GraphQL WG – April 2022

1. Agree to Membership Agreement, Participation & Contribution Guidelines and Code of Conduct (1m, Lee)
    - Specification Membership Agreement
    - Participation Guidelines
    - Contribution Guide
    - Code of Conduct
2. Introduction of attendees (5m, Lee)
3. Determine volunteers for note taking (1m, Lee)
4. Review agenda (2m, Lee)
5. Review previous meeting's action items (5m, Lee)
    - Ready for review
    - All open action items (by last update)
    - All open action items (by meeting)
6. GraphQL Conf scheduled for June 7-8 in Austin (5m, Lee)
    - Collaborator summit-style event focused on core spec and implementation work
    - Single track, 1.5 days, small setting
    - Please submit talk suggestions!
    - It will be recorded.
7. Discuss adding additional maintiners to GraphQL-JS (5m, Alex Reilly)
8. Review GraphiQL 2 design proposal (20m, Tim)
9. Discuss proposal to allow unions to declare implementation of interfaces (15m, Yaacov)
    - Spec PR: graphql/graphql-spec#939
    - graphql-js PR: graphql/graphql-js#3527
10. Defer/Stream - *NOTE: below are the potentially final two discussion items prior to advancement (!!!)*
    - Return arrays from stream payloads (15m, Rob)
        - Background: robrichard/defer-stream-wg#32
        - Goal: Confirm stream payload format.
    - Asynchronous iterators of iterables versus of items (15 min, Yaacov)
        - Background: robrichard/defer-stream-wg#38
        - Goal 1: Confirm whether AsyncGenerator payload in reference implementation should be an iterable or a single item.
        - Goal 2: Confirm whether the spec must specify the AsyncGenerator payload type used.
11. Agree to Membership Agreement, Participation & Contribution Guidelines and Code of Conduct (1m, Lee)
    - Specification Membership Agreement
    - Participation Guidelines
    - Contribution Guide
    - Code of Conduct

Determine volunteers for note taking (1m, Lee)

- Benjie
- Eloy

Review agenda (2m, Lee)

- No changes.

Review previous meeting's action items (5m, Lee)

- Ready for review
- All open action items (by last update)
- All open action items (by meeting)
- No actions.

GraphQL Conf scheduled for June 7-8 in Austin (5m, Lee)

- Collaborator summit-style event focused on core spec and implementation work
- Single track, 1.5 days, small setting
- Please submit talk suggestions!
- It will be recorded.
- Lee: this is the first time we've organized one so we're piggy-backing OpenJSWorld. We're expecting mostly remote attendance. Theme: what's the machinery that makes the GraphQL ecosystem tick. Wanted to make sure you're aware it's happening and ask for suggestions. Please submit talks! Preference: meet in person, but we're also allowing for remote delivered talks.
- Lee: because it's a joint conference you register for one of the other conferences and then there's a GraphQL dedicated space as part of the conference.
- Stephen: can we indicate that we're planning to attend the GraphQL sub-conf so we get a rough idea who'll be there?
- Lee: Unknown, will ask Brian.
- Roman: I have concerns that we're piggy-backing a JS conference, it should be more open to other languages.
- Michael: It's just GraphQL, we talk about GraphQL-Java and other technologies like at other conferences.
- Lee: it's open to all backend developers, we're not doing it frontend focussed.
- Lee: if not enough people can make it it'll be a social get together of WG folks. Would be nice for WG folks to have a get together!

Discuss adding additional maintainers to GraphQL-JS (5m, Alex Reilly)

- [Pulled - was addressed at GraphQL.js WG]

Review GraphiQL 2 design proposal (20m, Tim)

- Tim: the new GraphiQL design is a major change; we want GraphQL experts to look at it before we implement it. 5-10 minute overview.
- Conceptually we want to make it extendable - lots of people want extensions: replacing parts, doing their own docs view, etc. We want to make panels replaceable and allow plugins - provide your own editor, etc.
- When you open GraphiQL you only see what you need to see; but step by step you can enable additional things e.g. by clicking on the docs icon.
- The operation builder that OneGraph built has a tree structure, and the docs has a tree structure, so we thought we'd explore merging these. Docs and operation builder are one thing - you can open specific queries, view types, navigate, and add things to the query. When dealing with arguments, required arguments will always be shown, others can be "show more arguments".
- Clicking a specific type will open an overlay with the details about that type, and can sub-navigate through there.
- Search supports searching types, fields, but also through the descriptions.
- History displays in the same place, it's extensible and you could even use it to just select the operations to run and not have the docs.
- There's settings for theme/etc, this'll be expanded.
- Tooltips in the editor are much more powerful; moving to Monaco.
- React implementation should need much less code than it does with CodeMirror.
- Tooltips for errors.
- We have a minimal design that's suitable for embedding into docs.
- Dark mode of course!
- Drag to resize panels; enables more fields/descriptions to be visible.
- If you look at VSCode you'll see icons on the left that only change the sidebar, not the whole screen. The aim is to have the same pattern - it's either always sidebar or always whole screen. We plan to address this with a dropdown at the top where you can change between e.g. GraphQL Editor or AST Explorer.
- Variables are hidden by default, minimal design, but as you need them they appear.
- Considering plugins, on the result pane you could add more stuff to show additional details.
- Roman: regarding variables/headers - they need to be easily discoverable. It's not obvious you need to type them and have them appear. Additionally documentation on fields with arguments - quite limited space. Spec specifies it's markdown, doesn't seem to have space to show fancy markdown. Like YouTube you can see the beginning of the title but shows the hint with the full text.

