VIEW POINT

MODERN UI APPLICATION DEVELOPMENT

Abstract

User Interface development has come a long way in the last decade and a half.

Earlier UI development was primarily focused on generating presentable markup for one or two leading browsers. In modern times, UI development is expected to create remarkable user experiences (UX) that can run across a multitude of browsers and client devices. The main transformation drivers for this change have been the improvements in edge device capabilities and network bandwidth.

In the current era, UI/UX is arguably one of the key differentiators for online business success. This stream is currently drawing huge creative and engineering talent. It has also inspired creation of numerous UI development frameworks for building applications that target browsers, mobile and other client devices.

This paper presents a point of view on challenges, best practices and patterns for building UI applications in modern times - collectively referring to such applications as Modern UI Applications.

While there are many leading UI development frameworks for browser, mobile, desktop and other client devices, this paper focuses more on generic patterns and best practices rather than delving into any individual framework.

Infosys®
Navigate your next

Expectations and Challenges

Modern UI applications need to provide a rich user experience that is fine-tuned to the user, their choice of interaction channel and driven by insights and intelligence .

In online business, user experience is absolutely crucial for acquiring, retaining and keeping customers engaged.

Businesses continue to allocate a large part of their IT spend towards user experience improvement. This is one of the leading drivers for the large-scale re-architecture effort commonly referred to as "Digital transformation."

Modern UI should live up to the many and sometimes conflicting expectations. It should be intuitive, responsive, adapt to different screen sizes, accessible (WCAG compliant), support multiple languages/ locales, be dynamic, reactive, adapt to user behavior, run on multiple browsers/devices, be contextual and predictive, provide advanced visualizations and of course be visually appealing with high usability.

If we envision each of these expectations as a separate dimension, the application needs to shine in each of these dimensions independently.

Below is a (very) brief summary of what these dimensions mean

1) Intuitive: Easy to understand and use, without need of special assistance or training.

2) Responsive: Usable at all times, should continually provide feedback for user actions. At no point should it look frozen or non-responsive.

3) Adapt to different screen sizes: Should be fluid enough to adapt to different display and device sizes including resized browsers and device orientations.

4) Accessible: Should be usable by people with disabilities. It should provide alternate content and additional information e.g. text as backup for images and videos, additional information on the role of a specific UI control, improved contrast and so on. In essence, disabled users should be able to easily perceive and navigate through the content. More information here.

5) I18N and L13N support: Same UI should be able to support different languages, measurement units, currencies, date/time format and any other display metrics which are locale specific.

6) Dynamic: UI is likely to change much more frequently than anything else in an application. It should therefore be designed for ease of rapid change. The "dynamic" capability here refers to the ability of being easily and independently changed.

7) Reactive: UI should be always on; it should be able to allow new data to flow in and automatically react i.e. update itself to reflect the changes.

8) Adapt to User Behavior: Modern UI is anticipated to adapt to user behavior. UI should adapt and present the most relevant content to users. The experience should be as focused and frictionless as possible.

9) Run on Multiple Browsers and Devices: While server-side software runs in controlled and predictable environments, UI typically runs on user devices i.e. in environments which are uncontrolled, unreliable and likely not up to date. UI applications need to adapt to these environments gracefully.

10) Predictive: Modern AI/ML (Artificial Intelligence / Machine Learning) provide Natural Language Processing and predictive capabilities to predict and enhance user experience. UI applications should be designed to benefit from and leverage these capabilities.

11) Advanced Visualizations: Advanced visualization libraries and frameworks like AR/VR (Augmented Reality/ Virtual Reality), WebGL, OpenGL are now available natively in most UI frameworks. With client devices getting more powerful (GPU processing), modern UI applications are well placed to leverage these libraries to support business use cases, gamification and innovation possibilities.

For UI Applications to support these capabilities, they need to follow certain best practices, approach and design patterns.

The rest of this POV attempts to bring out some of the salient ones.

Consider building experience as a separate application

In the past, UI was mostly built as part of server applications. As different programming languages became more and more popular there were corresponding server-side web application development frameworks created for them e.g. ASP.NET (C#), J2EE & Spring Web MVC (Java), Symfony (PHP), Ruby on Rails (Ruby) and Django (Perl).

These frameworks handled all application concerns including UI specific concerns. They generate presentation using templating engines running on the server and send it over the network to the browser.

The main drawback with this approach is that UI concerns, which are fundamentally different, get very tightly coupled to the server-side framework, programming language and infrastructure.

