What is React JS?

ReactJS is basically is an open-source JavaScript library which is **used** for building user interfaces. It's **used** for handling view layer for web and mobile apps.

Why should we use React?

React also allows us to create **reusable UI components.**...
The main purpose of **React** is to be **fast, scalable, and simple**. Plus, it works with something called **Babel… Babel** allows your javascript to run on **any** browser.

React is open-source, has easy to read Docs and there is a large community or react developers out there. Many modern websites use **React in production** (to see if a website uses react you can add an **extension to chrome** and it will tell you if the page was built using react!)

Check out some **cool examples** of what people built using React:

- https://reactjsexample.com/
- https://github.com/Hermanya/awesome-react-bootstrap-components

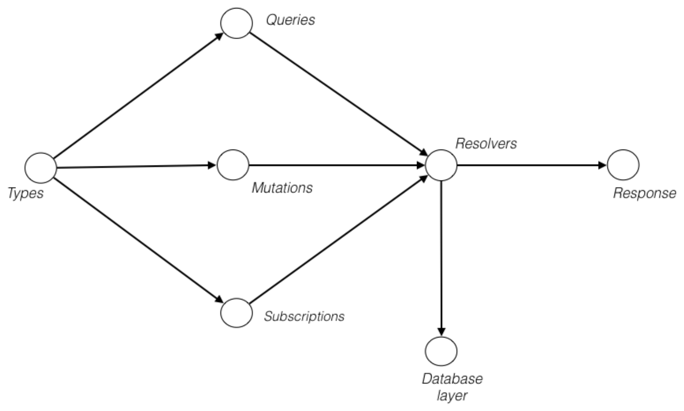
Since we'll want to display **dynamic data** with **React**, we can use **Props** and you can think of **props** as just an **object**, where the **keys** in the object are each of the **components properties**. Props and State can be confusing if you don't know how to use them properly, so i'll leave this link here on how to get a good idea of **how props and state work**:
https://reactjs.org/docs/thinking-in-react.html

React-Bootstrap, what is it?

It is a complete re-implementation of the **Bootstrap** components using **React**. It has no dependency on either **bootstrap**.js or jQuery. If you have a **React** setup and **React**-**Bootstrap** installed you have everything you need.

See more here: **https://react-bootstrap.github.io/components/alerts**

GraphQL



A bit about *what **GraphQL** is* and why its cool:

- GraphQL the QL meaning **query language** but with *declarative* data fetching
- Allows you to query your backend for ***just*** the data you need
- In other words, GraphQL lets you send the fields you want (make query) → The backend (SQL, DB, Django ect.) fills those fields and returns those **exact** values. Sweet right?
- It also eliminates any overfetching or underfetching (which will impress future employers if you're aware of this)

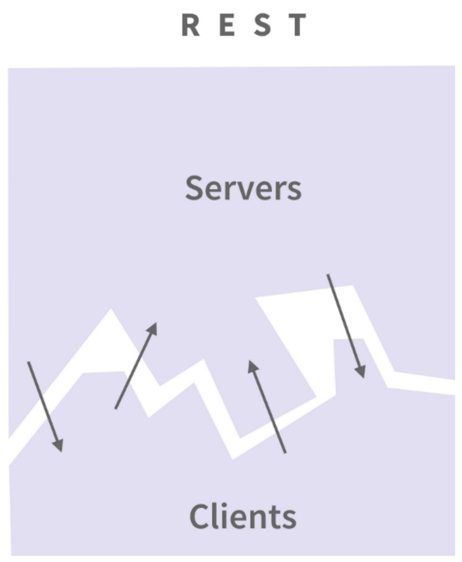
Main building blocks: ***Schemas*** ***Queries*** ***Mutations*** ***Resolvers***

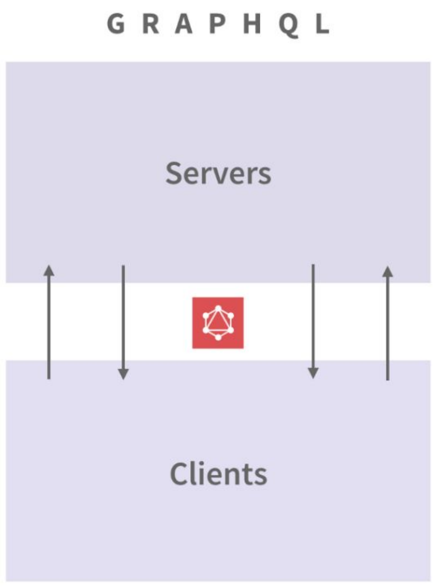
In **React**, graphql client uses **higher order components** (hoc) to get the data under the hood. (hoc's are a technique used in component reuse https://reactjs.org/docs/higher-order-components.html)

A GraphQL Client sends queries directly, all you have to do is write a query declaring the data requirements and the GraphQL client takes care of the **request** and **response handling** (which is why graphql is awesome)

Fetching Data with Queries

Recall that typically when working with REST APIs, data is loaded from multiple specific endpoints. Each endpoint has a clearly defined structure of the information that it returns. This means that the data requirements of a client are effectively encoded in the URL that it connects to.





Instead of having ***multiple*** endpoints that return fixed data structures, GraphQL APIs typically only expose a ***single endpoint***. This works because the structure of the data that's returned is *not* fixed. Instead, it's completely flexible and lets the client decide what data is actually needed.

That means that the client needs to send more ***information*** to the server to express its data needs - this is what forms the **query.**

Basic Queries

```
Example
{
  allPersons {
    name
  }
}
```

```
Example response {
"allPersons":[
{ "name": "Johnny" },
{ "name": "Sarah" },
{ "name": "Alice" }
]}
```

The `allPersons` field in this query is called the `root field` of the query. Everything that follows the root field, is called the **payload** of the query. The only field that's specified in this query's payload is name.

Summary: A GraphQL Query is an object where you provide it with only the properties that you want. So its like you only provide it with the left hand side stuff

Schemas

Schemas: Specify what the API can do and defines how the clients can request data, it's like a contract between server and client. Schemas are mostly composed of types but the schema for an API there are the special root types:

type Query { ... }
type Mutation { ... }
type Subscription { ... }

The Query, Mutation, and Subscription types are the ***entry points*** for the requests sent by the client

How are we going to track the users location? Caching query results

We want to maintain a cache of the previously fetched data as having information cached locally is essential to provide a fluent user experience.

In graphql caching, we want to **normalize** the data beforehand so that the potentially nested query gets **flattened** and the store only contains individual records that can be accessed with a globally unique id.

More to come… hopefully ive sold you on the idea of using graphql and react :)