Paper 1372-2017

Build Apps for your Enterprise with SAS® and HTML5

Nikola Marković, Boemska TS Ltd.

ABSTRACT

SAS is perfect for the job of building Enterprise Apps. Think about it: it speaks to almost any database you can think of, and is probably already hooked in to most of the data sources in your organisation. A fully fledged metadata security layer happens to already be integrated with your single sign-on authentication provider, and every time a user interacts with the system, their permissions are checked and the data their app asks for is automatically encrypted.

SAS ticks all the boxes required by IT, and it means that the skills required to start developing Apps already sit within your department. Your team most likely already know what your App needs to do, so instead of writing lists of requirements, give them a HTML5 resource and together they can write and deploy the back end code themselves. The Apps run in the browser, the server-side logic is deployed using SAS .spk packages, and permissions are managed via Management Console. Best of all, the infrastructure that would normally take months to integrate is already there, eliminating barriers to entry and letting you demonstrate the value of your solution to internal customers with zero up-front investment.

This paper will show how SAS integrates with Open Source tools like HTML5 Data Adapter for SAS (H54S), AngularJS and PostGIS together with next generation developer-centric analytical platforms like SAS Viya, to build secure, Enterprise-class Apps that can support thousands of users.

INTRODUCTION

This paper targets a variety of audiences both technical and not, with the intention to demonstrate how building Apps in SAS may be beneficial to your organisation. Some of the material presented here is intended to enable the reader to persuade other stakeholders of the benefits, to help get the kind of engagement necessary to develop and implement effective solutions. To that end we have split this paper into three main sections. The first, ‘Why build Apps in SAS’, is an introduction to how this approach fits within the big picture of business and enterprise IT, and outlines its benefits. The second, ‘How do I build Apps in SAS’, covers some of the technical and architectural aspects of programming a SAS-based Web Application with SAS, HTML5 and the Boemska HTML5 Data Adapter for SAS. Finally it takes a brief look at our ‘Sample Application’, which tackles the difficult task of demonstrating some of the types of integrations available to applications developed on the SAS Enterprise Analytics Platform using the medium of print.

WHY BUILD APPS IN SAS

When working within the constraints of Big Enterprise IT, an all too common occurrence is that in order to get a desperately needed application developed, business teams are forced to coordinate all of their actions through their often monolithic and risk averse IT department or service providers. Should they manage to overcome this inertia, they then are required to exactly specify and justify their requirements before passing them on to remote development teams which have little to no first hand understanding of

the problem they're trying to solve. They then have wait in line for months, sometimes years, before receiving a potentially obsolete-on-arrival interpretation of the solution they originally needed.

This problem is not new - for most large organisations it's simply an accepted side-effect of scale. Over the years this problem has been somewhat mitigated by what many industry analysts refer to as 'citizen developers': a class of power users and business experts who, left with no other choice, took it upon themselves to develop solutions for their teams and wider business units using the 'business-owned' tools they had at their disposal. Many a time it has been precisely these 'citizen developers' that have kept organisations ticking over with 'good enough' operational solutions that they built using the tools they had available at the time: generally Excel spreadsheets, VBA macros and network-drive-based-data-marts.

This same demand for rapidly developed business solutions, combined with recent advancements in HTML5 as a front-end technology, has enabled a growing market of 'Low Code Application Development Platforms'. These products often target the same segment of frustrated business-power-user, painting a vision of business-owned self-service app development - a world where empowered SMEs are free to rapidly develop and deploy secure, database-backed applications fronted by modern User Experiences, but with little or no involvement from (or dependence on) their IT department or service provider.

Most of these 'low code platforms' are very similar in architecture: they tend to come with a few simple database connectors, letting end users model data, create some basic user interfaces, bind their data models to certain elements in their UIs, and offer some sort of deployment mechanism for the apps they build, possibly with some basic authentication integration. The problem these platforms share are precisely that they are 'low code': that is, that *they still require custom code to be written in order to integrate with other systems within the organisation*. While the slick graphical HTML5-based interface designer functionality makes it easy to prototype application interfaces, these last mile 'customisations' mostly have to be implemented in languages like Java or .NET, forcing 'citizen developers' to either learn to program in an unfamiliar programming environment (with no prior experience of enterprise integration), or to revisit their old IT department and ask them for help. It's a problem that many other GUI-based code generation interfaces face: namely that once the developer hits the limitations of the GUI-based approach, in order to progress they are forced to revert to code-based workarounds and 'customisations' often more complex than they would have needed to be had the project been implemented using a standard 'codeful' approach from the beginning.