Even if these concerns can be better addressed by a more focused framework, programming language or rendering engine, that opportunity is lost. Also, presentation going via the network makes the experience quite sluggish.

The success of Gmail (Google's email application) brought in impetus to build experience as a separate application. Gmail was built as a Single Page Application (applications that run within the browser) which provided a much more responsive and richer experience.

A decoupled UI application provides full control and flexibility over addressing UI specific concerns independently. The server interaction happens via a headless backend.

Angular, React, Vue are some of the leading UI development frameworks based on JavaScript for building decoupled browser applications.

Some of the key experience capabilities like the ability to rapidly change and adapt experience, while keeping it responsive, can be much successfully achieved with a decoupled experience approach. Mobile Application development frameworks like Android and iOS also create fully decoupled UI applications that have truly transformed experience possibilities.

Layer your UI Applications well. Consider building client SDK for RESTful APIs to keep your Presentation tier decoupled

MVC (Model View Controller) or its variants like MVVM (Model-View-View Model) and MVP (Model View Presenter) are the fundamental "separation of concern" patterns for building UI applications. When building decoupled UI applications, it is essential to follow them and also the "layered architecture paradigm". The presentation tier within the UI application should be totally decoupled from other layers.

Most of the UI applications require integration with RESTful web services for supporting CRUD operations. One of the ways to approach this integration is to create client SDKs for RESTful web services with published APIs for presentation tier to consume. The presentation tier does not worry about how the data is provided – it only knows about the SDK APIs for CRUD operations. Like any good SDK the APIs should be well versioned and backwards compatible.

One obvious benefit of creating a client SDK for a given platform (e.g. JavaScript, Android, iOS) is reusability. It can be used for building other UI applications for the same platform.

The other important but often overlooked benefit is testability of the presentation tier. With client SDK abstracting data source and providing a standard API, it is very easy to mock data (test double) in order to validate presentation for different scenarios without depending on web services.

With this approach, it is much easier to make changes to the presentation independently (even creating a new presentation). Presentation specific concerns like fluidity and accessibility can be scoped to the presentation tier and changed independently. It also provides better structure to development teams with well-defined responsibilities. Presentation tier and backend integration teams can work independently and in parallel.

Making changes to presentation tier or integration layer becomes much easier and independent.

Don't build your Presentation tier, Compose it

The componentization of the presentation tier is a central theme in building Modern UI applications. Leading UI frameworks are built around the concept of creating reusable UI components and composing the presentation tier with them.

With composition, presentation is built with components – like building with Lego blocks. It provides ability to easily swap one component with another and promotes reusability; teams can build reusable component libraries that can be used across applications.

One important aspect of UI components is the state that they hold. State can be looked as the logic and smarts that a UI component possesses.

UI components that include logic and smarts are closer to the application state and hence lesser reusable. These are Stateful components.

On the other hand, UI components that hold no state have no knowledge of application state or context and are highly reusable. These are Stateless components.

