REST Easy with Apollo Client

GraphQL is taking the web development world by storm, and along with it a whole new set of tools, practices, and packages to learn. During my experience learning GraphQL I've encountered several hurdles and questions, and from talking to others I gather that these are common for developers trying to learn the technology:

1. Where the heck do I start? Schemas, queries, resolvers, mutations, Apollo, Relay...huh?
2. I work on the front-end and have little or no control over my backend. Can I even use GraphQL?
3. I've heard about this GraphQL thing and want to try it out. What's an easy way to do give GraphQL a shot?
4. Rewriting our entire backend in GraphQL is unfeasible - I've got code to ship and deadlines to meet. Can I use GraphQL without committing to massive changes?

Apollo Client, a toolset by the Meteor Development Group that manages your application's data via the power of GraphQL, is a popular way to integrate GraphQL into front-end applications. One of the Apollo platform's biggest strengths is its broad set of *links* that extends the Client's capability - kind of like middleware for GraphQL applications. Links can be chained together to shape your data exactly to your specifications, and offer powerful tools for common needs like error handling and state management.

One of Apollo Client's links offers solutions to all four of the questions above: *apollo-link-rest*, which allows existing REST API endpoints to be communicated with via GraphQL. This means you can start using GraphQL on the front-end, through Apollo Client, without any need for a GraphQL server on the back-end or any need to modify the back-end at all. It's a great way to try GraphQL without getting overwhelmed and for front-end developers who want to see what all the fuss is about. Whether or not it's suitable for full-scale production applications will depend on the team and application - if you want to have a fully GraphQL-powered application, it's generally recommended to leverage GraphQL on both ends; in this situation, apollo-link-rest offers a useful intermediary that can help the transition from a traditional REST setup to a GraphQL one (e.g. a front-end team doesn't have to wait for a GraphQL back-end to be functional).

Without further adue, let's take a dive into Apollo Client and see how apollo-link-rest can be used to manage data from a REST API in a React application. Our API will come from JSONPlaceholder, which offers a mock REST API with everything you'd expect for a CRUD app, minus persistent storage. We'll use create-react-app to get our project up-and-running, and explore how Apollo Client works with React, as well as noting some issues with apollo-link-rest - as the project maintainers note, it's a library that's *under active development*.

The finished repo for this project can be found at https://github.com/garethpbk/apollo-rest-example.

Creating The Project and Adding Dependencies

The first step is to create a new React App - commands here will use yarn:

```
yarn create react-app apollo-rest-example
```

Next, there's a few dependencies to add:

1. `apollo-client` ..*the head honcho, the actual GraphQL client
2. `apollo-cache-inmemory` ..*the recommended data store for Apollo Client 2.0

3. apollo-link ..*allows usage of all the cool links to extend the client
4. apollo-link-rest ..*the specific rest link
5. graphql ..*the JavaScript implementation of the GraphQL specification
6. graphql-anywhere ..*allows GraphQL queries to be run anywhere, without a server or schema
7. graphql.macro ..*for importing .graphql files into .js files
8. react-apollo ..*the React implementation of Apollo Client
9. @reach/router ..*for client-side routing, of course!

Let's add them all:

```
yarn add apollo-client apollo-cache-inmemory apollo-link apollo-link-rest graphql
graphql-anywhere graphql.macro react-apollo @reach/router
```

Notes: `graphql-tag` can be used in place of