API Caching

- **What is API caching**

  API caching using in-memory cache. Cache is a technique used to store responses from API calls in memory, reducing the need to fetch the same data from the server repeatedly. This improves performance by decreasing latency and server load.

  Libraries such as Apollo, ReactQuery, SWR etc provide functions and hooks to utilise the caching capabilities.

- **Libraries and their handling of API Caching**

  - ReactQuery (TanStack Query)

    Read the following docs to understand more about it

    https://tanstack.com/query/latest/docs/framework/react/guides/caching

```
import {
  QueryClient,
  QueryClientProvider,
  useQuery,
} from '@tanstack/react-query'

// Create a query client instance
const queryClient = new QueryClient()

export default function App() {
  return (
      // Provide the QueryClient to the app using QueryCli
    <QueryClientProvider client={queryClient}>
      <Example />
    </QueryClientProvider>
  )
}
```

```
function Example() {
    // useQuery hook to fetch data
  const { isPending, error, data } = useQuery({
      // Unique key for the query
    queryKey: ['repoData'],
    // Function to fetch the data
    queryFn: () =>
      fetch('https://api.github.com/repos/TanStack/query'
        res.json(),
      ),
  })
    // handle the loadin state
  if (isPending) return 'Loading...'

    // handle the error state
  if (error) return 'An error has occurred: ' + error.mess

    // handle the response data
  return (
    <div>
      <h1>{data.name}</h1>
      <p>{data.description}</p>
      <strong>👀 {data.subscribers_count}</strong>{' '}
      <strong>✨ {data.stargazers_count}</strong>{' '}
      <strong>🍴 {data.forks_count}</strong>
    </div>
  )
}
```

▼ SWR

By default, SWR uses a global cache to store and share data across all components. But you can also customise this behaviour with the `provider` option of `SWRConfig`.

Cache providers are intended to enable SWR with more customised storages.

Cache Provider

A cache provider is Map-like object which matches the following TypeScript definition (which can be imported from `swr`):

```
interface Cache<Data> {
    get(key: string): Data | undefined
    set(key: string, value: Data): void
    delete(key: string): void
    keys(): IterableIterator<string>
}
```

Create Cache Provider

The `provider` option of `SWRConfig` receives a function that returns a cache provider. The provider will then be used by all SWR hooks inside that `SWRConfig` boundary. For example:

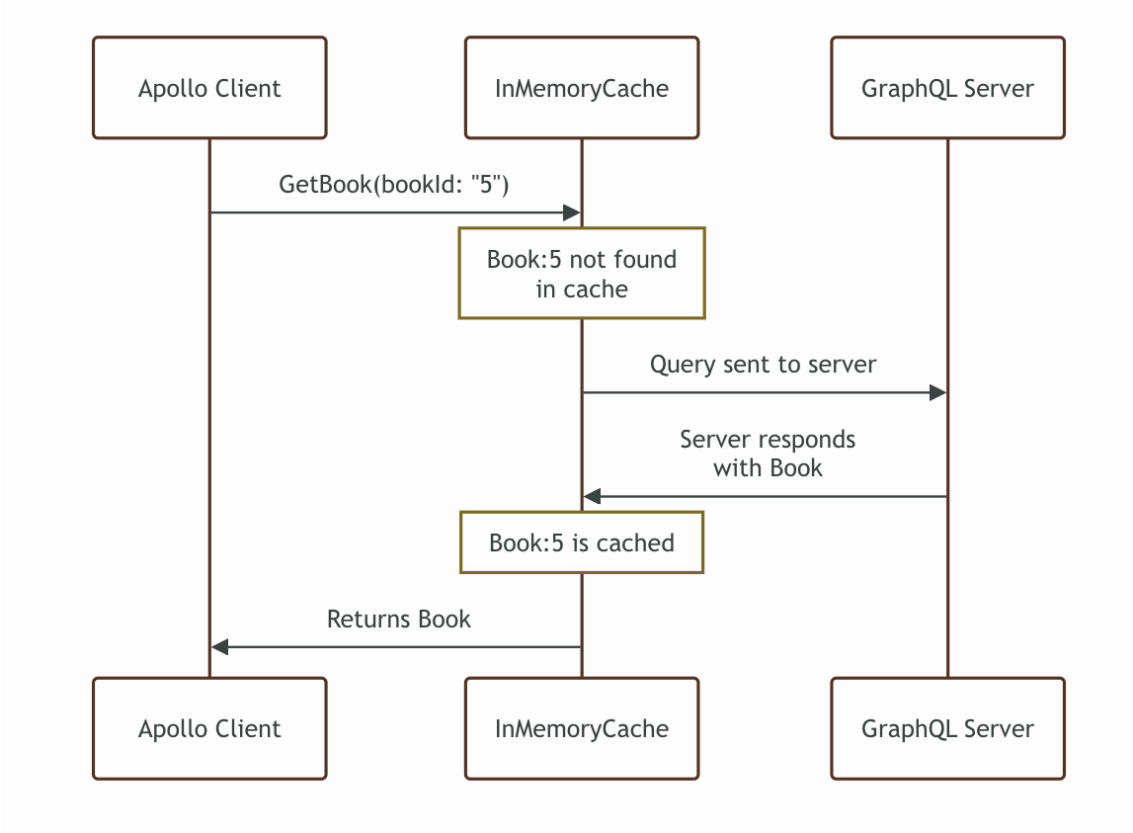
```
import useSWR, { SWRConfig } from 'swr'

function App() {
    return (
        <SWRConfig value={{ provider: () => new Map()}}>
            <Page/>
        </SWRConfig>
    )
}
```

