CHECKLIST

WEB APPLICATION CHECKLIST **NUMBER 1 OF 5**

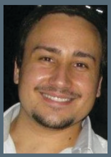
Web Application Development

BY ATA SASMAZ

✓	
○	**Log UI Errors**
	JavaScript allows exceptions to be caught and it's feasible to send them to an error logging service via ajax requests. Otherwise it's difficult to intercept UI errors in web environments.
○	**Interchangeable data layer**
	The data layer should be detachable and exchangeable with another data layer that conforms to the same contracts.
○	**Automated deployment process**
	Deployment process should be automated and project files for production environment should be generated by a deployment server and be deployed automatically without human touch.
○	**Use VCS**
	A version control system keeps a history of all code changes (thus serving as code backup) and keeps track of why certain pieces of code have been added – and hence is essential for collaboratively working on a codebase. While git is currently the most popular VCS, you might also try SVN or TFS. GitHub is the most popular VCS provider for open source projects; a free alternative is BitBucket or GitLab (for self-hosting). Microsoft has Team Foundation with extra features for collaboration.
○	**Code review**
	Nobody writes perfect code all the time, so a code reviewing system is necessary to keep code quality high. Code review also allows more than one developer to become familiar with the code – so if the original author of the code is not available, another developer can make changes easily. GitHub, GitLab and Team Foundation (among others) provide code reviewing features.
○	**Permissions and roles system**
	Every application needs permissions and roles. Application admins and user organization admins are a minimum. For other roles, a flexible, global roles system is required.
○	**Log all unhandled errors**
	All errors should be logged globally for future inspection. No error should be able to pass the global error logger.

✓	
○	**Automated Testing**
	Include as many as you can: Unit tests, End to End tests, Integration tests, Component interface tests, System tests, Performance tests, Acceptance tests. *Apply these tests to all levels of your application – especially to your JavaScript code, if you're writing a SPA.*
○	**Continuous Integration**
	Setup a continuous integration environment (e.g. with Jenkins) which, on every commit to your VCS, fetches the latest source code, builds it, executes all kind of automated tests and then deploys it to a staging environment. *Never deploy from your workstation; always deploy from your CI server.*
○	**Guidelines for developer machine configuration**
	One of the most time consuming (and pointless) problems occurs when different developers have different development environments. Let people know what should they install, which version, with which components, and how.
○	**Business layer should work in different environments**
	The code in the business layer must be generic. Even if it's targeted for web environments, it should also work in desktop, server, and mobile environments, with a different user interface and data layer, without changing any code.
○	**Define standards for coding**
	A well-defined coding standard plays an important part in the future of the project. Does every method need a comment? What are the naming conventions? Where should the sample code usages go?
○	**Use and know the tooling you already have**
	Know the tools already available, e.g., Chrome DevTools and Firebug. Use them to inspect your web application, find performance hits and other potential problems. Also check out popular productivity extensions for your browser like PostMan (or Advanced REST client), various JSON output prettifiers, etc.

ABOUT THE AUTHOR



ATA SASMAZ is a software engineer specializing in web application architectures. He blogs regularly at **www.ata.io**.

DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513
888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com
Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-77-6
ISBN-10: 1-936502-77-1
50795
9 781936 502776

Version 1.0 $7.95

CHECKLIST

WEB APPLICATION CHECKLIST **NUMBER 2 OF 5**

Web Application Performance & Reliability

BY ATA SASMAZ

✓ PERFORMANCE

○ **Use CDN**

Content Delivery Networks speed up your site by serving static files (like images, js and css) files from the location nearest to the visitor. CDNs also reduce bandwidth costs. CloudFlare is a good example.

○ **Minimize all .js and .css files**

Javascript and css files should be (a) minimized with a compressor like YUI compressor and (b) gzipped. Also put as much Javascript at the end of pages as possible.

○ **Log slow loading pages**

A web application should display extremely fast. A system to analyze and identify slow pages is mandatory. Pages that work fast enough for most users may load unexpectedly slowly for specific users; you need to figure out why.

○ **Use NoSQL for certain non-critical data**

NoSQL (document, key-value, column, graph) databases are very fast when it comes to receiving and storing certain kinds of data. In some cases, they also scale better. Although they don't have relational integrity (ACID) and it is therefore better to use relational databases for critical data, using NoSQL can save costs and can be safely used for non-critical actions like notifications, chat messages, etc. For a more detailed breakdown see Finding the Right Database for Your Use Case.

○ **Choose a datacenter geographically nearby**

The datacenter should be geographically close to most of your users. Keeping the datacenter in the same country as a user – or even better, in the same region within the country – can result in massive speed gains. Use multiple datacenters if necessary.

○ **Architect to use different data sources**

When stored data increases, the application loses performance. The application architecture should be ready to work with multiple data sources.

