Lecture 08

Tailwind CSS for React Native

i.e. NativeWind

Contents

- Get started with Tailwind CSS and NativeWind
- Installing & Configuring NativeWind
- Styling with utility classes
  - State variants
  - Media queries & breakpoints
  - Using arbitrary values
- Examples

What are TailwindCSS and NativeWind?

- TailwindCSS is a CSS framework originally developed for web.
  - It integrates well with React.js
- It offers low-level utility classes that can be used to build responsive UI more quickly.
- Nativewind is the React Native library that lets you use TailwindCSS in mobile app.

Advantages of Tailwind CSS

- **Fast Development**: No need to write custom CSS.
- **Flexible**: Allows direct customization within HTML/JSX.
- **Optimized Performance**: Generates a minimal CSS file for projects.
- **Easily Extendable**: Custom themes and configurations can be applied effortlessly.

Comparing TailwindCSS and Bootstrap

- **Similarities**
  - **Utility-First Approach**: low-level utility classes (e.g. `flex`, `text-center`, `bg-blue-500`…)
- **Differences**
  - **Bootstrap**: build UI faster, better for prototype
    - Comes with many prebuilt components like buttons, dropdowns, modals… with default visual style → build functional interfaces with <u>minimal effort</u>.
  - **TailwindCSS**: fully customizable, better performance
    - Build UI from scratch, no pre-built components or default theme.
    - Load only necessary styles.

About NativeWind

- Styled components can be shared between all React Native platforms, using the best style engine for that platform.
- NativeWind aims to bridge the gap of doing styling between web and mobile:
  - Traditionally, CSS style sheets are used on web and `StyleSheet.create` is used for mobile.
  - NativeWind wants to provide a consistent styling experience across all platforms: improving developer experience, component performance and code maintainability.
- NativeWind has lots of modern features
  - See https://www.nativewind.dev/overview

How NativeWind works?

NativeWind works by:

1. Scanning all components for class names
2. Generating the corresponding styles
3. Applying generated styles to the components

Benefits of NativeWind

- Allows you to focus on building your UI instead of creating a custom styling system.
- On the web, it reuses the existing Tailwind CSS stylesheet, avoiding runtime CSS injection.
  - This improves initial page load performance and supports Server-Side Rendering (SSR).

Installation

- You will need to install `nativewind` and its peer dependencies:

```
           npm install nativewind@2.0.11 tailwindcss@3.2.2
(optional) npm install react-native-reanimated
(optional) npm install react-native-safe-area-context
```

- **Note:** Installing the specific versions above is important since the latest version is not stable yet (as of March 2025).

Setting up NativeWind

- Run `npx tailwindcss init` to create `tailwind.config.js`
- Add the paths to all of your component files in your `tailwind.config.js` file:

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    "./TailwindApp.js",
    "./components/*.js",
    "./screens/*.js"
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Adding the Babel preset

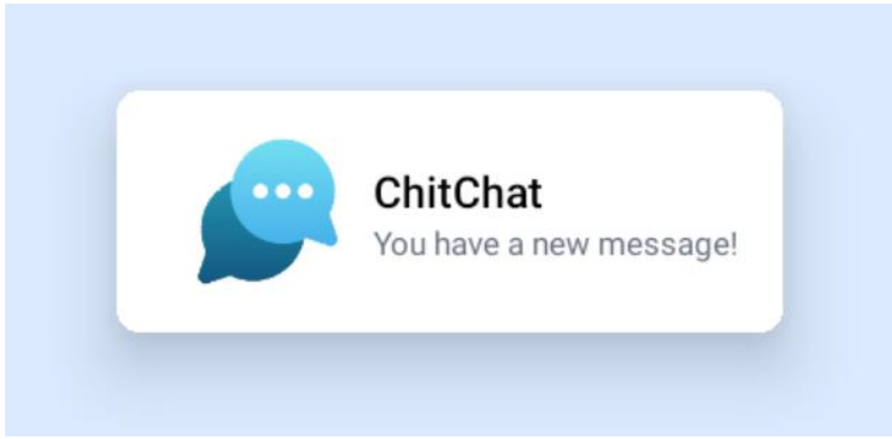
- This step ensures that NativeWind works correctly in React Native by transforming JSX syntax to support Tailwind CSS-style `className`.
- Create the file: `babel.config.js`

```
module.exports = {
    presets: ["babel-preset-expo"],
    plugins: ["nativewind/babel"],
};
```

