Jessica D. Moore

Vlad Oprica

Dave Gallerizzo

Jessica D. Moore
Fig Leaf Software,
1400 16th Street NW, Suite 450,
Washington, DC 20036,
USA

Tel: +1 202 810 6418;
E-mail:
jmoore@figleaf.com

Vlad Oprica
Fig Leaf Software,
1400 16th Street NW, Suite 450,
Washington, DC 20036,
USA

Tel: +1 202 797 7711;
E-mail:
voprica@figleaf.com

Dave Gallerizzo
Fig Leaf Software,
1400 16th Street NW, Suite 450,
Washington, DC 20036,
USA

Tel: +1 202 230 8922;
E-mail:
dgallerizzo@figleaf.com

Why single sign-on is so expensive and what you can do to reduce costs

Received (in revised form): 9th February, 2017

Jessica D. Moore

is an Engagement Manager at Fig Leaf Software, with 18 years of expertise in user experience design and a focus on usability and accessibility. She has presented at conferences for the Usability Professionals' Association and ASIS&T, and at Washington D.C.'s Mobile UX Camp. She has served as a product owner on several agile development teams tasked with implementing SSO and data integrations, across different CMS and third-party systems. She has an MFA in Art and Visual Technology from George Mason University, and a BA in English from Grinnell College. When she's not resolving issues with SSO integrations or helping design websites, she works at painting and writing, and enjoys sunsets from her canoe.

Vlad Oprica

is a Lead Architect and Senior Developer at Fig Leaf Software. He has worked as a developer for nine years, during which he built numerous sites using several programming frameworks, primarily ASP.NET and JavaScript. He also has a strong background in front-end development and design, using HTML, CSS and Photoshop. He has extensive experience with content management systems such as Ektron, Episerver and Sitefinity, and has delivered highly customised website functionality and features to many clients over the years. He is also well-versed in third-party data integrations involving AMS and CRM systems, such as NetForum, Membersuite and Microsoft Dynamics. During his tenure at Fig Leaf Software, he also acquired certifications in SharePoint administration and Agile Scrum methodology. In his spare time, he enjoys keeping up with the latest web technologies, working on creative visual designs, learning foreign languages and playing tennis.

Dave Gallerizzo

is the CEO of Fig Leaf Software, a leading digital agency with an international customer base. He also serves as a member of the Board of Directors. His prior responsibility was as Vice President of Fig Leaf Software's Consulting Services division. This division was responsible for all aspects of the company's services-based practices. During his tenure as head of Consulting Services, prior to his assumption of the position of CEO, the division saw yearly substantial growth over a seven-year period. He maintains enterprise certifications on the Drupal, Adobe Coldfusion, Amazon AWS, Google Apps, Google Maps and Google Search Appliance platforms and continues to teach a variety of technical classes on a regular basis, including: Acquia Site Building with Drupal, Acquia Drupal Layout and Theming, PHP for Drupal Developers, Acquia Drupal Module Development, Advanced ColdFusion Development, Administering ColdFusion Servers, Google Apps Deployment, Google Search Appliance Deployment, jQuery and Developing for the CommonSpot CMS platform. His academic credentials include a Bachelors of Science in Computer Science from the University of Maryland, with minors in Economics and Mathematics. He retired with 24 years of service from the United States Marines Corps, reserve component, in 2011. During his tenure of service he held enlisted and officer ranks, in both the Light Armored Reconnaissance and Combat Engineer fields, and obtained a final rank of Chief Warrant Officer 2.

Abstract

Implementing SSO is often expensive, and depending on your technical prowess, it may be hard to understand why. In this paper, we'll explain the basic technical challenge of SSO, the differences between a simple and a complex integration, and how to achieve what your business needs with the least cost and the fewest headaches. We'll focus on issues around developing custom single sign-on solutions for public websites and intranets, which presents different challenges from other scenarios such as SSO for internal business operations. We discuss the intersection of security and analytics with SSO. Finally, we offer useful tips derived from our experience with SSO integration, which you can use as a checklist to guide you through a successful SSO project.

