gatsby-plugin-layout

This plugin enables adding components which live above the page components and persist across page changes.

This can be helpful for:

- Persisting layout between page changes for e.g. animating navigation
- Storing state when navigating pages
- Custom error handling using componentDidCatch
- Inject additional data into pages using React Context.

This plugin reimplements the behavior of layout components in `gatsby@1`, which was removed in version 2.

Install

```
npm install gatsby-plugin-layout
```

How to use

Add the plugin to your `gatsby-config.js`:

By default plugin will try to use Layout component located in `src/layouts/index.js` (same as Gatsby v1)

```
module.exports = {
  plugins: [`gatsby-plugin-layout`],
}
```

If you prefer to keep layout in different place, you can use `component` option:

```
module.exports = {
  plugins: [
    {
      resolve: `gatsby-plugin-layout`,
      options: {
        component: require.resolve(`./relative/path/to/layout/component`),
      },
    },
  ],
}
```

Once the plugin is added, you don't need to manually wrap your pages with the Layout component. The plugin does this automatically.

Why would you want to reimplement the V1 layout behavior?

There are a few scenarios where it makes sense to reimplement the V1 layout handling:

1. You have a large or complex V1 site and refactoring to the new layout component is not feasible
2. Your site uses page transitions or other transitions that break if the layout component is unmounted and remounted when routes change
3. Your site attaches global state in the layout that doesn't persist if the component is unmounted and remounted

How layouts worked in version 1

In the original implementation, the layout component was wrapped around the outside of the page component, which, in pseudo-code, looked something like this:

```
<Root>
  <Layout>
    {/* layout is not affected when the page template changes */}
    <PageElement>{/* page content here */}</PageElement>
  </Layout>
</Root>
```

This meant that the layout component could manage things like transitions and persistent state without any special workarounds, because it never rerendered.

How layouts work in version 2

In version 2, the layout component is no longer special, and it's included in every page that wants to display it. This means that it *does* rerender on every route change:

```
<Root>
  <PageElement>
    {/* layout will rerender each time the page template changes */}
    <Layout>{/* page content here */}</Layout>
  </PageElement>
</Root>
```

This can make it complicated to support transitions or state without using the `wrapPageElement` browser API (and the SSR equivalent). This plugin implements those APIs for you, which reimplements the behavior of Gatsby V1.

Troubleshooting

Passing data from Layout to Page / from Page to Layout

Use React Context to pass data both ways.

For example you can use this boilerplate:

```
// Context.js
import React from "react"

const defaultContextValue = {
  data: {
    // set your initial data shape here
    menuOpen: false,
  },
  set: () => {},
}

const { Provider, Consumer } = React.createContext(defaultContextValue)

class ContextProviderComponent extends React.Component {
  constructor() {
    super()

    this.setData = this.setData.bind(this)
    this.state = {
      ...defaultContextValue,
      set: this.setData,
    }
  }

  setData(newData) {
    this.setState(state => ({
      data: {
        ...state.data,
        ...newData,
      },
    }))
  }

  render() {
    return <Provider value={this.state}>{this.props.children}</Provider>
  }
}

export { Consumer as default, ContextProviderComponent }
```

Use Provider in Layout Component:

```
import { ContextProviderComponent } from "./Context"

export default ({ children }) => (
  <ContextProviderComponent>
    <Header />
    {children}
    <Footer />
  </ContextProviderComponent>
)
```

And then you can use it anywhere:

- To read state:

```
import ContextConsumer from "./Context"

const ComponentThatReadState = () => (
  <ContextConsumer>
    {({ data }) => {
      data.menuOpen ? <Menu /> : null
    }}
  </ContextConsumer>
)
```

- To read and set state:

```
import ContextConsumer from "./Context"

const ComponentThatChangeState = () => (
  <ContextConsumer>
    {({ data, set }) => (
      <div onClick={() => set({ menuOpen: !data.menuOpen })}>
        {data.menuOpen ? `Opened Menu` : `Closed Menu`}
      </div>
    )}
  </ContextConsumer>
)
```

Handling multiple layouts

If you want to use different layouts for different pages, you can pass this information in the context of the pages you create, and then conditionally render in your layout file.

In `gatsby-node.js`:

```
exports.onCreatePage = ({ page, actions }) => {
  const { createPage } = actions
```

```
  if (page.path.match(/special-page/)) {
    page.context.layout = "special"
    createPage(page)
  }
}
```

And then in `src/layouts/index.js`:

```
export default ({ children, pageContext }) => {
  if (pageContext.layout === "special") {
    return <AlternativeLayout>{children}</AlternativeLayout>
  }
  return <RegularLayout>{children}</RegularLayout>
}
```