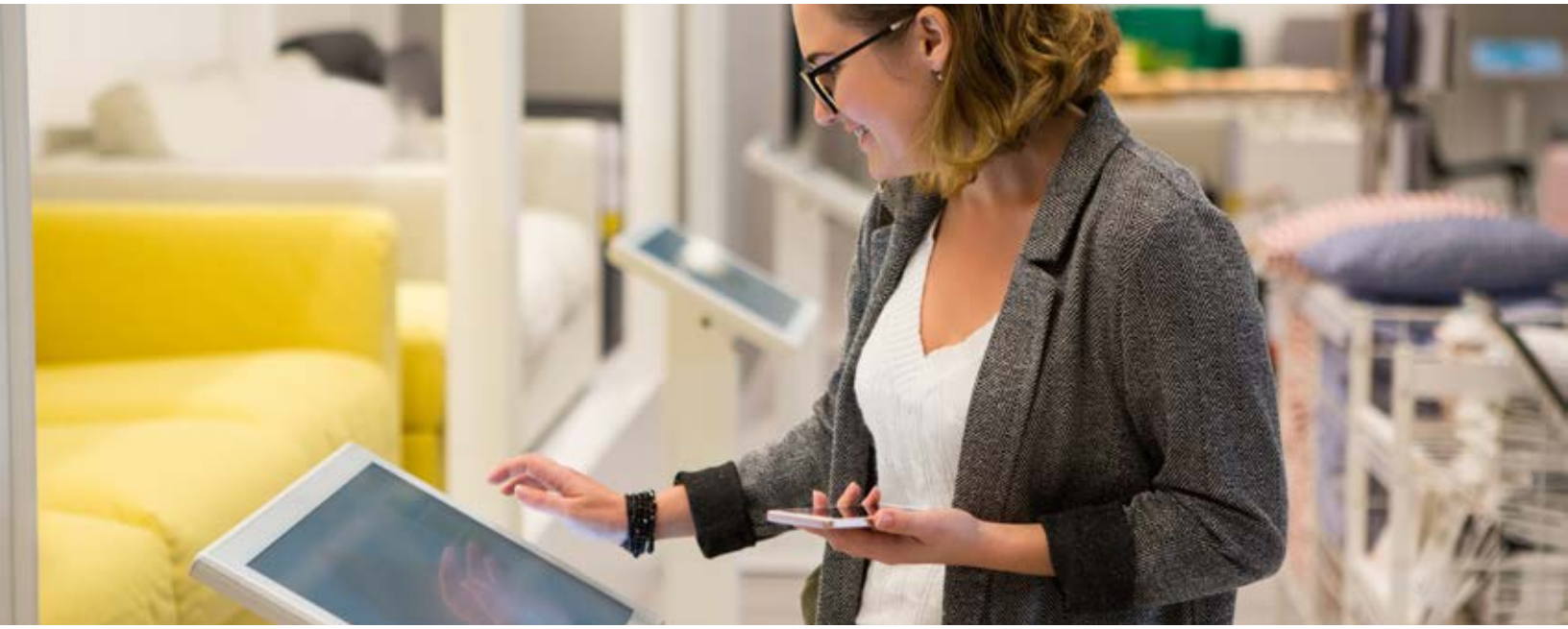
We are essentially categorizing UI components based on "display logic" and "only display" responsibilities. With this, if there are any display logic changes, they can be handled in Stateful components whereas if there are any display changes, they can be handled in Stateless components

Some examples of stateless components could be like date-picker, type-ahead search, dropdown, accordion or carousel type of components - they are devoid of any application state. Everything that these components require is passed to them as reference. Typically, properties to initialize and configure the component and functions to serve as event callbacks. Think of them as stateless functions – the caller needs to pass all the information to the function to operates on Examples of stateful components would be the entire page or workflow type of components. They hold the state and logic to decide what to display and how to configure the display (stateless) component .

The best practice is to keep most of your presentation composed of stateless components and keep your stateful components minimal and well structured. The stateful components should only focus on composing the UI without taking on any presentation responsibility.

This allows for easily changing UI composition i.e. replacing one presentation component with other and hence making UI changes faster and simpler.

This ability to easily swap, rearrange or present new components based on state is key for Modern UI applications where experience needs to keep adapting - to be as usable and frictionless as possible.



Keep your content markup well-structured and as semantic as possible

It is important to keep the markup of content well-structured and semantic. Semantic markup is the cornerstone for making your content readable and meaningful to browsers, accessibility tools, search engines and even users. Essentially markup should strive to convey everything about the content. Instead of using generic tags, sematic tags should be used to give meaning to content. ARIA tags should be used as fallback in case content cannot be described semantically. This way content is well understood and can be decoupled from how it should be presented.

Presentation may need to adapt to device form-factors, theming and other requirements but content structure usually stays the same.

CSS (Cascading Style Sheets) is a standard style sheet language to apply styling to content elements. Based on configuration (form factor, theming requirements), the right set of styling can be applied to the target markup element using appropriate CSS selectors.

Presentation or styling becomes a thin veneer over your content which can be easily changed.

A basic example could be a page having header, footer, navigation, main content area - all described by semantic tags. While mobile and desktop browsers would include the same navigation content, presentation would be very different. In case of a mobile browser, a slide out menu but for a desktop browser, a simple navigation bar.

Writing structured and maintainable CSS is an advanced topic and warrants its own point of view. Most of the CSS written today uses preprocessor style sheet languages like LESS and SASS which get compiled to CSS. These languages fundamentally allow writing DRY and maintainable CSS.

Semantic markup is a simple yet very powerful concept. UI fluidity and accessibility largely depend on how well the content is structured and how semantic it is.

Many-a-times this aspect does not get the desired attention from the beginning. This can lead to a lot of re-work and heartburn later when different presentation/ styling requirements come up and more importantly when there are accessibility compliance issues.