Keywords

SSO, single sign-on, integration, login, authentication, cost

INTRODUCTION

Broadly speaking, single sign-on (SSO) is a means for simplifying authentication for end users. Both website visitors and website owners want SSO for their websites. SSO offers a simpler user experience: for customers, there are fewer usernames and passwords to remember and maintain, and there are less 'walls' requiring a login. For businesses, this translates into higher conversion ratios for important transactions, and an enhanced experience of the online brand. It can also reduce costs for customer support, for lost passwords and confusion over multiple logins. Since few businesses have the skills they need in-house to develop an SSO solution, most turn to third-party experts and consultants to accomplish the task, and the cost can be surprising.

To control costs with SSO implementations, it's important to look at a few key factors: the method of implementation, which systems will be integrated, which application programming interfaces (APIs) will support the integration, the architectural approach, and what functionality is expected from the integration. Each of these factors has a different impact on the technical approach, the skills required, the management overhead, and the overall complexity of the project. So, carefully planning for these factors will help you understand and control project costs.

METHODS OF IMPLEMENTATION

Let's begin by exploring options for implementing single sign-on (SSO). When a visitor logs in to a website with SSO, one system—the authority system—validates the login information, and then creates a logged-in session for one or more additional systems, at the same time. This can be accomplished in a few different ways, including by using a token, an encrypted password, or a federated authority (including Active Directory authentication). The method of implementation will impact your cost, and individual methods may or may not be appropriate to your business needs.

Tokens

When SSO is accomplished using a token, the authority system creates a randomly generated key that expires in a short period of time. During that time, the key can be used to authenticate into a system. Any request that comes in with this token is compared with the authority system's token of record. If it matches, then the system recognises the user and authenticates them.

Importantly, the key resides with the login authority: it's saved there temporarily, and it's also sent with the end user's login request to any additional systems that are part of the SSO integration. The authority system may do this distribution itself, or one of the other involved systems—such as a CMS—might be responsible for distributing the token for the authority.

Part of implementing SSO in this situation is negotiating the means of token distribution, the order in which the token should be distributed among multiple systems, and the duration of time for which the token is valid.

Encrypted passwords

In the encrypted passwords method, after a successful login, the authority system returns the login information as a cookie, and then moves the user's browser through each of the involved systems, so that a similar cookie is generated for each of the websites in the SSO system. To the user, this usually looks like a short delay, like loading a new page, before they see login confirmation.

If all systems will allow this method, using encrypted passwords can be a less expensive way to implement SSO. This is less secure than implementations using the token method, since the process is often visible in the URL (since query strings are used to sequence login through each system); and if the cookie is intercepted during transmission or from the user's local machine, the credentials can be reverse engineered. What's more, some vendors make the mistake of not encrypting the password, which remains a security threat even if the information is sent over HTTPS.

Federated authorities

SSO can also be accomplished using a federated authority, which is an officially recognised login authority, such as Facebook, LinkedIn, Gmail, or Active Directory (within a Windows network environment). Simply using one of these authorities to authenticate your visitors is considered an SSO integration since there are multiple systems involved in creating the authentication—but additional systems may be tied together using this method, as well.

In SSO logins using a federated authority, visitors are forwarded to the third-party authority site to login. After a successful login request, the authority sends a token back to the requesting site, and to any other systems that are part of the SSO integration. In effect, this scenario is similar to a token implementation, except that only the federated authority stores the user's password, which leaves responsibility for security around the login to that party.

Some federated authorities may not be appropriate for your business, based on many factors, including your visitors' expectations and perceptions around privacy. Others, such as Active Directory, will only work to support internal business needs—such as authenticating employees for intranets, authenticating management of content within the CMS, or other organisation-wide uses.

THE CHALLENGE OF INTEGRATING DIFFERENT SYSTEMS

Regardless of the method that you choose for your SSO implementation, the challenge of SSO lies in integrating different systems, which means coordinating between systems with different architecture and logic. Content Management Systems (CMS), Association Management Software (AMS) and Customer Relationship Management (CRM) systems are often built on different platforms and programming languages (Microsoft.NET, Java, PHP, ColdFusion, JavaScript and so forth). As such, there are a large number of possible combinations of these systems, and each implementation will present unique issues.

