Migrating from TDS

This document aims to help you upgrade an existing application that uses TDS to use the new Universal Design System(UDS) with the TELUS ds-allium package. The package simplifies the installation process by bundling base-components, the TELUS (Allium) theme and web-only components (these web components are available only for the TELUS brand).

System requirements

Before installing, be sure to check the system requirements.

Compatibility

UDS has been rebuilt from the ground up using new technologies. As such there will be significant breaking changes compared to TDS. However, the team has made a concerted effort to minimise these changes, using APIs and idioms common in TDS wherever possible. The set of components offered by UDS is likewise inspired by TDS and the overwhelming majority of these components will have an extremely similar API to their TDS counterparts. Below we outline a few areas which are likely to involve some rework when upgrading.

Use of React Native

UDS uses React Native to build cross-platform components. React Native can be used on the web via React Native Web. This enables re-use and consistency across platforms, the cost is that it introduces a layer of abstraction between web primitives (DOM elements, CSS *etc*) and React code. This is similar to how most component libraries work, including TDS, in that it is not possible to directly manipulate the HTML and CSS generated by the component. However if you are currently relying on TDS elements rendering a particular HTML structure or element, you may need to refactor your code to work with UDS.

Another potentially surprising feature of React Native for web developers is that it uses flexbox by default, and **defaults to columns, not rows**. This is abstracted in all UDS React Native components, however some components

may expect to be rendered in a flex container. Where this is the case we have made an effort to call this out in the documentation.

Themes and variants

The UDS theming mechanism is used to style UDS Base components, it allows for unlimited visual "variants" of a component. These components accept a `variant` prop which controls which variant styles are applied. So whereas a typical TDS component might have looked something like:

```
<MyComponent size="small" priority inverse ... />
```

In UDS these variant props are typically grouped into a single object.

```
<MyComponent variant={{size:"small", priority: true, inverse:
true}} ... />
```

Strategy

You can choose to upgrade to UDS all at once, or on a component-by-component basis, or on a page-by-page basis.

All-at-once

This is the easiest method for small codebases, as you will not need to worry about visual inconsistency between the TELUS theme and TDS. It is only feasible for smaller applications where most active development can pause for the duration of the upgrade.

Component-by-component

From a technical point of view, the TELUS UDS theme can be used alongside TDS without any issues. All TELUS UDS styles are completely self-contained and don't require any additional CSS reset. However from a product point of view this is not desirable as the UI will look disjointed. This approach could work well for applications where relatively few TDS components are used (mostly custom UI) and/or where the UI is not customer-facing.

Page-by-page

This approach mitigates the problem of mixing the TELUS UDS theme with TDS by ensuring each page is consistent within itself. This approach will scale to a larger, customer-facing application but will require extra developer effort for the duration of the transition versus an all-at-once approach.

Edit this page