Design Systems

CS 346 Application Development

https://student.cs.uwaterloo.ca/~cs346

Contents

Introduction 2
What is it? 3
Benefits 4
Options 5
Material Design 3 6
Introduction 7
Dependencies 8
MaterialTheme 9
Theme Builder 18
Bibliography 20

Introduction

What is it?

Applications are complex. Even the simplest one will have multiple screens, with complex components and lots of ways for a user to interact.

How do you ensure consistency and polish across your application? If you're a large company, how do you ensure consistency across a suite of applications?

A Design System is a comprehensive framework of standards, reusable components, and documentation that guides the consistent development of digital products within an organization. It serves as a single source of truth for designers and developers, ensuring consist design across projects.

A design system may include [1]:

- **Design and style guidelines**: addressing font usage, colors, spacing, layout of visual elements.
- **Branding guidelines**: naming conventions, logos, and guidelines on how a product needs to be presented.
- **Component libraries** i.e., components that are designed to work together and provide a consistent look-and-feel.

Benefits

What at the benefits of a Design System [1]?

- **Consistency**: every product looks and behaves the same as the others.
- **Transparency**: teams agree on standards across design/development.
- **Efficiency**: make design decisions once, instead of repeating them.
- **Reusability**: common components can be reused instead of reinvented.

If you are a large company, it's worth the investment to ensure that your platform or products are as easy to use as possible. Examples include:

- UXPin is created by Paypal for their product suite. You need to be a Paypal developer to take advantage of it.
- Similarly, the Carbon Design System is used internally at IBM.

> **Tip**
>
> Most of the time, you cannot justify the time or expense of creating a custom Design System.

Options

There are different flavors of design systems out there:

1. **Platform design systems**: Apply and Google offer design systems that define the standards for interaction on those platforms. i.e.,

   - Apple's Design System for macOS and iOS.
     - The audience is anyone building iOS or macOS applications.
     - The benefit is that applications behave consistently and feel "native".

   - Google's Material Design for Android.
     - The audience is Android developers.
     - The benefit is also "native" look-and-feel.

2. **Third-Party design systems**: these attempt to provide a "neutral ground" between OS vendors. They are not nearly as popular as platform guidelines.
   - e.g., Composables UI is a third-party solution for Compose.
   - e.g., Carbon Composables is IBMs Design System created in Compose.

Material Design 3

Introduction

Material Design 3 is the newest version of Google's open source design system. `Material` includes in-depth UX guidance and UI component implementations for Android, Flutter, and the Web. Material consists of:

- Style attributes
  - Typography, color & themes, elevation, motion
  - Patterns for responsive layout
- Components
  - Standard components, and best practices.
  - Specialized components for Android e.g., carousel, dialog, toolbar.

> **Note**
>
> Composables are designed to use Material style attributes by default!

Dependencies

To use Material3 with compose, you need to import dependencies:

```
dependencies:
  - $compose.desktop.currentOs
  - $compose.material3
```

When importing classes, make sure to use the `material3` package:

```
import androidx.compose.material3.MaterialTheme
```

> **Tip**
>
> Material 2 is included in Compose as `androidx.compose.material`. If you don’t change your imports, you will be using the wrong version!

MaterialTheme

MaterialTheme is a Composable that serves as a container for our theme's properties. By just specifying `MaterialTheme` (base class) we get the default Android style.

```
@Composable
@Preview
fun App() {
    MaterialTheme { // theme
        Column(
            modifier = Modifier.fillMaxSize(),
            verticalArrangement = Arrangement.Center,
            horizontalAlignment = Alignment.CenterHorizontally,
        ) {
            BasicText("Hello, World!")
        }
    }
}
```

We can override and customize this theme as well.