For example, many popular AMSs (such as NetForum or iMIS) are built on the .NET framework, which makes them ideal for integration with other .NET CMSs (such as Episerver or Sitefinity). Integration is, of course, possible with other CMS platforms (such as Drupal or WordPress, which are built on PHP; CommonSpot, which is built on Coldfusion; or Adobe Experience Manager, which is Java based). In the absence of pre-built custom modules, development of cross-platform integrations may be more expensive.

The rise of standards-based communication between systems (XML and JSON data objects) has begun to minimise or eliminate complexities arising from cross-platform communication, since most web services tend to use platform-independent data standards. If your CMS or AMS is built on an older platform, or if the web services API of either system is not well developed, you will still encounter these cross-platform issues.

For the purposes of your SSO effort, this means that your development team needs to have capabilities in the platforms and programming languages involved for *each system* that will be part of the integration. For example, if your AMS is the .NET-based NetForum and your CMS is the PHP-based Drupal, then your SSO team needs to have experience with .Net, PHP, NetForum and Drupal. If your internal team or your primary vendor does not have this breadth of experience, then you will need to find a vendor with resources that complement those of your existing team.

Ideally, at least some members of your resulting team should have experience *across both systems*, as this minimises the risk of communication problems between two teams with different expertise. If that is not possible, you can also try to build and mediate a collaboration between two separate teams with complementary expertise. Members of the team should have the skills necessary to accomplish two important tasks: (1) communicate the login credentials between each of the involved systems, and (2) customise each system to store and reflect the logged-in state, in both the database and the user interface.

If you're in a position to select the systems you plan to integrate through SSO, your implementation will be simpler if the systems use the same platform and/or programming language.

THE ADDED COMPLICATION OF APPLICATION PROGRAMMING INTERFACES

Beyond the platform and language that it is built on, each software application has a unique architecture, and the application programming interface (API) that interacts with this architecture is different. An API is a collection of queries and functions that makes it possible for developers to retrieve information, initiate a process, verify the results of a request, and otherwise interact with a system. Each API is different, so what can be done in one system might not be possible in another, making communication between systems challenging. So, even if your developers know a platform and a programming language, they can still be limited by the depth of their understanding of the APIs involved in your SSO integration.

Also, while some are more robust than others, APIs are limited and may not contain the necessary features to perform particular tasks. Most of the time, developers are limited to what the API provides: they cannot create their own queries from scratch. This means that in order to achieve a particular business need, if your needs are not supported by the API out of the box, then developers might need to come up with a creative, custom solution. This can be expensive, as it may require developing and testing several solutions.

If you're in a position to select the systems you plan to integrate through SSO, your implementation will be less expensive if each of the systems has a robust web services API available for developers. To execute a basic SSO, the API for your software products must be able, at a minimum, to do the following:

- Retrieve an existing user, based on either username or ID,
- Create a new user, based on name, email and password,
- Log a user in and out programmatically (without the user making a physical click), and
- Keep track of the user's authenticated state, as they move between pages.

The vendor for each of your systems should provide all available API documentation, including a list and explanation of all the methods, data objects and service endpoints available to developers. This is especially critical for your AMS and CRM software. The information must be available early in the development cycle, to allow developers to properly plan and architect the SSO solution.

Similarly, you will want to ensure that your team has deep knowledge of the APIs involved in your integration and appropriate support for when they encounter knowledge gaps or unexpected obstacles. If your team doesn't have expert knowledge in each of the products you are trying to integrate—which is common and likely—then you will need to find a vendor whose skills complement your team's. Otherwise, you may encounter significant problems trying to get the two systems to work together.

You'll need experts on both systems who have experience implementing SSO, and who are willing to work together. If you have multiple teams or vendors involved in your efforts, there is a significant risk that each team will define its responsibilities more narrowly than you'd expect. As a result, there will likely be a 'grey area' of functionality in the middle that each team assumes the other one will take care of but in the end neither one will. To avoid this situation ensure that the team leads clarify expectations and assign responsibility for the work at both the project level and the individual task level. Wherever necessary make sure to step in, mediate and facilitate a collaborative approach between the multiple participants and create an environment where they can succeed together rather than becoming adversarial.