Styling with utility classes

- You style things with Tailwind by combining many single-purpose presentational classes (utility classes) directly in your markup:

```
<View className="flex-1 justify-center items-center bg-blue-100">
    <View className="flex-row items-center shadow-2xl shadow-gray-900
                    gap-x-4 rounded-xl bg-white p-6">
        <View>
            <Image className="w-20 h-20" resizeMode="contain"
                source={require('./assets/chitchat.png')} />
        </View>
        <View>
            <Text className="text-xl font-medium text-black">
                ChitChat
            </Text>
            <Text className="text-gray-500">You have a new message!</Text>
        </View>
    </View>
</View>
```


ChitChat
You have a new message!

Styling with utility classes

- For example, in the UI above we've used:
  - The `display` and `padding` utilities (`flex`, `flex-row`, and `p-6`) to control the overall layout.
  - The `justify-center` and `items-center` utilities to center the box inside the container.
  - The `background-color`, `border-radius`, and `box-shadow` utilities (`bg-white`, `rounded-xl`, and `shadow-2xl`, `shadow-gray-900`) to style the box's appearance.

Styling with utility classes

- The `width` and `height` utilities (`w-20 h-20`) to set the width and height of the logo image.
- The `gap` utilities (`gap-x-4`) to handle the spacing between the logo and the text.
- The `font-size`, `color`, and `font-weight` utilities (`text-xl`, `text-black`, `font-medium`, etc) to style the card texts.

Key Benefits of Utility-First Styling

- **Faster Development** – Eliminates the need to create custom class names or switch between StyleSheet and JSX.
- **Safer Changes** – Utility classes apply styles only to specific elements, avoiding unintended side effects.
- **Easier Maintenance** – Updating styles involves modifying class names directly in JSX, rather than managing complex StyleSheet objects.
- **More Portable Code** – UI components can be easily copied across projects (e.g. between React and React Native).
- **Small Memory Usage** – Only necessary styles are generated.

Why not just use inline styles?

- One may wonder: “**isn’t this just inline styles?**”
  - In some ways it is!
- However, utility classes have many important advantages over inline styles:
  - **Designing with constraints** — using inline styles, every value is a magic number. With utilities, you’re choosing styles from a predefined design system, which makes it much easier to build visually consistent UIs.
  - **Focus, press and other states** — inline styles can’t target states like onPressIn or focus, but Tailwind’s *state variants* make it easy to style those states with utility classes.
  - **Media queries** — you can’t use media queries in inline styles, but you can use Tailwind’s *responsive variants* to build fully responsive interfaces easily.

Core Tailwind/NativeWind Features

- **Variants for states** (e.g., focus:bg-orange-200)
- **Responsive design** using prefixes like sm:grid-cols-3
- **Dark mode** styling with dark:bg-gray-800
- **Arbitrary values** (e.g., bg-[#316ff6]) for custom styles

State variants

- Below is a `TextInput` component with different styles when is it focused and not focused:

```
<TextInput
    className="border p-2 rounded
    focus:border-blue-500
    focus:outline-none"
    placeholder="Enter text..."
/>
```

Media queries and breakpoints

- Just like state variants, you can style elements at different breakpoints by prefixing any utility with the breakpoint where you want that style to apply.

```
<View className="flex flex-row sm:flex-col">
    <Text>A</Text>
    <Text>B</Text>
    <Text>C</Text>
</View>
```

Using arbitrary values

- Many utilities in Tailwind are driven by **theme variables**, like `bg-blue-500`, `text-xl`, and `shadow-md`, which map to underlying configurations.
- When you need to use a one-off value outside of your theme, use the special square bracket syntax for specifying arbitrary values.

```
<Pressable className="bg-[#316ff6] ...">
    <Text className="text-[#ffe]">
        Sign in with Facebook
    </Text>
</Pressable>
```