Micro front-end design pattern – Breaking the UI Monolith

UI Applications can become very sizable, spanning multiple functional areas. Most UI applications were typically built as single monolithic applications. This approach is now being reconsidered as monolithic application are much harder to build, maintain and change.

With the popularity of microservices architectural style, the possibility of developing UI applications as modular Micro frontends is gaining traction Micro-frontend is not a new pattern, but now we are seeing higher adoption and this architectural style is getting mainstream.

Like Microservices, the micro frontend approach is based around breaking the UI into independent modules. It can be visualized as vertically slicing the layered architectural view based on functional areas. It aligns with the modern development paradigm wherein functional teams can build and release their applications very independently and on their own release schedule.

We have done large-sized UI implementations using the micro front-end pattern. One of the approaches used has been to build micro frontends as web components (that wrap a complete module) using frameworks like ReactJS or VueJS.

These web components are actually light-weight single page applications that can be wrapped and deployed as per specification and support of the Web CMS (Content Management System) or Digital experience platform being used.

Mobile development is also slowly moving away from the monolithic application mindset. Mobile applications that are sizable and provide a large set of functionalities are being broken into smaller modules that can be built and tested independently.

Implementation of micro frontend is an involved effort, it requires a well laid out architecture, careful planning and discipline. It is definitely worthwhile considering this approach for the huge flexibility and benefits that it brings in. With a micro-frontend architecture, we have the ability to independently and rapidly add or change individual modules without impacting the overall application.

Effectively use functional programming paradigm for building UI Applications

Most of the UI application runtime environments support development languages that are based on functional programming. JavaScript, Swift and Kotlin are leading examples. Effective use of functional programming paradigm is crucial.

You should strive to write most of your data processing and business logic code as pure functions. The main attribute of a pure function is that it does not cause any side effects i.e. it does not alter application state or mutate any data. For a given input, it always produces the same output without exception.

Functional programming is a different programming paradigm and requires a mindset change to truly benefit from it. It is quite different from the traditional object-oriented paradigm where it is common to share state and mutate shared data.

Functional programming languages do not provide any restriction in terms of usage. Developers can continue using these languages with an object-oriented mindset and this is where the main challenge lies.

Effective use of functional programming paradigm will make your code much more testable and declarative (as opposed to imperative).

One of the other areas where functional programming is essential is for building reactive applications i.e. applications that support processing of events as they happen in real-time. For UI applications this could be processing of UI events or API responses as they happen. A reactive application is always on. It keeps reacting and adapting to internal and external events. Functional programming is central for implementing reactive behavior.

Reactive frameworks expect you to write your code as pure functions, in a more declarative manner.

It is important to understand and leverage the true power of functional programming to build testable, composable, declarative and reactive UI applications.

Keep your UI Application lightweight – Consider using external cloud / third-party services when possible

One of ways to keep UI Applications more dynamic and maintainable is to keep them lightweight and simple by avoiding inclusion of intelligence or logic that is not presentation specific.

With the emergence of several cloud services that support/enhance UI applications, there is a huge opportunity to leverage them instead of building something custom and adding complexity to the application.

Some of the leading examples of such services are around Storage, Authentication, Realtime database, Notification, AI/ML, RemoteConfig, Dynamic Links, Logging, Analytics and GraphQL.

These services are available via client SDKs as APIs and provide a reliable and fully managed solution. The SDKs usually follow the same pattern across providers wherein UI applications can centrally initialize the SDK and use exposed services via APIs.

Moving processing and complexity to external services also make applications more secure and performant as most of the processing is moved away from client device to backend servers.

One other benefit with using external services is the ability to track usage and sharing of data across channels. It is common for users to use multiple platform versions of the same app e.g. iOS, Android or Web.

While using external services via SDKs it is important to design interfaces that abstract the service provider from the application. This allows for flexibility in terms of swapping one provider with the other.

Currently the leading UI service providers are the Firebase group of services from Google and Mobile/UI services from Amazon and Azure.

There are many third-party service providers as well that target niche segments.

In summary, by using external services via APIs, we get a range of infrastructure services (like logging, authentication and storage) and intelligent capabilities (like AI and ML) without including any significant code or logic. The key benefits are that the application is more lightweight, maintainable and can focus more on experience. It also brings in consistency and possibility to share data across applications. In modern times, the approach should be to consider composing your application behavior (using existing services) rather than building everything. This also aligns with cloud native architecture guidelines - to leverage SAAS service where possible.