ARCHITECTING A SOLUTION

The architecture for your solution will be developed based on the desired user experience and implementation method, combined with the specific features and limitations of the APIs involved in your integration. The other important factor is deciding which system should be your login authority.

In general, the login authority should be whichever system is best suited to handle user and member records, as well as sales and transaction history. Most often, this is your AMS or CRM. In contrast, an application like a CMS would be less suitable, since it is focused on content delivery and not on your audience. If your AMS or CRM is at the centre of your SSO solution, it will be easier to communicate information about members and transactions between each system in the integration, and you will spend less time developing workarounds and customisations to accomplish these tasks.

With this in mind, map out all of the systems that need to be integrated. Select which system should host the login and account creation screens, in what order the systems should authenticate, and where users should be returned after a successful login. Similarly, describe the logout sequence, and where users should be returned after logout. Determine session duration and the methods for maintaining a session between each system, and describe error handling for each system if authentication fails.

Describing all of these things in detail creates a shared understanding of the SSO development plan for your team, and forms the basis of documentation for your implementation. Your development team will advise on additional points to cover in the architecture, based on your requirements.

'SIMPLE' SSO AND MORE COMPLEX INTEGRATIONS

The complexity of your SSO project will, of course, influence the cost. In a simple SSO implementation, two applications, such as an AMS and a CMS, need to have a shared authentication channel. That is, when a user clicks the login button in either one of the applications and submits their username and password, they will automatically be logged into both applications. At this point, the user can perform tasks with both applications, since they are authenticated into each. The issues that we have discussed so far are the main challenges with a simple SSO implementation.

SSO implementations can become more complex in many ways, and sometimes multiple factors are involved. Most of the time, your business goals will extend beyond 'simple' SSO in at least one of the following ways. You may be integrating more than two systems, using different login or logout methods for specific systems, integrating data and services between the systems, or using non-standard session lengths or authentication rules.

More than two systems

With each additional system that you integrate, the SSO implementation becomes more complex, because the idiosyncrasies of each system and its API compound upon the others. Also, with more than two systems involved, most likely your development team will incorporate experts from multiple vendors. With this larger team comes a greater need for clear and open communications. It also complicates scheduling, as it becomes more important to synchronise work efforts. The integration components for each application should develop in parallel, so that issues are discovered and mitigated early in the project for each vendor. Otherwise, when issues arise, one team may be too deeply committed to their solution, which can impact the broader integration.

As your business grows and changes, you may find that you need to add more systems to an existing SSO implementation, or you may need to change one of the systems. If the new system is substantially different from what was previously integrated, this can require significant adjustments and revisions to the existing solution. Sometimes the developers of the original solution are no longer available to answer questions, which can introduce perplexing problems, and can mean that the entire integration needs to be reworked. For this reason, we encourage clients to document the methods and processes used in an SSO implementation, along with issues encountered and their resolutions, for reference in future development efforts.

Different login or logout methods

For most businesses, one goal of SSO is to unify the login and logout screens. In implementations with several systems involved, it's possible for one of the systems to be significantly limited or out-of-step with the others. Most often, this happens when a new application is added to the integration after the initial SSO implementation; and it's more likely to happen if the team working on the new integration does not have complete documentation on the existing implementation, and early access to the broader SSO team.

If one of your systems requires a separate login screen from the main SSO login shared by other systems, then there should be special planning to address this in the architecture, so that you can avoid a disjointed login experience—where users are logged into some systems but not in others. The planning will involve new triggers in the broader system to synchronise all the entry and exit points so that clicking login or logout from any screen in the system works the same way.