Nevertheless, and limited as they are, the fact that these tools are rapidly gaining in popularity is of little surprise. Even with the aforementioned challenges, they succeed in delivering value by the simple fact that they solve real business problems sufficiently - through simple, usable solutions, developed by the SMEs best positioned to develop them, while the need for them still exists, with a relatively small startup cost.

The Citizen SAS Developer and HTML5

Within many organisations, SAS developers both *manage* the data held within that organisation, and are responsible for *interpreting* that data (ie. developing reports and communicating their meaning to the wider business audience). As a result, SAS programmer analysts often hold some of the best, most detailed understanding of how that organisation actually operates, and what could be done to improve its operation.

Maybe due to SAS's history as a desktop-based analytical application, and maybe because it's mostly used for Business Analytics, SAS has always been considered a *business-owned tool*, likened more often

to userspace tools like MS Excel or MS Access than to IT-owned applications such as database platforms or Java or .NET application servers. This is interesting because, in addition to being great at Analytics and Reporting, SAS has also historically proven itself as an extremely capable Application Development Platform. SAS first began targeting application developers with their SAS/EIS and SAS/AF products many years ago, and these products have been used to develop a vast array of powerful Enterprise Applications since. The fact that we still see so many operational SAS/AF-based applications in use today is testament to the value of SAS as an Enterprise Application Platform.

However, much like the rest of the industry, SAS eventually moved away from using desktop-based thick clients and towards web-based thin client interfaces, meaning that most of the front-end UI components that came with these AppDev products have slowly became obsolete. Nevertheless, although its web-based presentation layer may no longer be as capable as it was on the thick client, the flexibility and power of SAS at the *back end* has meant that power users in large organisations still frequently use it to develop quick, data-driven business solutions - especially where an application needs to interface with an existing operational system, or the access to another application's data needs to be carefully controlled.

Using a combination of SAS Stored Processes and some simple HTML, SAS developers are still able to easily put together simple, functional, secure applications, and roll them out to their users via their web browsers in a controlled manner. This control is managed by the combination of SAS Stored Process Web Application (SPWA) and SAS's highly granular, role-based Metadata security layer, primarily designed as a means of securely rolling out its Self-service Reporting and Data Management capability. This works well because these components are ubiquitously available: the SPWA is a SAS Integration Technologies component common to most web-enabled SAS deployments, and the SAS Metadata Server is a core part of the SAS Enterprise Platform architecture - almost always configured for Single Sign-On and integrated with the organisation's standard Authentication Provider at the point of installation.

It is therefore no surprise that, for many SAS programmers, this idea of a *citizen developer* is a familiar one. What is most interesting here is that, while SAS's presentation layer pales in comparison to the shiny HTML5 interfaces offered by the new platforms, SAS-built solutions remain *far superior* in terms of ability to integrate and maintain into larger institutions. Some of this is due to SAS's sophisticated role-based Metadata Security implementation, but it's SAS's proficiency in Enterprise Integration that puts it in a class of its own; the database connectivity and transformational control that it brings as a Data Management platform not only means that no system new or old is out of bounds when it comes to connectivity and app integration, but also that code to talk to that system probably already exists somewhere within the same SAS team, alongside at least a 'good enough' knowledge of its function and data model.

Finally, when considering these technologies it is important to note that almost every one of the Low Code tools on the market today is dependent on at least one mainstream Open Source front-end framework, subsequently *extended* by the vendor to form their proprietary platform. In most cases, the front-end tooling responsible for the bulk of the cross-platform User Experience they offer is already freely available - the platforms just add some 'builder' functionality and a facility for enterprise integration. In addition (and with the usual critique of vendor lock-in aside), the fact that these platforms have little *philosophical* appeal to developers is a very real problem. It is far more difficult to find a talented engineer willing to risk skilling up on a proprietary framework with little market penetration and an uncertain future, than it is to hire one with experience of (and enthusiasm for) a widely adopted mainstream front-end framework like Google's AngularJS or Facebook's React.

