Web Application Deployment Checklist

Development

- 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 on web environments.
- Interchangeable data layer
  - The data layer should be detachable and exchangeable with another data layer which conforms contracts.
- Automate 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 the history of all the code changes and prevents losing written codes. Also it allows working collaboratively. GitHub is the most popular VCS providers, a free alternative is BitBucket. Microsoft has Team Foundation with extra features for collaboration.
- Code reviewing
  - Nobody writes good code always, and a code reviewing system allows high-quality output from developers. It also allows more than one developer to get familiar with the code so if the author of the code is not available, the other developer can make alterations easily. GitHub and Team Foundation are some of the products providing code reviewing.

- Permissions and roles system
    - Every application needs permissions and roles. Application admins, user organization admins, and for other roles a global flexible roles system should be there.
- Log all unhandled errors
    - All the errors should be logged for future inspections globally. That is, no error should be able to pass the global error logger.
- Test Driven Development
    - Apply as much as you can from: Unit tests, End to End tests, Integration tests, Component interface tests, System tests, Acceptance tests...

- Automatic testing process
  - Before every deployment a testing server should run all the tests and deploy the application if the code-base passes all the tests, and should report to administrators otherwise.
- Guideline for developer machine configuration
  - One of the most time consuming problems during development is having different development environments between developers. Let people know what should they install, which version, with which components and how.

- Business layer should work on different environments
  - The code in business layer must be generic and even if it's targeted for web environments, it should work in a desktop, server or mobile environment with a different user interface and data layer without any requirement of code change.
- Define standards for coding
  - A well-defined coding standard play a big role for the future of the project. Does every method needs a comment? What are the naming conventions? Where should the sample code usages go?

Performance

- Use CDN
  - Content Delivery Networks speed up serving your site by serving static files like images, js and css files over the nearest location to the visitor. They also reduce bandwidth costs. CloudFlare is a good example for a CDN.
- Minimize all js and css files
  - Javascript and css files should be both minimized with a compressor like YUI compressor and gzipped when being served. Putting javascript at the end of pages is also good.
- Log slow loading pages
  - A web application should serve super-fast. A system to analyze slow pages is mandatory to identify slow pages. Pages that work fast enough may take long time to get loaded for specific users according to the data.

- Use NoSQL for non-critical data
    - NoSQL databases (document databases) are very fast when it comes to receive and store data. They also scale better. Although they don't have relational integrity and it is better to use relational databases for critical data, using NoSQL saves costs and can definitely be safely used in places like notifications, chat messages etc.
- Choose a datacenter close by
    - The datacenter location should be close to most of the users. A datacenter in the same country makes huge impact in page speeds. Use multiple datacenters if necessary.

- Allow working with different data sources
  - When stored data increases, the application loses performance. Application architecture should be ready for working with multiple data sources for scaled data from different sources.

Security

- Isolate critical information in the DB
  - Database user should be restricted to access 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 are possible even from authenticated users. Always track the last X operations of users with their times to prevent 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 application's responsibility to protect users' passwords.
- Global XSS protection
  - XSS is short for Cross Site Scripting web vulnerability which let's user execute a malicious URL.

- Protect from SQL injection vulnerability
  - SQL Injection is a common vulnerability where SQL commands are manipulated as strings by the attacker and allows harmful SQL commands to be executed. Using an ORM is a 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 requests pages from the application while user is not in the application. Not making any modification with GET requests is mandatory, and protecting POST requests outside of application's domain also has protection but providing a token in each form and validating against it the best solution.

- Ask for password before modifying critical information
    - Even if user is remembered in that computer, or even if user successfully logged in some time ago the password should be always asked when accessing or modifying critical data like password itself, email address, or getting 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 interprets the data by letting the user make requests in HTTP.

- Use HTTPS in all the application
    - HTTPS is a world-standard encryption and there's no overhead except after the initial connection hand-shake. All the pages and resources should be transferred over HTTPS. Using HTTPS also reveals referral information when the source is also HTTPS, otherwise browsers don't provide it for security reasons.
- Validate session against browser and location
    - Session cookies can be hijacked . Browser headers and user's last IP address location can be validated against original session's. An aggressive protection is matching sessions against IP addresses, but this is problematic on dynamic IP addresses and on mobile devices.

- Save every data you can
  - Every data, every request and event should be saved in a Big Data storage. Those data will become valuable in the future and data mining techniques will reveal useful reports
- Observe users to find out intentions
  - Finding the reasons of why users use or not use your application is important to plan ahead.
- Allows users to get flexible analysis reports
  - Data analysis is very critical in these times. Analysis reports reveals where and how should the business must be headed. A good web application does not just assist the users, but also generates reports according to user's desire.

Reliability

- Distribute requests and go for 100% uptime
  - Instead of getting connections directly to application servers, try to add 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 and furthermore backups should reside on distinct stores then application servers, even in distinct data centers to prevent any catastrophic failure.
- 100% test coverage for Business and Data layers
  - All of the codes in Business and Data layers should be covered with tests. Mixing up user's data or calculating wrong results and serving or storing them means losing users and losing money.

- Monitor server uptime
  - There are 3rd party services for monitoring online time of the servers. Also a custom service can be implemented for checking the status of the servers in specified time intervals.

Usability

- Localization
  - Even if the application targets audience with single language, it may change in the future. A multi-language ready application is important to reach out more users.
- Minimize page changes
  - Page changes are slow compared to ajax requests and also causes users to get lost across pages. Single Page Apps (like Gmail) have high usability experience, but development is more difficult and bugs may easily occur. If there are enough resources (i.e. manpower) then go for a single page app, otherwise use ajax abundantly.

- Simplistic user interfaces
  - The age of "learning how to use programs" is over and the applications should be easy enough to be used until the user gets familiar with it. Advanced operations may be revealed after user gets familiar. Complex interfaces scare off the users.
- Global search system
  - The tendency to search has been increased over the years. Google search, Facebook search, Twitter search... all major giants have one global search system which can be filtered after search results are served. Let your users have the same functionality that they used to.

- Global search system
    - The tendency to search has been increased over the years. Google search, Facebook search, Twitter search... all major giants have one global search system which can be filtered after search results are served. Let your users have the same functionality that they used to.
- Always take the user back or forward after an event
    - When there's an error, or a request like password-entry happens users should land where they want to go or where they came. Always take user either where they came from, or where they want to go.

- Mobile-First UI
  - The common way of designing UIs is designing for the desktops first and then adapting it to mobile devices. Although this works, it gives the overhead on mobile devices. The UI must be designed for mobile, and adapted for desktops.
- Global feedback system
  - There are always cases when developers and testers cannot forecast the issues. The best way is to get user feedback with a global mechanism which can be accessed 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 same in every environment. Best way to achieve that is confirming the standards and not using non-standard components. Also design frameworks like Bootstrap or Foundation helps.
- Use friendly urls
  - Although a web application is generally not focused on organic visitors (from search engines) when people share URLs in emails or in IMs the shared person would want to know what will be opened after clicking that link. People tend to explain less, so the URLS they share should at least explain what is that URL related to.

- © http://webappchecklist.com/