# Iterative UI Design Process Technical Implementation



## Need:

UI Analysts need to produce high-fidelity prototypes and pre-production concepts to iterate and test screens **prior to passing them into production**. A/B tests, Interactive demos, and process flows need to be produced, flushed and changed in a higher fidelity than screen shots, but don’t need to incur the cost of a development team for proof of concept. The need to push UI design ahead-of and outside the production process, is essential to user-centered design for ideation and vetting of new interactions, processes and user-centric solutions.

When working in data-dependent code-generating environment, the overhead, technology requirements, and dependencies make it impractical to produce UX-derived work efficiently. We need a pencil to sketch, before we put the design to pen. We need to grow a process that fills the gap between UI mockups to high-fidelity functional prototypes that can be managed by a more generalized skillset. We need a low risk, self-deterministic process, that lets us cycle through ideas without expending valuable programming resources too early in the process. UX needs to work independently, designing and ideating close to where the user interacts as an ideal. Completion of this process is marked by a deliberate hand-off to a production. This hand-off would consist of a dialog between developers and UX designers as to how best to transfer the code for a server environment, and how best to implement it so the output matches the prototype, as dimensionally and syntactically as possible.

### HTML Output must to be…

* Valid (W3C/Accessible/Platform Generic)
* Light and Clean (Minimize DOM traversal, and power consumption)
* Syntactically correct (Provides portability, legibility and the ability to re-use)
* Optimize speed and isolate UI Client-side bottlenecks before other dimensions are added (Be able to evaluate how it plays in the browser)

Over time a wide range of templating solutions and systems have come and gone. For best-of-class web applications, modifying and customizing auto-generated libraries often proves difficult. Often, they need to be patched out of the box, (ExtJS, PrimeFaces, FrontPage, and Smarty to name a few). Yet many developers cling tightly to the dream of magically generated browser code that is light-weight and valid. As web technologies evolve, we come closer to this dream, and can compromise with higher performance machines and products. The reality is that light code is becoming more essential than ever, especially moving away from pure desktop systems. Heavy DOM traversal can have negative effects on battery life and power consumption. This matters more when large numbers of users within the same organization use the same app, or remote users depend more on battery life. It is becoming a standard benchmark in most application performance evaluations.

## Marketing UI/UX Design Environment Heuristic Considerations

We need to start with a solution that provides rapid revision process for interaction and design. UI Designers should be able to maintain their own implementation with minimal intervention. The design structure and process should be simple, and as close to the user as possible. It should be able to run without any outside dependencies, and try to focus assets that can measure in the browser. It should not depend or a larger application structure, but be easily transferred either as a whole or with minimal manual transformation.

## UI context in the larger development process

This process answers the need to design and test before sending new UI flows and screens into production. This way we can test and shape the interaction before implementing something that is not well considered. It will result in a clearer direction for new work, and provide greater reuse. This process also sets the foundation for valid generated browser code. It does not seek to replace any development processes for enhancement and modifications of screens in the system, but provides the seeds for new interactions. Nothing says that the code cannot be transformed, reused, or replicated by fragments, faces or whatever technology is utilized to generate HTML. But the end output should match the UX prototype as closely as possible.

Using a dedicated design prototype site will assist Marketing/UI to remain more consistent in generation of new interactions. It offers a way to implement and reuse code as UI patterns to make the UI more consistent and predicable, and provide more mature content and communication to Production developers. Revisions and changes may always occur during development, but the intent is to reduce much of the churn, and need to compromise, swapping functionality for deadlines.

Ideally, this would have been created in pure html/CSS/js to keep it as simple and straightforward as possible. However, development for prototypes would result in chaos for standard includes (headers footers nav) in a complete prototypical web application. Each page containing its own code for navigation, and global elements would cause confusion, and using relative paths or post-JavaScript rendered includes, would also cause churn. Keeping it simple is one thing, but we also need to take advantage of the ability to reuse code and absolute paths for simplicity and consistency. So I was forced to add a basic Apache instance to solve these two problems. By developing one step removed from server-less technology (i.e. Using Apache) allows us to maintain our own simple server platform. We can test flows, use absolute paths, and leverage includes which, in turn, can map directly to whatever server architecture we decide to use in production.

## UI Design Site Structural Layout

### Global structure

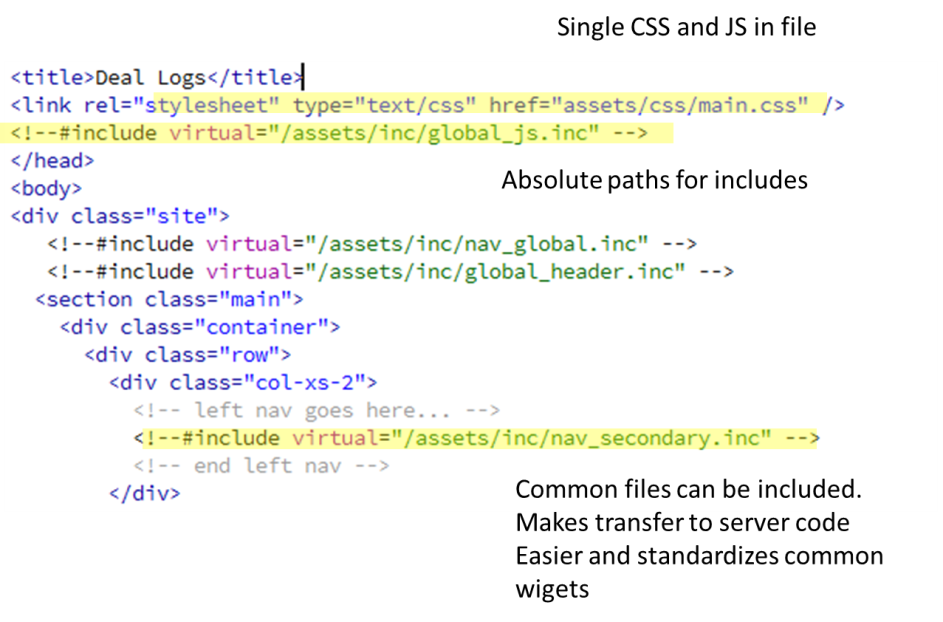
This is a basic Hybrid structure that resembles the standard structure of many web sites, (including corporate site), for location of global resources, such as JS, JQuery, Twitter Bootstrap, Includes, and standard Corporate branding. In this structure, nesting within the site can be kept to no more than 4 levels regardless of page or screen.