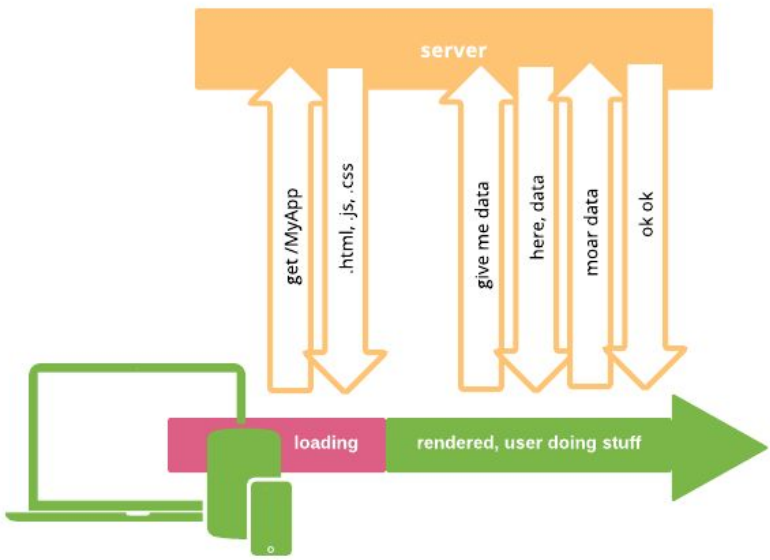
The rest of this paper outlines architectural principles which enable the vanilla integration of those same mainstream front-end frameworks with the SAS Enterprise Analytics Platform.

HOW DO I BUILD HTML5 APPS WITH SAS

What is a HTML5 APP

HTML5 is the latest iteration of the HyperText Markup Language - a markup used to describe Web Pages, which, when interpreted by a Web Browser (alongside CSS files and JavaScript code) allows for thos e pages to be displayed. Although HTML5 is actually just an iteration of the markup language, the term is very commonly used to describe this triad of technologies that accompany it - HTML5 for markup, CSS3 for styling, and JavaScript for scripting. With the latest iteration, these technologies have enabled the rapid development of cross-platform solutions previously only possible with native Apps, and it is responsible for the Web & Mobile Web application experience we have today.

When trying to picture how HTML5 Apps work, it can be easier to just think of them as Web Pages. When an App runs, what really happens is the HTML, CSS and JavaScript files are loaded from a server (or local storage), the browser's rendering engine renders the page, and the Javascript code on the page *runs.* Subsequently, as a user interacts with elements on that page, other JavaScript code handles the interaction events, and where required sends data (back) to a server or retrieves new data, updating what is displayed to the user.



This principle where a web page sends requests for data to a server in the background without refreshing entirely is commonly referred to as AJAX (Asynchronous JavaScript and XML). In terms of the HTTP protocol, these requests are no different to the requests a web browser has always made when a user has typed something into the browser's address bar and hit enter; where they differ is in the structure of the data that is sent format in which that data that is returned. Many server technologies are commonly used to serve application data, and although the data itself is usually encoded using a notation known as JSON, there can be considerable variation in how those requests are formed and how the responses are handled. As a result, much of the work in implementing this communication is a manual process requiring close cooperation between the front-end and back-end developers.

This process can sometimes be made easier by using a library to facilitate that communication. The **Boemska HTML5 Data Adapter for SAS** is a library which aims to provide a familiar programming

interface to ease the process of implementing this communication, for both JavaScript front-end programmers and SAS back-end programmers.

The Boemska HTML5 DATA ADAPTER FOR SAS (H54S)

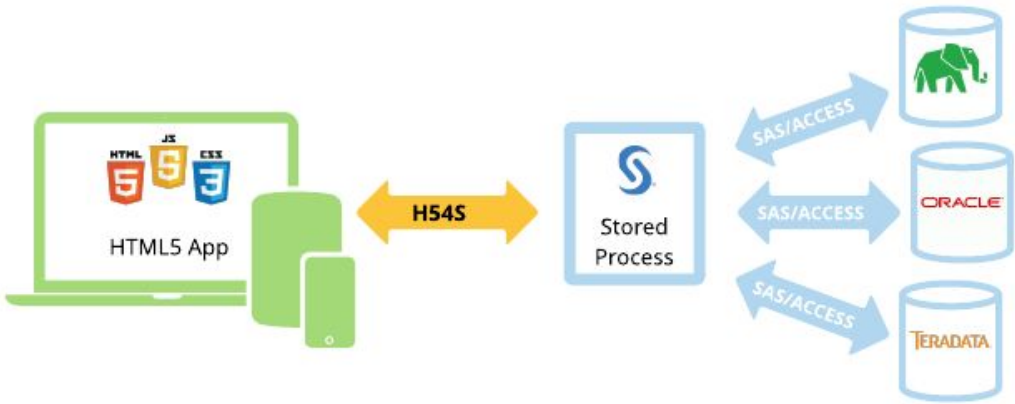
Note: A detailed overview of hands-on application development using these methods is available in Allan Bowe's (excellent) SGF 2017 paper, titled ***SAS1091-2017: Build Lightning Fast Web Apps with HTML5 and SAS®****. This paper focuses on application architecture, and the code provided here is used as an example only. If you are looking to get started programming your own apps, then Allan's paper is highly recommended… once you've read this one.*

