Complex UI with Flutter Web

Fundamentals

Before we get into, the flutter side of things, let's take a look at some of the coolest websites, and what makes them so well made.

1. https://camillemormal.com/
2. https://advanced.team/

And perhaps a very well known one,

3. https://github.com/

One of the most remarkable, easy and yet impressive parts of these websites are that irrespective of the screen size you look at it from, they are built for that. But that's nothing new right? Responsive design has existed since the dawn of time.

There's something else about these websites. These are the kind of web designs UI designers would recommend, and developers would scream "Nightmare" at.

Flutter Web

<quick brief of flutter for those who don't know about it>

<benefits like cross platform and open source>

<how web didn't hit stable for a long time.>

Reactive Design - The Magic

We will be using a package for this called provider : https://pub.dev/packages/provider

We can do this entire thing without provider, by using Valuelistenables and ValueNotifiers, but that's really messy, and provider allows us to propogate the changes down the widget tree with use.

The idea for this began when I was researching ways to build complex UI, but couldn't find much beyond "https://pub.dev/packages/responsive_builder". In my opinion, responsive_builder wasn't elaborate enough or good enough for my use case.

Template

So first we begin by quickly building an app, with the following template

Scaffold
├── Row
│ ├── Navigation Rail
│ ├── Expanded with a dummy container child

<Presentation demos>

This is the base. The expanded will contain all the child data.

Immediately this looks great for a basic website, replace the expanded with a bunch of switch statements, and fill in widget details, and you have a basic blog.

Now how can we make, this better?

Replace the container with a column and let the column have 10 containers with the size of window.

We can get window size by: **final screenSize = MediaQuery.of(context).size;**

Call screenSize.height for the height, and width respectively **.**

A column overflows, so let's wrap it with a SingleChildScrollView, and make it's axis vertical so we can scroll vertically, and add a ScrollController.

Declare the scroll controller above, and give the navigation rail destinations control to change it.

Now, we can scroll through the website by simply click on the menu options.

<demo changes here>

This website is already looking better, let's go ahead and use the provider here, by wrapping the Column with a change notifier provider

```
├── Row
│   ├── Navigation Rail
│   ├── Expanded
│   │   ├── SingleChildScrollView
│   │   │   ├── ChangeNotifierProvider
│   │   │   │   ├── Column
│   │   │   │   │   ├── Pages….
```

<Insert demo here>

Now instead of Pages being containers and cluttering up our code, let's declare a new stateless widget for all the pages separately.

The default structure for it will be

Two required variables, height and width (sometimes we want pages with different height and width)

All the page widgets will be wrapped with a Consumer<ScrollController> at the highest level.

```
├── Consumer<ScrollController>
│   ├── Expanded
│   │   ├── <page widgets>
```

Now let's propagate our page with some sample widgets.

<Insert demo here>

Perfect, everything is ready, now how do we add that magic touch of scroll and hover.

Scroll Animations

So the reason we added all these items (Consumer, and ChangeNotifier) are to propagate the scrollcontroller position down the widget tree.

Combining a little bit of math, curves, AnimatedContainer and Matrix4, we can achieve results like this.

<demo code>

<demo result>

The idea is to scroll everything on screen in a different way, different directions, different speeds. Sometimes we want to show widgets simply based off what is visible on the screen.

Using basic linear math, we can achieve all this, and some more fancy animations like this.

<demo results>

Hover Animations

Hover animations are fairly straightforward compared to what we dealt with for scrolling. We're still going to use AnimatedContainer and matrix4 values, but instead of all that scroll controller math, we just pop in a hover state. By using a mouse region.

<insert demo with sample code snippets>

Idle Animations

Now some websites also have something I like to call Idle animations. Where regardless of what you do, there are animations on screen. I am personally not a fan of it as it draws a lot of resources. But used correctly they can be very appealing.

A quick sample for it is >> <insert demo>

We can achieve this by using basic gradients and GradientRotation under the transform of LinearGradient this way.

Something like this could be created : https://codepen.io/gayane-gasparyan/pen/jOmaBQK.

Flutter Drawbacks

- A little bit about how new it is
- Growing Community, but still buggy
- Everything is rendered on a canvas so bad loading speeds
- Little to no SEO because of the same reason.