VRIJE UNIVERSITEIT BRUSSEL

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Applied Sciences and Engineering: Computer Science

INTERTEXT

The Everything App

OGUZ GELAL

Academic year 2020–2021

Promoter: Prof. Dr. Beat Signer
Advisor: Prof. Dr. Beat Signer
Faculty of Sciences and Bio-Engineering Sciences

Abstract

maybe add one or two sentences to put everything in the right context

From a feature-based standpoint, many front-end systems share similar patterns but implement them in their own ways. The correctness and completeness, as well as the intentions of these implementations relies mostly on the developer due to the lack of a systematic enforcement mechanism for quality control and standards in the open market. Moreover, one who aims to add value with their data and services needs to build accessible front-end applications with best practices that are optimized for different devices and screens; oftentimes resulting in inconsistent, unstable and low-quality byproducts. The aforementioned problems entail~~s~~ a bad user experience and bring~~s~~ along security and privacy concerns for the users; and increase~~s~~ costs and efforts significantly for the providers. To address all these problems we propose Intertext, an open platform that offers an alternative front-end for providers and a private and secure unified viewing experience for the end-users. Intertext allows developers to generate and serve the description of their front-ends from a generic backend endpoint in Intertext UIDL (User-Interface Description Language), a JSON-based syntax to âĂIJdescribeâĂİ fully functional front-ends agnostic of style, layout, device and environment by composing a given set of components and commands. It also offers client-side applications built and optimized for multiple different platforms and devices, that can receive Intertext UIDL and render user interfaces in the most appropriate way based on the host device and environment.

a f ?,

Acknowledgements

Contents

1 Introduction

The leap in the IoT (Internet of Things) in the recent decade has opened doors to a new era in information technology. New realms of interconnected devices and device families introduced different ways of interacting with information. Emerging new tools, techniques, frameworks, libraries, and ~~sdk~~'s made SDKs it possible to build consumer-facing products in ways that were never possible before. These advancements attracted many users and developers, and helped to create a large and diverse market of consumer products, goods and services. However, it did not come without some hurdles. In this thesis, we group these hurdles under three main categories; consuming data, privacy and ~~&~~ security, and providing data. The Consuming data section discusses the problems faced by end users due to the inconsistencies of user interface and experience, and lack of device and accessibility support. The Privacy and Security section argues why there are no truly private and secure environments. Providing data section gives some insights on the challenges faced by data and service providers in creating front-end applications.

Consuming data

The simplicity of consuming data is lost within the complexity of the modern day applications; nowadays something as simple as checking the weather, reading a news article, browsing an image gallery, buying a product or service,
Or filling out a form ~~etc~~. could be a frustrating and time consuming task. One

of the common reasons for this is the inconsistencies and errors in front-end implementations. In the open market, where everyone can create user-facing applications, there exists little to no enforcement or quality control measures on how a user-interface should look like, or how it should behave. The way the same functions are presented can greatly differ based on the implementations. For instance, a navigational menu component ~~in~~ on a website could be on the top while ~~in~~ another ~~could~~ implementation might have it on the left or right, some could offer a hamburger-menu style functionality where the user has to click / tap on the icon to toggle it on and off, in some swiping from left of the screen gesture might toggle the navigation menu on, while some other might implement the same gesture as âĂIJgo backâĂİ function, and so on. In some cases the developer can choose a bad selection of a color palette causing some components to blend into the background or make it hard for the users to see. Users oftentimes have to take a second to adjust to every different experience from every different provider. While some providers with higher development budgets create flawless experiences for their users, this is not always the case and there is no guarantee.

use \dquote and \squote

Another major hurdle in consuming data for users is device and screen size support. Providers typically need to spend significant extra effort to support different devices and screen sizes, should they want such support in the first place. Many providers choose not to have this support in order to reduce development cost, leaving some users with bad experiences or no experience at all. Different levels of support for different environments might cause confusion and frustration, for instance some providers might offer a native mobile application with native mobile gestures, while some might have a PWA (Progressive Web App) (PWA) that comes with native-like gestures and some might offer a mobile-web experience that may or may not come with smooth gestures at all. Another major experience difference between mobile applications and web-based applications is the style of navigation. Users who are accustomed to using native mobile applications could expect the navigational history to be retained between tabs, and get frustrated when they realize that this is not the case on a mobile-web application. At times improperly handled navigation in a SPA (Single Page Application) might even cause the native âĂIJbackâĂİ functionality to throw the user out of the application back to the previous one.

Creating accessible front-ends is another thing that providers have to spend significant efforts on, so much so that there are even developers who specialize only in this field. Creating accessible applications is not always the case first priority on many development projects due to the costs and efforts involved. The diversity in user interfaces has a direct effect on the accessibility aspect, as it is another implementation detail that is left to the developer. There are many different ways of creating accessible user interfaces; different implementations for different kinds of accessibility needs, and different ways of implementing each one of them. For instance in a web application, adding âĂIJfocusâĂİ states to DOM elements is the most basic form of accessibility implementation that allows users to tab into a specific element to be able to interact with it. Designing the order in which they receive focus is a whole other dimension.