The H54S GitHub page at github.com/boemska/h54s describes the HTML5 Data Adapter for SAS project as follows:

> *H54S is a library that facilitates and manages seamless bi-directional communication between a HTML5 (JavaScript) based Web Application and back-end data services written in SAS and deployed on the SAS Enterprise BI platform. It lets Web Programmers and SAS Developers collaborate easily to build general purpose Web Applications with unprecedented speed and agility.*

Translating some of that marketing speak: what this library does is act as a wrapper to native browser AJAX functions, providing a 'native' interface to both front-end HTML5 programmers and back-end SAS programmers wishing to take advantage of SAS as a readily available, pre-integrated Enterprise Application Development Platform.

For those that prefer pictures, where the library fits can also be explained diagrammatically as so:



The codebase itself is split in two parts: a JavaScript library that runs on the client and facilitates the sending and receiving of data, and a set of SAS macros that respond to its requests and handle the data that is sent.

Using this library, JavaScript developers can use a simple, familiar interface to asynchronously send data structures known as *object arrays* (essentially tables) back to SAS from within their apps. SAS-side, these are deserialised into *SAS datasets* by the aforementioned macros inside a Stored Process. SAS developers are then able to write native SAS code that uses those input datasets (as either control tables or actual data input) to service the application requests, returning SAS datasets (again, either control tables or actual data) back to the application front end as output. The datasets are returned by SAS as the output of a Stored Process, and eventually deserialised by the library's JavaScript counterpart to appear as object arrays at the front-end.

These *datasets* are intended to replace the macro-variable parameters method of communication used by the SAS Prompting Framework. Things like serialisation, deserialisation, type conversion, exception handling and response validation are all handled by the adapter. Enforcing *datasets* as a primary interfacing format is also intended to eliminate distractions and allow the programmers to focus on rapid development without worrying about the complex interface specifications mandated by some other server-side technologies. Here, we're simply saying 'these datasets in, these datasets out' - a concept hopefully familiar to most Data Management professionals.

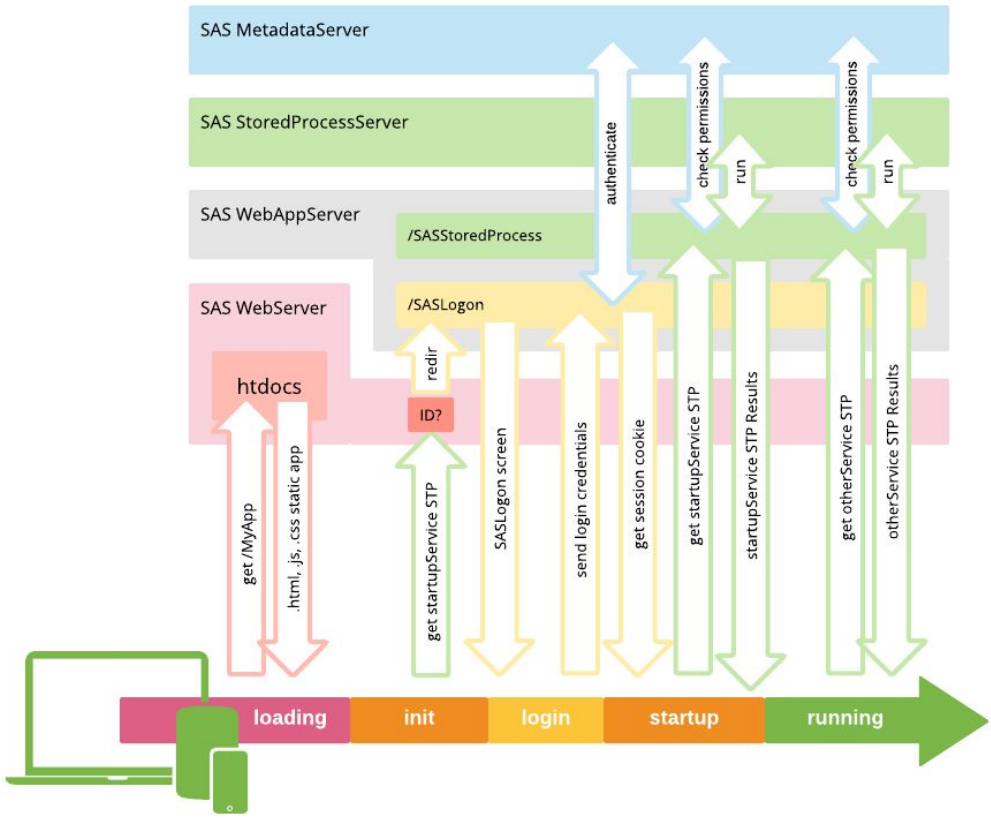
So, how does it work? To make their application contact SAS when it needs something from the server, the front-end JS programmer would write something like the following code:

```
// Instantiate adapter
var adapter = new h54s({hostUrl: 'http://myServer:8080/'});

// some data to send SAS (array of objects)
var myFirstTable = [
  { name: 'Allan', sex: 'M', weight: 101.1 },
  { name: 'Abdul', sex: 'M', weight: 133.7 }
];

// Instantiate a h54s tables object and attach object array to it
var tables = new h54s.Tables(myFirstTable, 'datain');

// make the call to a SAS STP called myFirstService
adapter.call('/Apps/myFirstService', tables, function(err, res) {
  if(err) {
    // if there was an error object returned, handle it here
    console.log(err);
  } else {
    // do what is needed with the tables returned by SAS
    console.log(res.myReturnedTable);
  }
});
```