The level below the application root contains project-specific variants. To hold the place, I created a generic ST directory (As the framework team has defined the generic project directory for generic state). This would provide state specific content and overrides. We can still leverage includes from the global parent (web root).

Within each state (ST) project may also be page specific variants, which would account for individual “one-offs”. It is my contention that **these variants should be non-existent or very rare, since they interrupt both the intention of UI patterns for predictability and code reusability.**

### Page Structure

Screens are designed to reuse common elements from the parent global structure (corporate header, navigation etc.), as well as local includes. Client-side code, like JavaScript and CSS are integrated by one line of included code for each. This provides the advantage of managing external elements and libraries without having to change anything on each page. It is fundamental to any server generated web application, and though it goes above the tenet of simplicity and minimalism in this project, it makes it much easier to manage and port to other systems. The content area of each page will provide the main variant as that will contain the unique functionality for each screen, and differing combination of JS components, forms and data for generation interaction and display.

 **In page CSS:** The Page links to a single CSS file which provides all required CSS via @import cascade. That way you can do whatever you want to the CSS without having to edit the page. You can also eliminate pesky inline styles, as well as one off CSS class instances that add bloat and confusion.

**JavaScript:** I tried Require.js but it seemed to add unnecessary complexity given our goal, and was my main motivation in the decision to use a server at all, so we could provide logical SS Includes. As I contemplated this, I realized that most of the server-includes could even map 1 to 1 to a production server. JS should be provided by max 2 page includes. The first one would be at the top of the page, and provides core JS libraries. The second may not be needed, but it would be located at the page bottom and provide calls to functions located at the page top, if needed. This include could basically replace the “onReady” or “onLoad” function.

## CSS Implementation

#### CSS structure cascades throughout the platform

**Global.css** This leverages the cascade part of CSS and provides all the include files that are needed for branding and used on every page. This includes the DT styles, Twitter Bootstrap, and “JQueryUI”

**Main.css** In the state specific instances there is a main.css that provides state specific styles, i.e. State logo etc. and can be used someday to brand themes if needed. This links up to Global.css  
**Component.css** *(named for whatever component)***.** Prov**i**des component specific CSS, (although we will try to avoid its use).  
**Style.css** Provides any page specific CSS that isn’t provided by global.css. It is linked through the single CSS file (main.CSS). It should not be needed, but in case it is, I have accounted for it.

## Proposing UI Coding Standards: Pages should…

* Utilize valid IDs for all form tags, and structural containers
* Work with a minimum and maximum page width
* Use absolute paths so they are easy to manage if they are moved or repurposed.
* Pages (specifically reports) should include CSS Media options for print and variable adaptive display, so we don’t need an applet or PDF generator in the application to make the print view acceptable, and we can accommodate a greater range of displays
* Provide calls to included JavaScript files rather than contain any JS logic
* Minimize or eliminate static data like strings passed to function calls

## UI development/design environment Stays within these fundamental technologies:

* HTML (HTML 5 or XHTML)
* JavaScript
* CSS (CSS 2 or 3)
* Apache Web Server (SSI Absolute Paths)
* Adobe Dreamweaver (Code validation, Site Management, UI Design)

Applications like **Grunt** or **Ant**, (tools for scripting things like mapping file paths and running transformation tasks), can automate the process of mapping files and paths to the server via a DIST type operation thus offering a logical structure for developers, while providing a simplified structure for design. We don’t have to argue how files are structured. Each concern can optimize a structure for specific comfort/needs.

## Remaining work and next step decisions…

* Coordinate and verify with Framework team that this will interface with next generation application, while allowing our internal UI research to function.

******

Figure 1. Automated scripts map UI Design structure to one favored by Development.

* Define a process for linking to whatever server technology we use to generate/maintain the code base.
* If it’s one initial conversion then a page at a time, it could be a manual process, since the work would be minimal.
* Otherwise if the code is generic enough, we could create a script in Ant or Grunt to map the code to production. But mappings should be documented, and could provide.
* Define strategies for comparing UI output with original code for basic (or automated) code review.

## Summary

Creating a basic UI/UX specific application container for UI prototyping will help us independently iterate UI prototypes and designs that will benefit users by having tested UIs, and speed development by having more mature and better vetted solution with reusable code to move into production. There are many communication puzzles yet to resolve, but as we evolve to the next generation of the application, we need to evolve the user experience more rapidly. This pseudo development/design process will help us achieve that goal.