for more information visit : https://swr.vercel.app/docs/advanced/cache

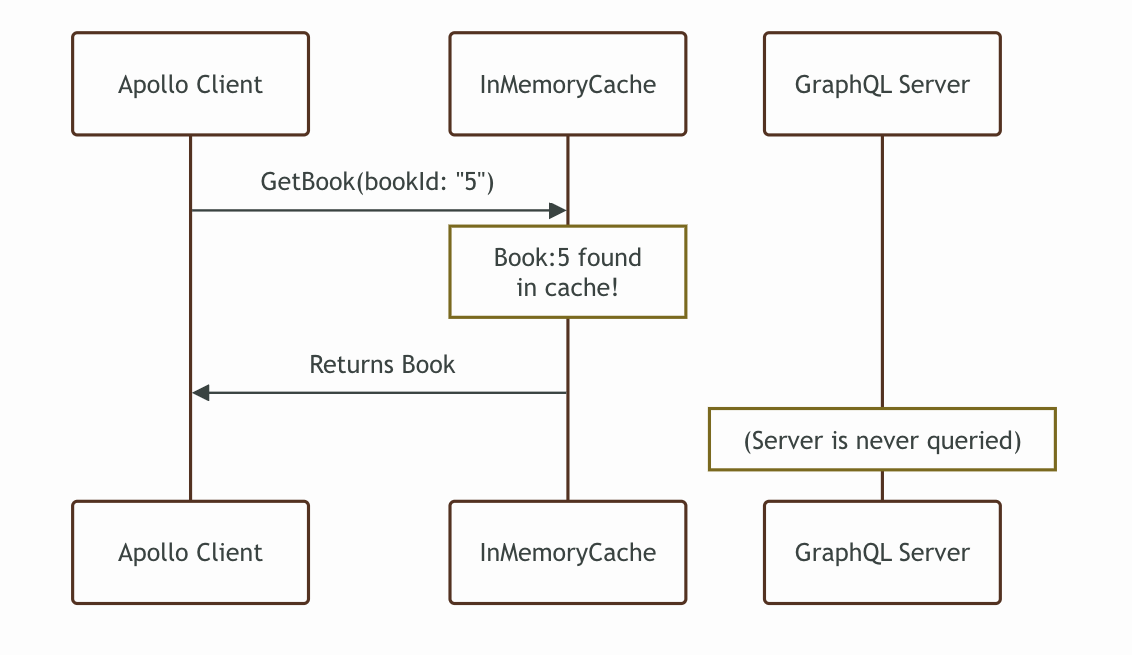
▼ Apollo Client

Apollo Client stores the results of your GraphQL queries in a local, normalized, in-memory cache. This enables Apollo Client to respond almost immediately to queries for already-cached data, without even sending a network request.

For example, the *first* time your app executes a `GetBook` query for a `Book` object with id `5`, the flow looks like this:



And each *later* time your app executes `GetBook` for that same object, the flow looks like this instead:



The Apollo Client cache is highly configurable. You can customize its behavior for individual types and fields in your schema, and you can even use it to store and interact with local data that *isn't* fetched from your GraphQL server.

Setting a fetch policy

By default, the `useQuery` hook checks the Apollo Client cache to see if all the data you requested is already available locally. If all data *is* available locally, `useQuery` returns that data and *doesn't* query your GraphQL server. This `cache-first` policy is Apollo Client's default **fetch policy**.

You can specify a different fetch policy for a given query. To do so, include the `fetchPolicy` option in your call to `useQuery`:

```
const { loading, error, data } = useQuery(GET_DOGS, {
  fetchPolicy: 'network-only', // Doesn't check cache bef
});
```

learn more about fetch policies here: https://www.apollographql.com/docs/react/data/queries/#setting-a-fetch-policy

▼ **Network Policies**

▼ cache-first

If the data is available in the cache, it will use that instead of making a network request. If the data is not in the cache, then it will fetch the data from the network.

▼ network-only

The network-only policy ignores the cache entirely and always makes a network request to fetch the data.

▼ cache-and-network

This policy uses the cache first (if available) and updates the client and simultaneously makes a network request to fetch the latest data. Once the data is fetched, it updates the cache and re-renders the component with the new data.

▼ cache-last

The data is first fetched from the network call. If the network request fails, then only the cache is utilised. cache-last acts like the last measure

▼ no-cache

This policy bypasses the cache completely. It neither reads from or writes to the cache. Each data fetch is independent and always goes to the network. network-only may write to the cache but no-cache does not interact with cache at all