This code spins up an instance of the adapter, creates a 'tables' object to which one or more object arrays can be attached (from here on also referred to as *datasets*), and using the call() function makes the AJAX call to the named SAS Stored Process. To handle this data, the SAS programmer would write the following code:

```
%include '/pub/sasautos/h54s.sas';

Receive dataset from the client ;
%hfsGetDataset(datain,work.additions);

* Do something with that dataset. Standard SAS Merge and Sort used as an example here;
data mydata;
  set sashelp.class (obs=3 keep=name sex weight) work.additions;
run;

proc sort data=mydata;
  by name;
run;

* Return the output of the above SAS code to the client ;
%hfsHeader;
  %hfsOutDataset(processed, work, myData);
%hfsFooter;
```

The background processing that this library handles actually looks something like this:



The order of events outlined by the above diagram can also be described as:

1. A user requests the app and its .html, .js and .css files are loaded from the server
2. The app *runs.* As it runs, it makes its initial call to SAS (a Stored Process typically called startupService) to get the data it needs before it is ready to be used.
3. If a user has not logged in or their SAS session has expired, that request to the SPWA is redirected to the SASLogon application (just as if a user had attempted to run a Stored Process without being logged in).
4. The app prompts the user for their SAS username and password.
5. This is sent back to the SASLogon application, and upon successful authentication the a new Session ID cookie is returned.
6. The adapter tries to load that initial call again. SAS now checks whether the authenticated user has sufficient Metadata permissions to access the STP that was requested. If so, the STP will be permitted to run. The STP itself may then use some Metadata based authorisation sub-mechanism, such as Metadata-bound libraries or Metadata DATA step functions.
7. The output of the startupService STP is returned to the application front end. Usernames, dropdowns, and other data-driven components are initialized and the user begins to use the app.
8. As the user interacts with the application, steps 6 & 7 above are repeated, reverting to 3 if a user allows their session to expire.

The SAS Stored Process Web Application itself was initially built by SAS with Application Developers in mind (well ahead of its time). As such has many useful features intended for debugging and optimisation. The H54S library abstracts some of these features, making them available as callbacks in the JavaScript

frontend; this is done in order to make the developed applications as user-friendly and 'supportable' as possible.

Session expiry and capture of SAS Logon redirects

With traditional 'linked Stored-Process' based SAS applications, if a user spent too much time entering data into a page of their application, the next time they interact with the the server their request would be redirected to the SASLogon application for re-authentication, in the process losing the data their app was sending to the server, and therefore the context of what they were doing.

The adapter provides a way of mitigating this problem by interrupting background redirects while keeping a cached copy of the data request initially sent to the server. The `call()` method will return an `err` object, with a type property explaining that `err.type = 'notLoggedinError'`. This allows for the session timeout to be handled by the front-end programmer, for example with a modal Login window that prompts the user to re-enter their username and password without losing their application context. The programmer can then pass those credentials to an `adapter.login()` method, which logs the user back into SAS, resuming the request queue and the normal operation of the application.

Application Logs and User Messages

The Adapter provides facility for the SAS user to maintain a client-side application log for each user (to help support large userbases in production), as well as a way for them to communicate ad-hoc messages such as imminent maintenance outages to the application front end alongside the standard data transmission.