MaterialTheme

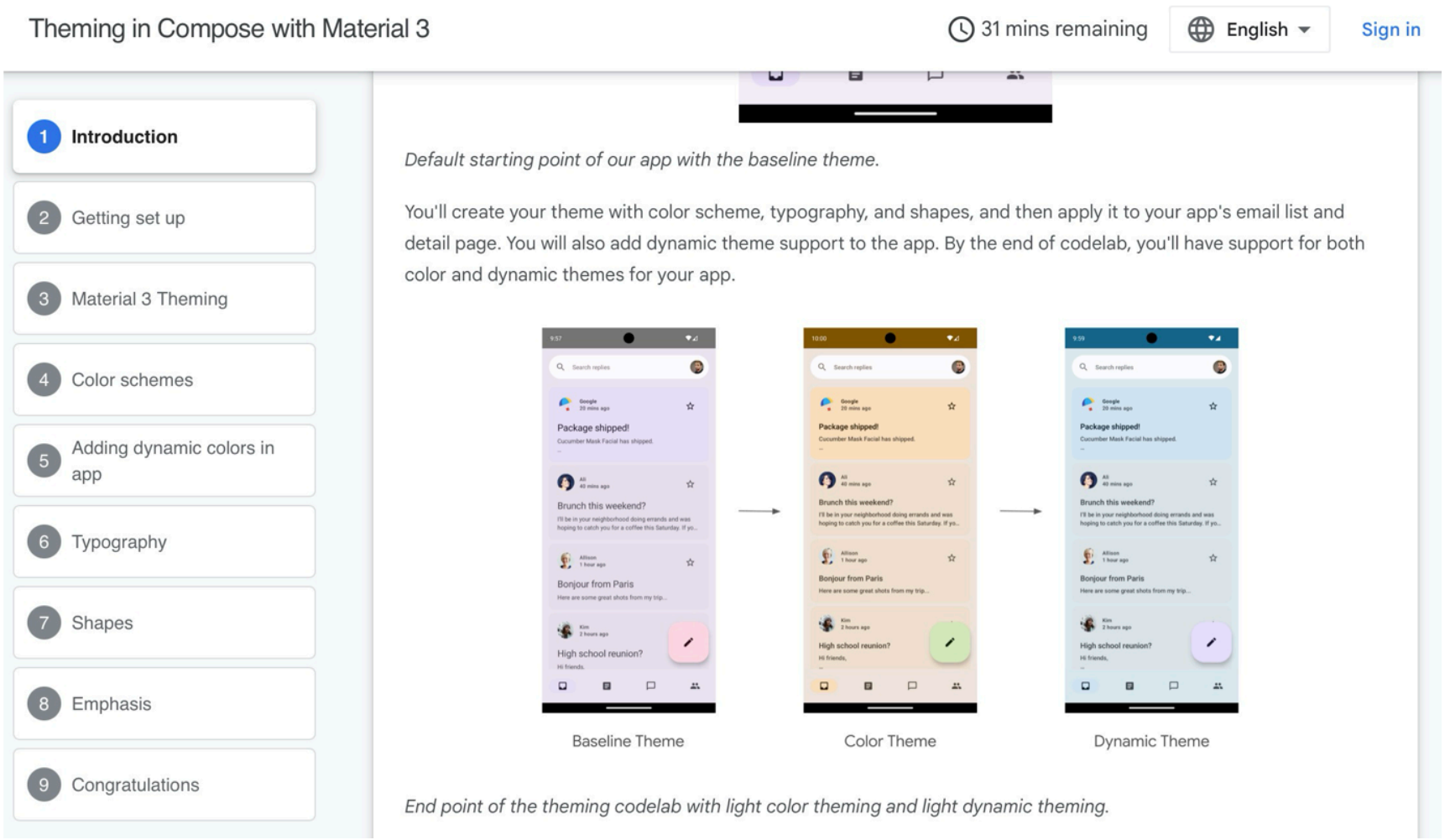


Figure 1: See the [Theming in Compose with Material 3] tutorial for a detailed walkthrough.

MaterialTheme

This theme defines a number of subsystems:

- Color scheme: Create accessible, personal color schemes
- Typography: Use typography to make content readable and beautiful
- Shapes: original shapes, a corner radius scale, and built-in shape morphing

When you customize these values, your changes are automatically reflected in the M3 components you use to build your app.

```
MaterialTheme(
    colorScheme = ...
    typography = ...
    shapes = ...
) {
    // M3 app content
}
```

MaterialTheme

Color Scheme

The foundation of a color scheme is the set of five key colors. Each of these colors relate to a tonal palette of 13 tones, which are used by Material 3 components.