One other area where integration with cloud services is gaining momentum is on real-time processing of clickstream data and applying machine learning. This brings in opportunity of optimizing experience and providing improved recommendations in real-time. It is a rapidly evolving area and slated to drive the next wave of experience and personalization possibilities.

Maximize usage of third-party (sometimes De-facto) libraries

A whole eco-system of tools and libraries has evolved around UI application development frameworks to make development easier, faster and standardized

There are several open source and third-party libraries that have become de-facto standards. They significantly reduce development effort and keep complexity out.

It is important to analyze and identify such libraries right at the outset to avoid any unnecessary custom code and associated complexity.

While there are many open source UI component libraries as well, in our experience we've seen that the presentation part is always a more custom effort to align with the UX and branding needs of a specific brand.

The third-party libraries referred here are more focused on non-presentational concerns like Networking, Dependency Injection, Data Serialization/Deserialization, Reactive extension, State Management etc.

Few Examples -

Android: Retrofit, Picasso, Dagger, RxJava.

iOS: AFNetworking, SDWebImage, Almofire, SwfityJSON

React: Redux, Enzyme, Axios

Leverage Native Capabilities – for a more predictable and consistent experience

Over the past decade, client device capabilities have become extremely powerful, most of these capabilities are exposed as Native APIs but many-a-times they remain underutilized.

Location, camera, audio/video playback, data storage, notification, biometrics are some of the leading capabilities that modern client environments expose. For example, Biometric authentication using fingerprint of Face Id is still not being utilized in many mobile apps.

Applications should leverage these capabilities as much as possible. In fact, one of the main objectives of UI application should be to utilize these capabilities for a much richer and consistent experience improving overall usability and avoiding any custom implementation.

Plan for Browser/Device compatibility for your UI Application

As earlier mentioned, decoupled UI application run in client browser/device environments which the application developers has no control on. This brings in additional complexity – to ensure that the application runs seamlessly on all these environments.

For browser environments, HTML5 specification is trying to bring standardization in terms of APIs and capabilities that different browsers provide. Still there are many discrepancies in browser implementations. Also, many customers don't upgrade their browsers and continue using older versions .

The application development effort should plan for such discrepancies.

The approach could either be to include that capability via Polyfills, provide alternate experience or any other approach but it should be planned for and not left to be discovered later. One of the techniques we typically use is to encourage developers to use different browsers in a round-robin fashion, so the application keeps getting tested in all browsers.

caniuse.com is a reliable reference to validate browser support for various capabilities.

Conclusion

Front-end development cannot be considered by any means superficial. Modern UI Applications need to be extremely well designed and architected to be able to address the ever-increasing list of challenges and expectations.

In this point of view, we've tried to cover some of the important design and approach considerations that we've employed in our implementations, but the list is in no manner complete or comprehensive in nature.

We believe that it should provide guidance to UI developers and architects and hopefully trigger more thoughts and ideas to improve the design and architecture of your UI applications.

About the Authors

Manish Kumar Jain

Senior Principal Architect, DX - Infosys

Manish is a Senior Principal Architect – and 22+ year Infosys veteran – who has architected IT solutions for numerous clients in the Hi-Tech, Retail and Financial sectors. His depth of experience across Digital Enterprise, Mobility, Cloud, Social Media, Analytics, eCommerce, and engagement solutions has allowed him to partner with clients to effectively define and design technology solutions for large, complex, multi-platform enterprise architectures. Manish currently heads the Architecture practice in our Digital Experience Unit. In the past, Manish has served as Chief Architect for our SocialEdge Platform solutions, Engineering lead for iEngage Social Commerce platform, and executed numerous architecture and design engagements for global, enterprise finance, technology, entertainment, and pharma clients. Manish holds a Bachelor of Technology from IIT Bombay. His interests include squash, cricket, tennis, cycling, yoga and most other sports.

He can be reached at manishkj@infosys.com.

Amit Nigam

Principal Technology Architect, DX - Infosys

Amit is a Principal Technology Architect with Infosys. He has over 20 years of experience in IT with recent focus on Digital Modernization and Transformation programs.

His current technology focus areas are Modern Web Applications, JavaScript UI Frameworks, Mobile Applications and Cloud Native Application Architectures.

Amit holds a Bachelor of Technology for IIT (BHU) Varanasi.

He can be reached at amitnigam@infosys.com.

For more information, contact askus@infosys.com

Infosys®
Navigate your next