To log application activity to the client-side log, a SAS macro variable named `&logmessage` is simply set from within the code of the SAS Stored Process. This is then added to a client-side log, the retrieval of which can be programmed into the application front-end (under a 'support' or 'debug' menu). The log can be loaded into a displayable object by calling the `adapter.getApplicationLogs()` Javascript function.

By setting a macro variable named `&usermessage` in a similar fashion, a message is transmitted to the front end (again alongside the data that was requested). In the event that it is set, the application can then display that ad-hoc message, within a modal popup alert for example. The usermessage is plaintext, and developers are able to agree conventions on the format of different types of user message, to be handled differently by the front end application (ie. whether informational or important).

Debug mode

The SPWA allows programs to be run in *debug mode*, displaying the SAS logs alongside standard program output. The H54S library abstracts this functionality, allowing the programmers to implement a 'debug mode' into their application front-end. This is intended to make productionised applications both easier to develop and easier to remotely support than using a server-side logging model.

Debug mode within an instance of the adapter is enabled either at instantiation, or by calling `adapter.setDebugMode()`. Similarly to the getApplicationLogs() call above, an array of logs for all calls made to the Server is then accessed by calling `adapter.getDebugData()`.

More documentation on this can be found on GitHub. For examples of what an implementation of this can look like, it is worth looking at some of the demo projects available on GitHub, such as the H54S Angular Seed App.

Security Considerations and Role-based Access

A useful benefit of this approach is that, by default, any communication between the application front-end and server back-end is done via secured (HTTPS) channels, to a standard that has otherwise been approved for the secure transmission of sensitive reporting data.

However, one of the greatest benefits here is the ready availability of the in-built SAS Metadata Security layer. Every time an instance of an app makes a call to the SPWA, SAS ensures that that user has *current* SAS Metadata permissions to access the STP service that was requested. This not only means that access to SAS-based applications can be entirely managed using Groups via SAS Management Console, but that, by segregating services into sub-directories with differing Metadata permissions, developers are able to separate functionality within their Apps into different roles - again, managed simply by using SAS Metadata Group permissions.

One of the purposes of the starupService convention mentioned above is to communicate *available functionality* to the application front-end, so that portions of the interface can be disabled (improving UX). This can be done by grouping the functionality for different roles within the application into respective Metadata subfolders, and then creating a 'Permissions' table at startup using the following SAS Data Step-based Metadata lookup:

```
data permissionsOutput(drop=id type);
    length functionPermitted 8.;
    length id $20;
    length type metadataRoot $256;
    metadataRoot="&_METAFOLDER";
    id=""; type="";
    set listOfRoles;
    functionPermitted=max(0,metadata_pathobj("",metadataRoot,"",type,id));
run;
```

This example code should iterate through a table named `listOfRoles` and validate whether the user currently logged onto SAS has *readMetadata* permissions on any of the subfolders. A `functionPermitted` column is added to the input table, with a value of 1 for available functionality and a 0 for anything that is not currently permitted. This table can be sent to the front end upon application startup, allowing any unavailable functionality to be disabled by the front end prior to any user interaction.

This is a simple, SAS standards-compliant way of solving the often complex problem of managing role-based access to enterprise applications. It allows both top-level access and sub-application role-based access to be managed entirely from SAS Management Console using a standard SAS Metadata Security based approach.

Up-front Sizing, Scaling and Optimisation

A benefit of developing for an already existing SAS environment is the ability to perform up-front testing of infrastructure capacity before any development takes place. This enables developers to build applications to capacity while planning for infrastructure upgrades where required.

Standard industry load-testing tools like LoadRunner or the better, *free-er* JMeter can be used to perform tests on existing SAS infrastructure using 'representative' back-end SAS services. Performance

monitoring tools such as ESM can then be used to understand the division of workload between the SAS JVM, Stored Process Multibridge sessions, SAS Metadata Server and SAS Web Server, and how that fits within the workload currently handled by that SAS environment.

As a general rule of thumb, applications that make small requests at very high frequency will place the most load on the Metadata server; in these situations the Metadata cluster may need to be scaled for optimal performance. Computationally intensive applications that require the spawner to queue requests will require the number of available STP Multibridge sessions to be increased (although these defaults should always be increased regardless). Performance can also be improved by assigning a RAM-based SASWORK location to a dedicated Stored Process Server context, although on many modern operating systems this provides little benefit over standard kernel-managed filesystem caching. Some high data volume requests can also put pressure on the SAS Web Application Server JVMs, which can be tuned for performance and availability using standard clustering techniques or increases in JVM heap and thread pool size.