Privacy and Security

are you planning to deal with security?

As mentioned before, there is a lack of a systematic enforcement mechanism for front-end applications. There are some protective measures in place, however it is most likely the case that these measures do not guarantee a safe, secure and private environment. For instance, most popular application stores typically have policies and guidelines on what the applications they distribute are allowed to do, but offending practices, especially the non-obvious ones, likely flies under the radar as most app stores do not require the source codes to be provided. Browsers and operating systems block suspicious activities to some extent, but they do not (or cannot) interfere with the legitimate (or legitimate-looking) ones. Governments implement various forms of cybersecurity laws to protect users, but of course laws are for the law-abiding, and as long as these laws cannot be enforced effectively, there is no safe environment for the users. The bottom line is, as long as users are required to execute code on their devices, they cannot be truly safe.

Providing Data

The word Providers refers to anyone who offers data and services to end users, and it is safe to say that the problems discussed under the Consuming Data section are shared also by the providers. The complexity of building a decent, well designed, accessible front-end experience, not even once but once per every client built for different platforms forces providers to choose between supporting multiple devices, following best practices, creating a good experience etc. It is typically the case that at the start providers cannot have it all, as it requires significant costs and large teams.

Code sharing is one of the recent trends in front-end development that has been gaining some traction, which aims to solve some of these aforementioned problems; by using technologies such as react-native, react-native-web and Flutter, one can have a single codebase that produces applications for multiple environments. Modern front-end libraries / frameworks typically abstracts away the view layer, making it possible to create applications for different environments by swapping the view layer out and wiring it up to the logic layer, and this seems to be the current status quo in cross-platform application development. The issue that comes with this approach however is the difficulty in reducing the codebases for different environments into one, as it is hard to address different requirements and features of each platform. Even though it is possible to share some extent of the code, at times platforms are conceptually different and the software for these platforms needs to be built in a different way.

Another pain point faced by providers is maintenance. The world of front-end development is an extremely fast growing and evolving ecosystem. The changes are very frequent, and at times breaking. There are thousands of tools, libraries, frameworks, SDKâĂŹs available at the fingertips of developers at no cost. It is a common practice to make use of these libraries as they help with the development significantly. However, the diversity in the libraries used to build the software likely results in a diversity of maintenance problems. Even the most well-tested and maintained libraries could break after an update, causing headache for the developers and hardship for the users.

- problem statement
- methodology
- contributions

2
Related Work

3
Intertext

Intertext is a platform based on a very simple premise, it is a family of front-end applications that can ~~interpret Intertext~~ UIDL (User Interface Description Language), and generate appropriate front-ends for the host platform. Simply put, a provider wanting to create an Intertext application will create a generic backend that generates Intertext UIDL, and serves it from an endpoint, say at intertext.example.com. ~~And~~ users who wish to use this application will, just like in a web browser, pull up an Intertext client and visit the domain intertext.example.com. The Intertext client will then make a request to this domain and fetch the Intertext UIDL served by this endpoint, and generate the user interface as per the instructions received. Moreover, rather than rendering a simple static view it will perform some tasks such as accepting user input, navigating to different screens, making additional requests to fetch more data, keeping the UI updated and reading and writing some data to users local storage; all of which will again be orchestrated based on the instructions received by the backend in Intertext UIDL syntax. Intertext will have multiple software clients built natively for various platforms that can interpret Intertext UIDL in the most appropriate way; for instance users browsing an Intertext app through the smartphone app will receive an experience optimized for touch screens, command-line interface client users will receive an optimized experience for the command line, or a user browsing from a low-end device with limited capabilities will use the version optimized for low-performance devices to get a comfortable viewing experience.

Intertext gives providers a set of components to build and serve their UIs for their services. It is agnostic of what this service is, and given the involvement of a backend to handle all the logic, this service can be anything. In other words; users can enjoy their todo lists, habit tracker, notes, calendar, email client, social apps, news, weather etc. all through one single app, using the Intertext client of their choice. Providers can describe the components their user interface should consist of using Intertext UIDL, in a way that is agnostic of their styles. For instance, providers could specify using a âĂIJCall To ActionâĂİ component, and specify certain properties such as what it should say and what it should do, but they cannot decide how it should look. Having a unified set of UI components brings many advantages, most notably consistency. Every single application on Intertext will look and feel the same, made out of components that users are familiar with. Customizability is another major advantage, users will be able to adjust the look and feel of these components, allowing them to personalize their browsing to their likings all across the platform applications. Last but not least, all components will come with accessibility built in. ? you will have built it This is particularly important that with this approach, the accessibility implementations are not left to the developers responsibility, therefore users that are in need of certain accessibility aspects will be guaranteed to have the accessibility features they need for every single application on the Intertext platform. Moreover, Intertext will be an open platform, and Intertext UIDL will be well documented for developers to build client applications, allowing the community to develop very specific client applications that can interpret Intertext UIDL in meaningful ways to respond to very specific needs. Whether it is a VUI (Voice User Interface) client, a Tangible UI client, or even clients built for particular devices with specific requirements for targeting various communities or use cases, they will all be able to support all existing Intertext applications served from backend services.