Data and services integration

Companies often implement SSO as part of achieving a bigger goal, such as unifying an interface for registration or sales or personalising the user interface. Making this happen requires more robust systems and APIs than simple SSO, as well as a greater understanding of the APIs and the systems involved. Common examples of data and services integrations include using your CMS to mimic AMS and CRM functionality, such as:

- Registering, joining or renewing
- Managing an account profile or password
- Creating a shopping cart
- Payment processing
- Displaying purchase history

Similarly, businesses often want to leverage data stored within an AMS or CRM to customise the website experience, personalise content delivery or determine roles or access to content. Examples of such data include all profile fields and customer data beyond the customer ID, such as:

- Name, address or profession
- Membership or registration type and status
- Sales funnel flags
- Purchase history
- Group or list identifiers

If the AMS or CRM provides a front-end interface for your visitors, then using these interfaces will minimise your costs. Otherwise, you will be re-creating interfaces that already exist, and that are designed to work seamlessly with the application. Using these interfaces may not be practical if they are difficult to use, inaccessible, hard to customise or otherwise lack features and functionality that you want to offer.

If you want your visitors to use one system to accomplish tasks that are driven by another system, you need a data and services integration. In such an implementation, SSO is the starting point for your project, but it will not accomplish your goal. Development will need to continue to retrieve and communicate the necessary data, and further, to update the data stored in each system as a result of various transactions.

That is, in this situation, you're not processing just a login but individual transactions as well. When someone makes a purchase, the authority system has to be notified, has to log and validate the purchase, and has to return the results of those operations. It may also need to send along information that the user needs to see and the companion system needs in order to continue providing appropriate information, such as compounding discounts, showing related purchases or preventing certain items from being purchased according to business rules.

A data and services integration is attractive because it keeps your website visitor within the same ecosystem: the visitor does not have to move between two or more different systems to accomplish their goals. It increases development costs astronomically since the website system has to act as a one-stop shop, mimicking—indeed, almost replicating—the features and functionality of another system.

In effect, each of these additional transactions beyond login requires the same attention to detail as the primary login transaction. This compounds the sets of data that have to be communicated and stored within each systems, and it likewise increases the burden of keeping this information in sync between each system.

When setting a budget and defining requirements for an integration, make sure you decide whether the integration should only involve SSO, or whether it should also involve data and services integration. As described earlier, the data and services integration may be much more expensive. Each additional layer of data and services increases the cost. The ROI may be justified, but be aware of how your business needs will impact the cost.

Non-standard session or authentication rules

With security in mind, many applications are designed to reflect standards for session duration and authentication. For example, most applications will log a user out after 20 minutes of inactivity, and most will prevent simultaneous logins for a single user. If your business needs require something different, particularly if your requirements are not available out-of-the-box for all of the applications involved in your SSO integration, then your developers will need to find a creative solution to the problem that works for all of the involved applications.

It's important to identify and discuss your requirements for session duration, simultaneous sessions, 'remember me' functionality and similar issues early in the process, so that your development team can identify whether customisations are required for any of the systems involved.

COMPLEX IMPLEMENTATION: AN EXAMPLE OF THE ISSUES

In a recent SSO project that we worked on at Fig Leaf, our client wanted to ensure that purchases within the AMS would be reflected in the user profile in the CMS without requiring a login, so that content access restrictions handled by the CMS would update at the same time—effectively allowing the visitor to access the content that they had just purchased. Importantly, the client desired for this process to happen without requiring the visitor to logout and login again. At this point, you'll recognise that this is a data and services integration, built with an SSO implementation.

This request makes a lot of sense from a user experience perspective. When any of us makes an online purchase through Amazon, for example, we expect that this purchase will appear in our purchase history immediately; and if we have purchased online content, we expect for it to be delivered to us as a result of our purchase. In most situations where we experience this process as visitors, we are usually visiting e-commerce websites, where the content presentation and the purchasing are executed on the same software.

In contrast, the client making this request is a membership association, and the purpose of their website is twofold: sell and renew memberships and deliver content. They rely on two separate software applications to meet these needs. To accomplish sales, they used their AMS. To accomplish content delivery, they used their CMS, which offers a more powerful publishing interface that includes personalised marketing and role-based content delivery. Using a single application for both purposes was not an option for our client. So, these two systems needed to be integrated to reach the client's goals.