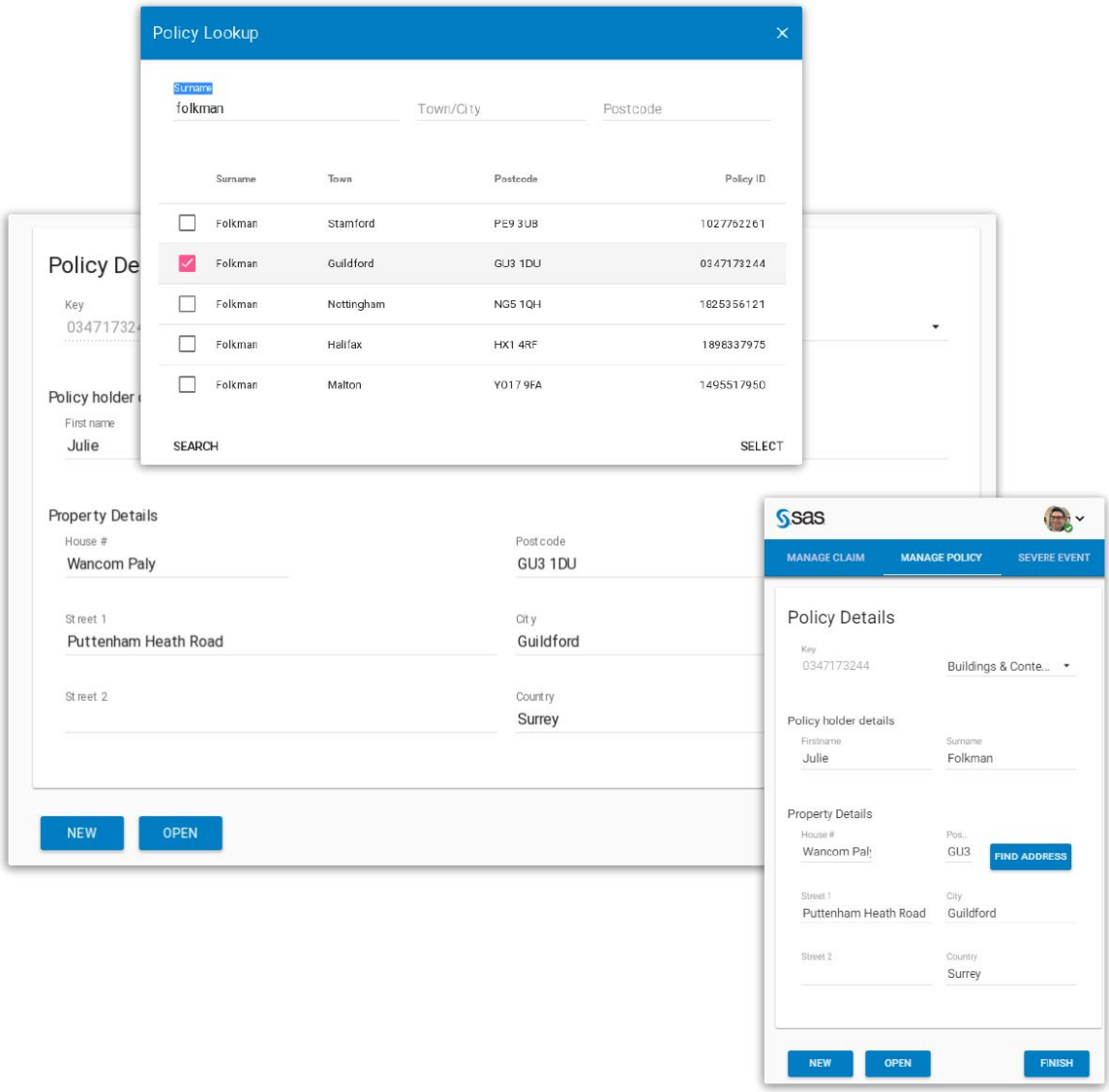
In any case, as is the case with productionising any SAS-based applications, a measurement-based approach to environment scaling is recommended for predictable performance and stability. Although scaling, sizing and optimisation are briefly mentioned here, this discussion is outside the scope of this paper; as a general rule of thumb, most modern single-node environments with 16 CPU cores should easily be able to handle 120+ simple service requests per second, which is enough to support a few thousand concurrent users for most production applications.