When it comes to privacy and security, the bottom line is that Intertext takes away the ability for providers to execute code on users devices. Applications running on Intertext clients are expected to implement all their application logic on the server side, and serve some instructions to the Intertext clients on what to do, how to function, what to show the user and so on. These instructions are a part of the Intertext UIDL, and they are purely based on data, nothing that is executable is allowed from the providers. Intertext allows applications of certain functions that are required to build a meaningful front-end application, for instance an application can instruct Intertext to store some data to the local storage, read some previously stored data, make requests and so on. Applications are also allowed to ask certain user data, such as access to camera (for devices that has one), notifications, location etc. however given that all these instructions are bits of data that could be read and understood by the Intertext client, it is essential for them to be communicated to the user and ask for permission before granting permission to the provider. Intertext by default blocks cross-origin requests, that means a provider serving data from an origin can only ask the Intertext client to make a request to the same origin. Local storage access is also bound to the origin, providers serving data from an origin cannot access the data stored by another origin, which prevents users to be traced across the web for targeted advertisements and such. And for transparency, it is a requirement that all the requests that go back and forth be displayed to the end users in a way that they could understand. This controlled approach guarantees maximum level of privacy and security. Last but not least, Intertext receives data from backend servers in small packages, and the entire communication between Intertext clients and the server can be encrypted. This encryption can be enforced by the Intertext client.

As convenient Intertext is for end users, it is the goal of this project to create advantages as attractive for data or service providers as well. It may not be the best option for all cases; such as for front-end heavy applications that require client-side computations, custom styles or advanced graphics. However for most cases it serves as an alternative front-end for providers that has so many advantages over building and maintaining front-end applications from scratch. To start with, it removes the necessity of building and maintaining front-end applications. Granted, there is an overhead of creating frontend business logic in the backend and to generate and serve the Intertext UIDL, however the effort required is nothing compared to all the hurdles mentioned earlier. Intertext is agnostic of where the data is coming from and has no opinions on how it is generated. Therefore, the providers can easily make use of their existing backend services to add in the front-end logic. Intertext UIDL is simple JSON-based and working with data is an essential part of every application, therefore it is a minor effort but greatly rewarding. Once a provider starts serving their front-end in Intertext UIDL, that means they immediately obtain front-end applications for every platform that Intertext supports. Furthermore, as more Intertext clients get built, either officially or by the community, their applications immediately start working with those clients without requiring any change.

Intertext creates equal opportunities for everyone. The advancements in backend technologies these days made creating backend services possible in ways that was never possible before. Service providers such as AWS, Google

Cloud and Azure made spinning up a backend infrastructure with necessary components as simple as a few clicks, all without requiring significant DevOps skills. Their generous free tiers, scalable infrastructures and the recently emerging serverless technology pulled the costs down so significantly that anyone who is familiar with backend technologies can create an application that could scale up to serve hundreds of thousands of users. However, one of the biggest blockers in launching applications that could gain popularity and evolve into a successful startup is arguably the branding and quality of the front-end applications. An application that suffers from user-facing problems mentioned earlier doesnâĂŹt leave a good impression and creates a bad image, and it is rather uncommon for such applications to be taken seriously, regardless of the quality of the data and services provided. Those who can achieve a good user-facing presence while targeting multiple devices and platforms are commonly large companies with high development budgets. This hurts indie developers, particularly backend developers who do not specialize in front-end development but are capable of creating applications as personal projects that could easily qualify as a good product. This project aims to remove this hegemony of âĂIJjudging a book by the coverâĂİ, when all the applications look the same and feel the same, they will be judged the same. Not only indie developers will have better chances of success, there will be more options for everyone.

One of the potential future plans of the Intertext projects could be the efforts made in web syndication. Web syndication is a form of communication between service providers and clients, where services make their contents available to websites or clients in a standardized way. A popular example of this is RSS; the technology that allows services to offer a feed of their contents in a standardized xml syntax, and RSS clients to subscribe to multiple RSS sources and aggregate all content into one single feed for the users to easily consume. While Intertext already offers a somewhat similar experience where users can consume data through a standardized user interface, it is still not the case that data from multiple sources can be aggregated into one view. If it is ever the case that Intertext gains enough traction to attract end-users, application developers and build a community around itself, then it wouldnâĂŹt be unreasonable to think that it can introduce similar standards in which the applications could offer designated endpoints to accept query parameters in a standardized format and return Intertext UIDL in a standardized format. This could allow Intertext clients to offer RSS-reader-like functionality allowing users to consume data from many different sources into one single view. Furthermore, combining this concept with the power already offered by Intertext clients and Intertext UIDL, experiences significantly richer than what was possible with technologies like RSS could be made possible. For example, a dedicated âĂIJsocial media endpoint standardâĂİ could be introduced by Intertext, and social media services that offer Intertext applications could create endpoints that comply with this standard. Then feeds from these multiple sources could be gathered in one dedicated âĂIJsocial media viewâĂİ. The standard like, dislike, comment etc. interactions would function and update the relevant source again in a standardized way.

A
Your Appendix

Bibliography