Since the client had been using the AMS for substantially longer than the CMS, and there were several other business processes hooked into it, the software had been customised over the years. Knowledge about these customisations was dispersed over several teams of consultants and internal team members, some of whom were no longer available, and documentation was sparse. As a result of this, it was important that the AMS remain the interface for both authentication and purchasing, as well as the authority for all member data. So, we worked with the current team of AMS consultants to ensure that after each update to a member record—whether initiated by an online transaction or a manual record update—the AMS would ping the CMS to retrieve the updated member record.

Initially, although the CMS received the ping to retrieve the record, records were not being updated. After a diagnostic working session with the AMS team, we discovered that the ping to us was being delivered too soon in their internal business logic. By postponing the ping, this issue was resolved.

Then we discovered a bigger problem: with the ping to update, we could retrieve the member record and all of the relevant fields, and update the corresponding user record on our end. User access was not affected by this update. We could see that the roles and permissions were being updated for the user *in the database*, but the user was still unable to access their newly purchased content *on the front-end*. To actually update content access, the user had to logout and login again, or wait for a seemingly arbitrary period of time for the changes to apply. Recompiling the website would also resolve the problem, but was clearly an undesirable solution due to the resulting period of system downtime.

At this point, despite our advanced knowledge of the CMS, we had to directly consult the CMS support team. In discussions with top tier support, we learned two things. First, the disconnect that we were witnessing between user roles and permissions, and actual user access to content, was caused by a basic architecture principle of the CMS: at login and at site compilation, the CMS effectively builds an individualised sitemap of content for each user, showing what they may or may not access. Secondly, the use case that we were trying to solve was specifically not supported by the CMS. In fact, the CMS had been architected to prevent this use case, as part of its security measures.

Despite this, in the interest of the client, the CMS support team worked with us on several possible approaches to resolve this issue. Together, our two teams explored several different solutions to the problem until we found one that worked in all of our test scenarios in the staging environment. There was a great deal of time and effort involved in architecting a solution. For example, at one point, the CMS support team shared with us an API call designed for diagnostic purposes only. When we realised the effect of the API call, we discovered that we could embed this call during our member record retrieval process, and it would produce the desired mid-session update of user content-access sitemaps. The call produced a sitemap for each user in the CMS database, so it had the potential to bog down the system in significant ways. When we discussed our idea and our concerns with the CMS support team, they worked with their engineers to develop and share a variation of this API call that would create a sitemap for only one user record at a time.

While this produced a solution for the client, it required a significant investment of resources from the client and three teams of experts, as well as daily progress calls with detailed communication and documentation, several working sessions, and the development and testing of multiple possible solutions. This requirement was only one part of a larger and more complicated SSO implementation, but we share it to illustrate how seemingly simple or obvious requirements can, in fact, be deeply complex and resource-intensive.

SECURITY

As you plan your SSO solution, it is important to keep in mind a few concepts regarding your system's security. This involves not only satisfying the technical concerns your IT department might have, but also managing expectations for your members and users about what a secure login actually looks like and what restrictions they should be aware of.

From a high level, single sign-on represents a single point of entry into two or more systems. While this provides great value, ease of use and reduced headaches regarding the login process, it also represents a streamlined point of attack into your system. Essentially, the threat of compromised account credentials is magnified in proportion to the number of systems that are connected by that one login screen.

Fortunately, simply following the few techniques outlined below will go a long way towards ensuring a comfortable baseline of security for your SSO solution:

- Enforce strong passwords— require alphanumeric and symbol combinations, and educate users to not base their passphrase on obvious words.
- Limit session duration—ideally, a logged-in user should be automatically logged out after 20 minutes of inactivity.
- Encrypt passwords and sensitive data as they are communicated between sites and service endpoints—make sure your developers are using HTTPS instead of HTTP, and make sure sensitive data isn't visible in browser cookies or url parameters.
- Restrict power user roles—do not automatically grant Edit, Delete or Admin rights to any of your systems when a user or member creates an account. Grant only basic viewing permissions if possible, and only add the preceding rights manually to pre-selected, trusted users.