✓ RELIABILITY

○ **Distribute requests and aim for 100% uptime**

Instead of connecting directly to application servers, consider adding a reverse proxy to forward requests internally. This allows operational servers to continue serving while some of the servers are down.

○ **Backup data automatically**

The data should be backed up automatically – every day at least. Backups should reside on different stores than application servers – ideally in distinct data centers – to prevent catastrophic failure.

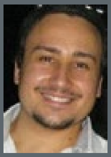
○ **100% test coverage for Business and Data layers**

All code in Business and Data layers should be completely covered by tests. Mixing up a user's data or calculating results incorrectly and storing or serving incorrect results means losing users and money very quickly.

○ **Monitor server and application uptime**

Consider using 3rd party services to monitor servers' time online. You might also implement a custom service to check the status of the servers at specified time intervals. Besides verifying that the server is functioning correctly, run automated tests on your application to make sure it works as expected.

ABOUT THE AUTHOR



ATA SASMAZ is a software engineer specializing in web application architectures. He blogs regularly at **www.ata.io**.

DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com
Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-77-6
ISBN-10: 1-936502-77-1
50795
9 781936 502776

Version 1.0 $7.95

CHECKLIST

WEB APPLICATION CHECKLIST **NUMBER 3 OF 5**

Web Application Security

BY ATA SASMAZ

✓	
○	**Isolate critical information in the DB**
	Database users should be restricted from accessing critical information, like retrieving user passwords even if they are hashed, or retrieving all of the user email addresses. Stored Procedures or Views should be used for validation purposes and for customized data.
○	**Protect from Remote Code Execution**
	Remote Code execution allows attackers to execute code when the application relies on weak code inclusions.
○	**Flood and spam protection**
	Flood and spam attacks are possible even from authenticated users. Always track the last X operations of users with their times to prevent users from making too many requests.
○	**Hash passwords with unique salts**
	All user passwords should be hashed with a salt and salts should be unique for each user. People tend to use same passwords in different services and it's the application's responsibility to protect users' passwords.
○	**Global XSS protection**
	XSS (Cross Site Scripting) lets users execute a malicious URL.
○	**Protect from SQL injection vulnerability**
	SQL Injection is a common vulnerability wherein SQL commands are manipulated as strings by the attacker, which allows harmful SQL commands to be executed. Using an ORM is one good way to be protected.
○	**Protect from CSRF**
	Cross-Site Request Forgery is a common web vulnerability which allows attackers to place an iframe in their websites and request pages from the application while the user is not in the application. To avoid, this do not allow any modification with GET requests; protect POST requests outside of application's domain; but the best solution is to provide a token in each form and validate against it.

✓	
○	**Ask for password before modifying critical information**
	Even if user is remembered on a particular computer, or even if the user successfully logged in some time ago, the password should be always requested when accessing or modifying critical data like the password itself, email address, or a backup of the data.
○	**HTTP Strict Transport Security**
	When serving pages in HTTPS, serve only in HTTPS. Otherwise a middle-man can act as a HTTPS-HTTP transformer and interpret the data by letting the user make requests in HTTP.
○	**Use HTTPS throughout the application**
	HTTPS is a world-standard encryption; there's no overhead except after the initial connection hand-shake. All pages and resources should be transferred over HTTPS. Using HTTPS also reveals referral information when the source is also HTTPS; otherwise browsers won't display it.
○	**Validate session against browser and location**
	Session cookies can be hijacked. Browser headers and user's last IP address location can be validated against the original session's. For a more aggressive solution, match sessions against IP addresses; but this is problematic on dynamic IP addresses and on mobile devices.

ABOUT THE AUTHOR

ATA SASMAZ is a software engineer specializing in web application architectures. He blogs regularly at **www.ata.io**.

DZONE RESOURCES

DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513
888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com
Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-77-6
ISBN-10: 1-936502-77-1
50795
9 781936 502776

Version 1.0 $7.95

CHECKLIST

WEB APPLICATION CHECKLIST **NUMBER 4 OF 5**

Web Application Usability

BY ATA SASMAZ

✓	
○	**Localization**
	Even if the application targets audience with a single language, your user-language-base may expand in the future. A multi-language-ready application is a great way to anticipate growth.
○	**Minimize page changes**
	Page changes are slow compared to ajax requests and also causes users to get lost switching pages. Single Page Apps (like Gmail) offer excellent user experience, but development is more difficult and bugs may occur more easily. If you have enough resources (i.e. manpower) then go for a single page app; otherwise use ajax freely.
○	**Simple, intuitive user interfaces**
	The age of "learning how to use programs" is over. Basic functionality should be easy to access without any practice. Advanced operations may be revealed after user becomes more familiar with the software. Complex interfaces scare off users and lead to user error.
○	**Global search system**
	Everyone expects search these days: Google, Facebook, and Twitter have made users expect a global search system that can be filtered after search results are served. Let your users have the same functionality they are used to.
○	**Turn off verbose errors in production**
	Every developer needs verbose error pages that output all information related to the error. But in the production environment, verbose error pages should not be served to users while the application is still able to continue logging the problems (for the developer to check later).
○	**Always take the user back or forward after an event**
	When there's an error, or a basic request like password-entry is issued, users should land where they want to go or where they came from. Always take the user either where they came from, or where they want to go.
○	**Mobile-First UI**
	The most common way of designing UIs is designing for the desktops first and then adapting the design to mobile devices. Although this does work, it often increases overhead mobile devices. The UI must be designed for mobile first and then adapted for desktops.

