Debugging Asynchronous Lifecycle Methods

Various lifecycle methods (see: Gatsby Node APIs) within Gatsby are presumed to be asynchronous. In other words, these methods can *eventually* resolve to a value and this value is a `Promise`. You wait for the `Promise` to resolve, and then mark the lifecycle method as completed when it does.

In the context of Gatsby, this means that if you are invoking asynchronous functionality (e.g. data requests, `graphql` calls, etc.) and not correctly returning the Promise an internal issue can arise where the result of those call(s) happens *after* the lifecycle method has already been marked as completed. Consider an example:

```
exports.createPages = async function ({ actions, graphql }) {
  // highlight-start
  graphql(`
    {
      allMarkdownRemark {
        edges {
          node {
            fields {
              slug
            }
          }
        }
      }
    }
  `).then(res => {
    res.data.allMarkdownRemark.edges.forEach(edge => {
      const slug = edge.node.fields.slug
      actions.createPage({
        path: slug,
        component: require.resolve(`./src/templates/post.js`),
        context: { slug },
      })
    })
  })
  // highlight-end
}
```

Can you spot the error? In this case, an asynchronous action (`graphql`) was invoked but this asynchronous action was neither `return`ed nor `await`ed from `createPages`. This means that the lifecycle method will be marked as complete before it's actually completed–which leads to missing data errors and other hard-to-debug errors.

The fix is surprisingly simple–just one line to change!

```
exports.createPages = async function ({ actions, graphql }) {
  // highlight-next-line
  await graphql(`
    {
      allMarkdownRemark {
        edges {
          node {
            fields {
              slug
            }
          }
        }
      }
    }
  `).then(res => {
    res.data.allMarkdownRemark.edges.forEach(edge => {
      const slug = edge.node.fields.slug
      actions.createPage({
        path: slug,
        component: require.resolve(`./src/templates/post.js`),
        context: { slug },
      })
    })
  })
}
```

Best Practices

Use `async` / `await`

With Node 8, Node is able to natively interpret `async` functions. This lets you write asynchronous code as if it were synchronous! This can clean up code that previously was using a `Promise` chain and tends to be a little simpler to understand!

Use `Promise.all` if necessary

`Promise.all` wraps up *multiple* asynchronous actions and resolves when *each* have completed. This can be especially helpful if you're pulling from multiple

data sources or abstracted some code that returns a Promise into a helper. For instance, consider the following code:

```
const fetch = require("node-fetch")

const getJSON = uri => fetch(uri).then(response => response.json())

exports.createPages = async function ({ actions, graphql }) {
  // highlight-start
  const [pokemonData, rickAndMortyData] = await Promise.all([
    getJSON("https://some-rest-api.com/pokemon"),
    getJSON("https://some-rest-api.com/rick-and-morty"),
  ])
  // highlight-end

  // use data to create pages with actions.createPage
}
```