|  |  |  |
| --- | --- | --- |
| **Criteria** | **GraphQL** | **REST** |
| Basic Jargons | Query/Data | Request/Response |
| Over-Fetching /Under-Fetching | Gets just the required query data | Data Over-Fetching/Under-Fetching in client |
| Data Bandwidth | Ideal for small client devices | Higher bandwidth due to possibility of Data Over-Fetching |
| Data-Fetch style | Declarative Data-Fetching    In GraphQL, just two steps, other low-level details like sending http request and receiving response is all handled by GraphQL client library like Apollo | Imperative Data-Fetching |
| Method specification | Mutation | POST/PUT/PATCH/DELETE |
| Server-side event | Subscription | Equivalent could be client-side call-back |
| Complex data-relationships | Brings in one trip | Might need multiple trips |
| Highly performant UIs | Since needed data subset available for client | At least multiple round-trips to accomplish a “seemingly” performant UI |
| Domain knowledge | Needs careful design thinking to build the API in server-side | Not so – since the data requirement is distributed across many endpoints |
| Strongly typed system | Schema Definition Language and Resolvers | Not natively supported but can be accomplished with JSON schema |
| Error handling | Not so good due to generic http responses | Good error handling |
| Field-level data fetching | Resolvers | Business logic applied to response object properties |
| Web Cache | Caching individual queries is difficult since all queries are handled by one POST endpoint | Better caching since each request is separate endpoint |
| File Upload | Not in-built, look for libraries | Supported through POST |
| Architecture | Micro-services / composable | Resource-based / CRUD |
| Data structure | Hierarchical | Flat |
| Strength | Client-side query | Server-side processing |
| Tech Stack | JS frameworks like NextJS | .Net Framework or JAVA esp., MVC |
| De-coupled FED | Possible due to SDL | Might need some workaround if not possible |

Over-Fetching/Under-Fetching: In an ideal REST world, an API endpoint has a request and a response. Based on the project requirements, the API developer codes the request, response objects and writes necessary logic to fetch data. Any new fields must be first added to the request and response objects by the developer. Also, at any point, there might me more or less fields than what is required. So, this is termed over-fetching and under-fetching respectively. Understandably, this process of making changes to API code will also involve new development iteration.

References:

<https://blog.logrocket.com/graphql-vs-rest-api-why-you-shouldnt-use-graphql/#:~:text=REST%20can%20do%20much%20of%20what%20GraphQL%20does,-It's%20important%20to&text=If%20you%20want%20the%20benefit,OData%20is%20a%20great%20alternative>.