✓	
○	**Global feedback system**
	There will always be issues that developers and testers cannot forecast. The best way to handle this is to get user feedback with a global mechanism that can be accessed on every page.
○	**Consistent UI behavior**
	Users may be using a Windows, Mac, Linux, Mobile device (or a device not commonly known) and the UI must behave the same in every environment. The best way to achieve this is to conform to standards and never use non-standard components. Major design frameworks like Bootstrap and Foundation take care of a lot of standardization for you.
○	**Use friendly URLs**
	Although a web application is generally not focused on organic visitors (from search engines), nevertheless, when people share URLs in emails or in IMs the shared person will want to know what will be opened by clicking the link. Users explain links increasingly less, so the URLs they share should at least explain what the URL is related to.
○	**OAuth Authentication (or social logins)**
	Allow people to use their favorite OAuth provider (e.g. Google, Facebook...) for authenticating your service, rather than having to create another separate username and password.

ABOUT THE AUTHOR

ATA SASMAZ is a software engineer specializing in web application architectures. He blogs regularly at **www.ata.io**.

DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com
Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-77-6
ISBN-10: 1-936502-77-1
50795
9 781936 502776

Version 1.0 $7.95

CHECKLIST

WEB APPLICATION CHECKLIST **NUMBER 5 OF 5**

Web Application Management

BY ATA SASMAZ

✓ ANALYTICS

○ **Save all the data you can**

Every data-point, every request and event should be saved in a 'Big Data' store. These data-points will become valuable in the future – data mining will reveal unexpectedly useful information. Also think about how you're going to use the data in order to plan how you're going to save it.

○ **Observe users to discover their intentions**

Finding the real reasons why users use or don't use your application is essential for future development. Getting a clear sense of user intent is often very difficult, but doing some observation is much better than doing none.

○ **Allows users to get flexible analytical reports**

Good data analysis is more critical than ever. Analytical reports reveal where the business should go and how it might get there. A good web application does not just assist users perform specific tasks; it also generates actionable reports, customized by the user.

✓ CONVERSIONS

○ **Referral system**

Referring is one of the oldest and most effective conversion techniques for obtaining new users. A successful referral system awards the referrer and also attracts new users with offers.

○ **Support system**

Users will always have problems, so every application needs a support system. Lack of a support system will scare off users. Some external support solutions: ZenDesk, Desk, Freshdesk, Zoho Support, AnswerHub.

✓ CONVERSIONS *CONTD.*

○ Always over-deliver

Whether you have only one client or thousands: whenever anyone buys your product, always over-deliver. Providing more than expected will help offset the defects that every software product has.

○ Social integration + incentives

Chances are low that visitors or even paying users will share your app on social networks. There should be incentives for sharing, like discounts. This requires using APIs from Facebook, Twitter and other social networks.

○ Mailing list

Keeping users up to date is important. When people use a product, they like knowing if you are continuously supporting the product and making it better. Creating a mailing list and letting users know about new improvements monthly shows responsibility.

○ Know your potential customer base

Don't expect users to come to you. You have to work for it. Although there are great premium advertising solutions, it is possible to market your app on the internet without paying a dime by offering some value for free or almost free. Then you can refer users your actual, paid product.

○ Don't let customers go away

It is very important to understand why a user left your service. For example, you might email users after they leave, offer value (discounts etc.) for them to return, or just ask for their feedback.

✓ COMPETITION

○ Research users' desire to use your product

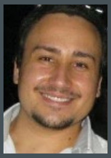
No software product starts off knowing where it will end up. Analysis and re-evaluation often guides developers and managers to somewhere they didn't quite expect. Always try to understand your customers' desires by analyzing which parts of the application users use most frequently.

✓ COMPETITION *CONTD.*

○ **Follow your competitors**

No product is 100% original. One company develops, another improves, the first one improves again – this is the circle of development in every industry. Every product has competitors; learn from them and then do better.

ABOUT THE AUTHOR



ATA SASMAZ is a software engineer specializing in web application architectures. He blogs regularly at **www.ata.io**.

DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513

888.678.0399

919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-77-6
ISBN-10: 1-936502-77-1
50795
9 781936 502776

Version 1.0 $7.95