedulers

Time and Event based schedulers should be available as services, which can be used in the business application.

### 5.3 Auditing

The system should support Auditing Services, where audit trail of any transaction or event could be activated by simply calling a service.

### 5.4 Metering

Metering of events and transactions will be required and this can be used to set usage quota for each tenant / user group. This should be tied in to the access control system.

### 5.5 Usage Auditing

This will be used to monitor the usage level / pattern of the various system components by different user groups. The data can be later used for Product Analytics.

### 5.6 Module & Feature provisioning

Not all modules and features should be accessible to all the user groups / tenants. A higher level administrator (SaaS Provider / Parent Tenant) should be able to control which modules and features of the application are accessible by which tenant.

### 5.7 Pre and Post Processors and Policy Injection

The system architecture and plumbing layers should support pre and post processors and policy injection mechanisms that are also tenant aware / tenant specific.

### 5.8 Logging and Exception Management

Tenant wise Logging and Exception management will help in tracing and trouble shooting tenant specific problems.

### 5.9 Master data and Pick up lists Management

Master data and Pick up lists created for each tenant should be rolled down to sub tenants, but should remain customizable by each tenant.

## 6.0 NFR Stack Design

A piece meal approach of building the NFRs described in the previous sections, will lead to inconsistency. Different developers will deal with these issues differently. Multiple applications requiring the same NFRs are often built ground up, following different approaches. Maintenance and Support of such applications also become expensive.

Traditional on-premise applications built for a specific user group (Single tenant architecture) have an NFR overhead of 10-20%. This means in a 100 people month project, 10-20 people months are spent on NFRs.

Whereas in Multi-tenant SaaS applications, the time and cost overheads associated with NFRs could be as high as 30-50% of the total development effort. That is, in a 100 people month project, 30-50 people months have to be spent on NFRs.

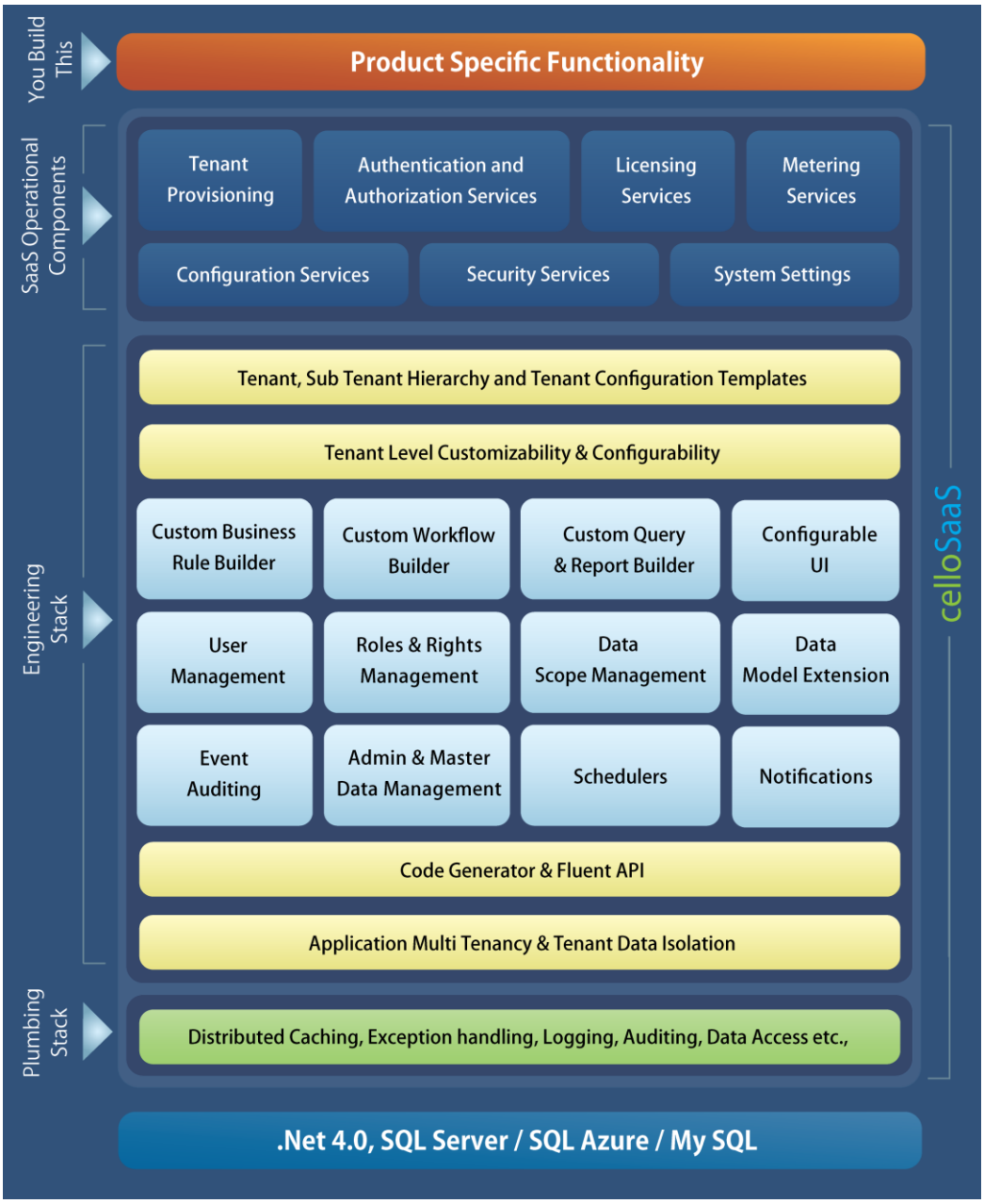
Ideally, NFRs have to be handled at the architectural and framework level. A Strong foundation stack or framework (NFR Stack or Engineering Stack) can be built and maintained independently. Using this, developers can focus on building domain specific functionalities instead of struggling with the plumbing and engineering aspects.

The NFR stack / framework should be loosely coupled with the functionality layer. The SaaS provider should be able to maintain and upgrade the NFR stack / framework independent of the domain specific functionalities.

The SaaS provider should also be able to re-use the NFR stack / framework for other SaaS applications. This ensures consistency and quality across multiple applications while reducing overall cost of ownership.

While building or buying such an NFR Stack / Framework, care should be taken that the SaaS provider retains ownership and control over the entire application stack both technically and strategically. Vendor lock in or platform lock in should be avoided.

**A Typical NFR Stack / Framework for a Multi-tenant SaaS Application**



**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