ANALYTICS

When you set up your SSO integration, you may also wish to set up analytics to monitor user success, and to leverage data available through complex integrations. Exactly how you implement your analytics will vary, depending on the analytics product that you use, the architecture for your implementation, and the flow that you have designed through these processes.

To track metrics on your SSO implementation, the simplest thing to do is add event tracking. Event tracking basically follows how often people click on important buttons or links. Usually, people set up event tracking for links that represent conversion: for many sites, this is a click to login, a click to create an account, or a click to purchase something. So, for example, if you set up event tracking on the login buttons on your site, you'll be able to see how often people click each login button, and where they are when they click it. You'll also be able to see if this changes over time. If authentication takes place on a system other than your website, you'll be able to see how many people leaving the site are actually leaving to *log in*. With advanced analytics tools, you can follow whether they come back to you after logging in.

Sometimes, your site may have clear paths defined for specific tasks. For example, you may want new visitors to land on the home page, see and click on a promotion for creating an account (taking them to a page talking about how great it is to have an account), and then click on the button to create an account. If you are keeping users on your site throughout the account creation process, then you can continue to track all the way through to successful account creation.

Wherever there's a clear path like this, it makes sense to add a second form of tracking, by defining a conversion funnel. You can define a conversion funnel in many analytics software packages, or using separate conversion optimisation plugins. Once this is set up, you'll be able to see if there are points in the account creation process where users drop off, so that you can optimise essential workflows.

Finally, if you have a complex data and services integration, you have the opportunity to track how different audience segments behave on your site, using advanced analytics customisation. With what you learn, you can further personalise content delivery for these audiences, and continue to improve both funnels and messaging for these groups.

THE ROUNDUP

As you can see, requirements, software and API limitations, team experience, and team communication can make or break your SSO project. Here is a short checklist to keep you on track.

Important points for businesses *planing* an SSO integration:

- When selecting the systems you plan to integrate, look for applications that use the same platform and/or programming language, and that have a robust web services API.
- For each of the applications involved, obtain all available API documentation, early in the development cycle.
- Make sure your development team has capabilities in the platforms and programming languages involved for each system that is part of the integration. Also be sure that your team has experience with implementing SSO using the applications involved in your implementation. If not, then find and hire the complementary expertise that your team needs.
- When setting a budget and defining requirements for an integration, make sure you decide whether the integration should only involve SSO, or whether it should also involve data and services integration.
- Where reasonable, using the built-in user interfaces for specialty applications like an AMS or CRM will minimise your costs.
- If you plan to phase in data and services integration, inform the team, since this may influence the architecture of your solution.
- When defining the requirements for your integration, be careful, detailed and specific. Identify which features and functionality are critical, and prioritise others, in the event that some features conflict with the options for implementing others.
- Identify which system should be your login authority, and map out the architecture of the integration, so everyone is building towards the same solution.

Critical items for business *executing* an SSO integration:

- Make sure the development teams have access to a sandbox/testing environment in which to develop the SSO solution. Avoid making any changes to the production environment until the solution has been thoroughly tested through multiple use cases, and you're certain it works as expected.
- Facilitate transparent, detailed, early and regular communication among your team—particularly among the team leads for each application. It's critical that your team is willing to work together, and that any issues in this regard are identified and mitigated early in the process.
- Make sure that the team leads clarify expectations and assign responsibility for the work at both the project level and the individual task level, particularly when multiple teams or vendors are involved in the project.
- Ensure that each team is working towards a solution together, on the

same schedule, so that the approach can be adapted to resolve issues as they are discovered. If one team is finished when the other is starting, it will be more difficult to make needed changes.
- Use your AMS or CRM as the login authority, when feasible, to facilitate development of data and services integrations.
- Thoroughly document the architecture of your SSO implementation and related data and services integrations, for future development efforts with new team members.
- Develop documentation as you work, so that if another team needs to step in, they can understand and use existing work.

CONCLUSION

SSO can be a costly and intimidating project to undertake, because there are so many variables in each solution. Knowing the most costly aspects of an SSO project, you are in a better position to prioritise your goals and make your project manageable, affordable and successful.