[empty space]

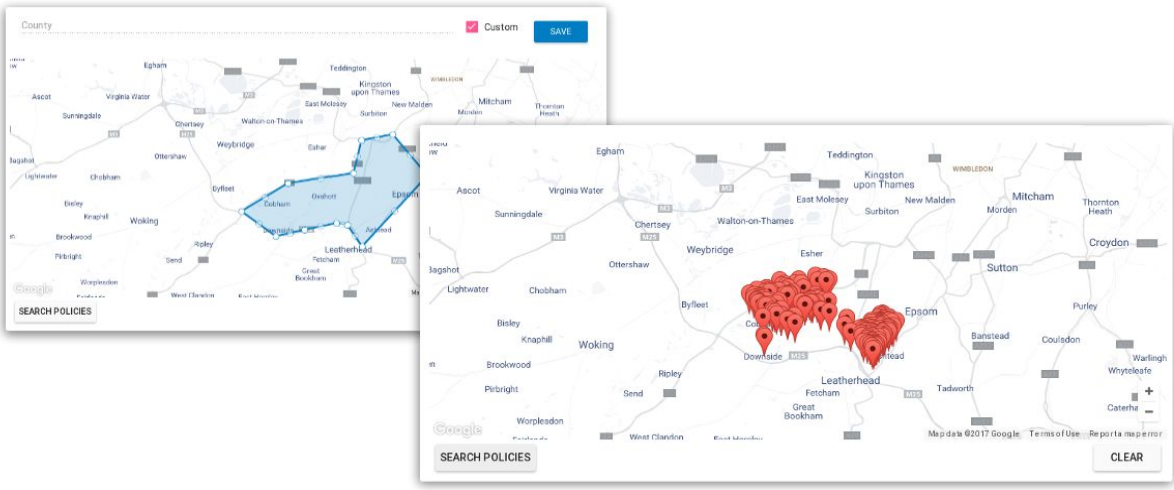
SAMPLE APPLICATION

Over the last 3-4 years, the techniques mentioned in this paper have been successfully used to develop a multitude of sophisticated Enterprise Applications, ranging from crucial operational systems supporting thousands of concurrent users, to small reporting applications, single-use-case mobile apps and lookup forms.

Many more examples of these will be shown in the presentation counterpart to this paper, a recording which should eventually be available online; however, the following are some screenshots of an example 'Insurance App' demonstrating functionality enabled by the interfacing of SAS with other readily available Open-Source technologies.



A 'Policy Search' page constructed in HTML5: AngularJS, Angular Material, H54S, SAS Application Layer and PostgreSQL persistence layer. Desktop and Mobile



A 'Geographical Search' of the same policies: In addition to the above, using Google Maps and the open source PostGIS Spatial and Geographic extension to PostgreSQL

Although it is not shown in its entirety, the demonstration system shown in these screenshots took around a week of front-end and SAS developer time to construct. While it is not yet production release quality, it does provide a basic, working system which can already operate securely across a broad swath of platforms and devices. The agility of this approach means that stakeholders were reviewing basic UI, securely populated with data, on the afternoon of the second day.

The examples shown above are a very, very small subset of what is possible when SAS is used as a Governance Layer for the readily available ecosystem of modern Open-Source Application Frameworks. In addition to this, the plethora of SAS/ACCESS engines enabling back-end connectivity and platform interoperability, alongside an abundant back-end programmer skillset.

CONCLUSION

This paper doesn't even begin to talk about the doors opened by SAS next-generation Analytics platform codenamed Viya, which is bringing features like intelligent Image Recognition, Machine Learning and easy Scaling into the Application Development mix.

The primary function of most server-side code in a deployed application, by its very nature, is Data Management. As SAS developers, we are very good at Data Management. In contrast to the proprietary 'codeless' application platforms, SAS's renewed commitment to Open Source also perfectly positions it as not only a competitor in this market, but as a readily available Enterprise Application Platform that your organisation can take advantage of today.

Without realising it, SAS is once again well ahead of its time.

Contact Information

Your comments and questions are valued and encouraged. Contact the author at:

Nikola Markovic
https://www.linkedin.com/in/nikmarkovic
Email: nik@boemskats.com
Boemska
22 Upper Ground
London, SE1 9PD
United Kingdom
+44 (0) 20 3642 4643

Relevant GitHub links:

Boemska HTML5 Data Adapter for SAS:	https://github.com/Boemska/h54s
H54S Seed Applications:	https://github.com/Boemska/h54s-angular-seed-app
(avoid UI5)	https://github.com/Boemska/h54s-sap-openui5-seed-app
(avoid extJS too)	https://github.com/Boemska/h54s-extjs-toast-skeleton
Sample Data Editor App:	https://github.com/Boemska/sas-hot-editor
GitHub Repository for Insurance example:	https://github.com/Boemska/sassurance