Accent Colour Roles

- **Primary** is the base color, used for main components like prominent buttons, active states, and the tint of elevated surfaces.
- The **Secondary** key color is used for less prominent components in the UI, such as filter chips, and expands the opportunity for color expression.
- The **Tertiary** key color is used to derive the roles of contrasting accents that can be used to balance primary and secondary colors or bring enhanced attention to an element.

Others

- **Surface** is used for backgrounds and large, areas of the screen.
- **Container** is used as a fill color for foreground elements like buttons.

MaterialTheme

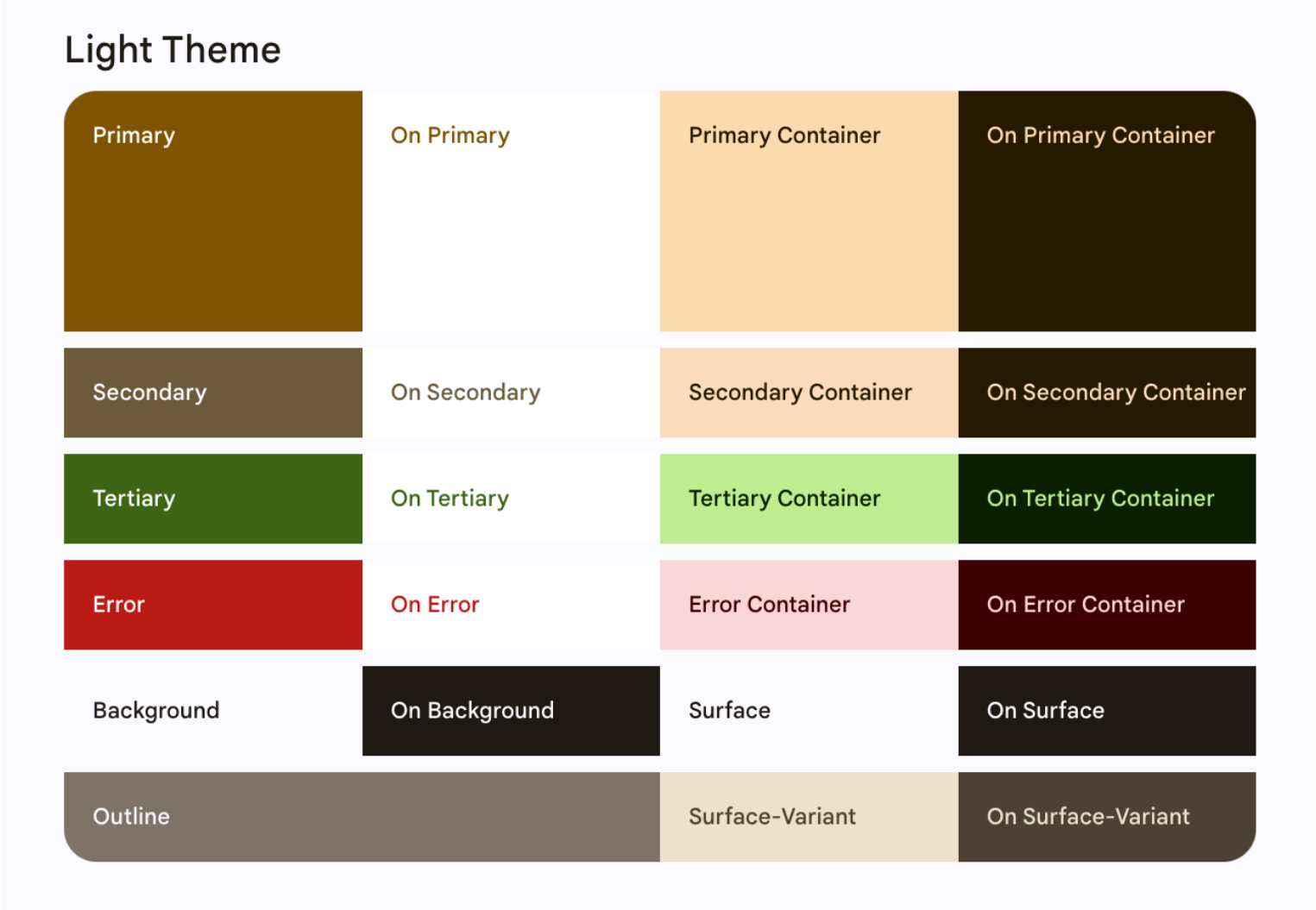
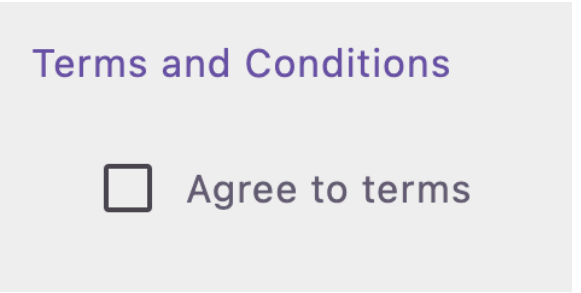


Figure 2: Reply sample app light color scheme. Courtesy of Material 3 Design.

MaterialTheme

Applying Colours




```
MaterialTheme()
{ /*  ... removed ...  */
  Text(
      text = "Terms and Conditions",
      color = MaterialTheme.colorScheme.primary
  )
  Row(
      modifier = Modifier.fillMaxWidth().padding(16.dp),
      verticalAlignment = Alignment.CenterVertically
  ) {
      Checkbox(checked = isChecked,
          onCheckedChange = { isChecked = it }
      )
      Text(
          text = "Agree to terms",
          color = MaterialTheme.colorScheme.secondary
      )
  }
}
```

MaterialTheme

Typography

Material Design 3 defines a type scale. It provides named semantic levels, akin to other typographic systems.

Names:

- `display`, `headline`, `title`, `body`, and `label`

Sizes:

- `large`, `medium`, and `small` sizes for each named group.

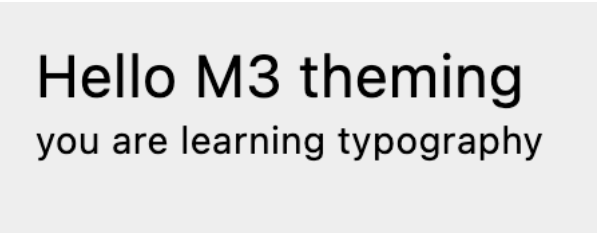
Figure 3: Material 3 Typography Scale.
Courtesy of Material Design 3.

MaterialTheme

Applying Text Styles

Apply text from your theme using the

```
fun Typography() {
    MaterialTheme() {
        Column(modifier = Modifier.fillMaxHeight().padding(16.dp)) {
            Text(
                text = "Hello M3 theming",
                style = MaterialTheme.typography.titleLarge
            )
            Text(
                text = "you are learning typography",
                style = MaterialTheme.typography.bodyMedium
            )
        }
    }
}
```



MaterialTheme

Customizing Text Styles

You can customize a Text Style by defining the element that you wish to customize, and then passing that into your theme.

```
bodyLarge = TextStyle(
    fontWeight = FontWeight.Normal,
    fontFamily = FontFamily.SansSerif,
    fontStyle = FontStyle.Italic,
    fontSize = 16.sp,
    lineHeight = 24.sp,
    letterSpacing = 0.15.sp,
    baselineShift = BaselineShift.Subscript
)

MaterialTheme(
    typography = replyTypography,
) {
    // M3 app Content
}
```

Theme Builder

You can customize all of these properties into a custom theme, which can be used directly in your project.

To customize colors, or create a dynamic color scheme, use the Material 3 Theme Builder.

From that site:

- Choose your colors
- Choose your fonts
- Export code that you can use in your project

In this example, our theme is exported as:

- `Color.kt` - color Constants
- `Type.kt` - fonts and typography
- `Theme.kt` - custom theme

Figure 4: Constants are exported as resources that you can use directly in your project.

Theme Builder

Color.kt

```
  val primaryLight = Color(0xFF415F91)
  val onPrimaryLight = Color(0xFFFFFFFF)
  ...
```

Typography.kt

```
val AppTypography = Typography(
    displayLarge = baseline.displayLarge.copy(
        fontFamily = displayFontFamily),
    ...
```

Theme.kt

```
  @Composable
  fun AppTheme(
      darkTheme: Boolean = isSystemInDarkTheme(),
      dynamicColor: Boolean = true,
      content: @Composable() () → Unit
  )
```

Bibliography

[1] T. Fessenden, “Design Systems 101.” [Online]. Available: https://www.nngroup.com/articles/design-systems-101/