- Tim: this makes sense. We should also expand the descriptions in the popup.
- Tim: for now we don't show the checkboxes in a drill-down view because it might be confusing.
- Benjie: looks amazing! Really like the cleanliness, the visual priority, etc. Regarding variables: we should be encouraging their use because static queries are generally preferred these days. Can we make the sidebar add variables when you click the args?
- Lee: it'd be really nice if you could hover over a value and you get an option to turn it into a variable. A single button push.
- Michael: quickfix in VSCode
- Tim: OneGraph lets you edit variables in sidebar; we would make this add variables so that you don't have to edit them in the sidebar to keep things simple.
- Lee: what happens once you get deeper and deeper in the query? I don't have a suggestion, but we may need to be careful with ambition.
- Michael: also, what does this mean for fragments - once you have these maybe the tree won't align so well?
- Tim: good question; we've not thought about it. Even if it's not represented on the left and you have to write it yourself, we need to keep it working.
- Matt: top level things could be hinted with an executable definition: "query from GetFilms" "fragment from FragmentUser". One thing that's hard to get at the design stage (though I'm sure this has been thought through) is writing complex schemas/queries/etc - make sure you're not blocking tasks on peripheral plugins being available. At Meta we frequently have an issue where loading GraphiQL takes a minute or two because our schema is absurd - if I can't copy/paste code and hit run… It's hard to test that in the initial stages, but making sure the plugins/components don't block each other would add a lot more value.
- Tim: virtualization may be an option here. React-scroll, etc.
- Matt: you could by default hide (...) stuff between title and releaseDate - there might be 500 fields between them, so just show the ones you've queried - they're the most important.
- Tim: we have a toggle to "only show what's in the query". The docs show up on demand as you write the query, it's cool.
- Matt: another thing that's really cool about this design is people can replace docs explorer/etc and prove that it's better by writing code without having to worry about the entire ecosystem.
- Tim: at Meta, do you use the off-the-shelf GraphiQL?
- Matt: "yes, and" - we use it, but it's forked.
- Jordan - we do use it, we do have a fork of it that's more specific to our use case.
- Lee: would be great to upstream some of those - there are more folks than Meta with large schemas now!
- Lee: can you talk more about the editor mode switch.
- Tim: AST explorer, full screen docs view, etc can be toggled here.
- Lee: rather than having that hover over the query text editor, maybe you can use it to swap out the right pane - AST explorer, documentation view, etc. You normally want the

query text. That would mean if you have those (a single table) it might get away from eating all the vertical space.
- Tim: the domain motivation was that we'd have a full screen use case
- Benjie: perhaps we can make all the individual panes full-screenable, and make the right hand side replaceable as Lee suggests - putting SQL queries, query plans, etc in the RHS whilst still editing the query.
- Roman: [note taker: sorry, couldn't keep up]
- Tim: yes, that's why we're using Monaco
- Roman: create fragment on type. Clicking on a field that returns a type might create a selection subset so we don't have to select all.
- Tim: like a dropdown menu with a few options rather than adding directly?
- Tim: We definitely need to cover the fragment usecase somehow.
- Roman: what about a fragment library?
- Tim: definitely a plugin!
- Tim: the goal right now isn't to become Postman.
- Benjie: in terms of throwing spanners in the works, make sure that it handles specifying the same field multiple times with different aliases.
- Tim: yes, thanks.
- Lee: this is the all-star vision; we should move GraphiQL towards this slowly rather than trying to jump right to the finish, e.g. Monaco, then themes, then plugins; each thing could be a humongous piece of work, so even getting the broad strokes of this in place would be a dramatic improvement versus what we have today.
- Tim: agreed; Monaco first, then the visual part. It will be a rabbit hole to implement.
- Tim: there's the GraphiQL WG next week, there's lots of people interested in helping out, Rikki has worked a lot on the fundamental stuff on the React side.
- Matt: is there an online link on how to join the GraphiQL WG? Doesn't seem quite as documented as this one.
- Tim: it didn't exist for a while, I've created documents in the `working-group` folder of the GraphiQL repository.
- Lee: sidebar; we should rethink how we think about these - they're subcommittees of this WG, so we should have better link outs from here to them. Thanks Tim for getting some organization in place for them.
- Lee: very exciting work; we're nit-picking but it looks fantastic! Where should feedback go?
- Tim: go to Discussions tab of GraphiQL; GraphiQL 2 Design
    - https://github.com/graphql/graphiql/discussions/2216
- Tim: I believe there will be many edge cases, but we're excited.

Discuss proposal to allow unions to declare implementation of interfaces (15m, Yaacov)

- Spec PR: graphql/graphql-spec#939
- graphql-js PR: graphql/graphql-js#3527

- Allowing unions to declare that they implement interfaces - then they can only consist of object types that also declare they implement those interfaces.
- Yaacov: Main motivating case: e.g. `type HousePetEdge implements Edge`; `Node` is an interface, so for `HousePetEdge` to implement `Edge` then `HousePet` must implement `Node`; but if `HousePet` is a union we need to require that the union implements the interface.
- Michael: I'm intrigued by this, it gives more flexibility to unions. You could do it with interfaces, but this is a nicer way of combining them. The Relay spec doesn't require you have these interfaces.
- Yaacov: yes, but you could implement them with interfaces, and that enables additional functionality - especially for autogeneration. Possible future directions.
- I considered using `HousePet` as an interface instead.
- Matt: there's a surface area here; we could allow empty interfaces which would probably cover the same use cases, but allowing abstract types to implement abstract types is weird. Flip-side - unions can implement interfaces, but interfaces cannot be part of unions.
- Yaacov: I was considering handling that next, but it seems tricky!
- Matt: we may never get to the point where we want it even if we can have it.
- Roman: but there's already asymmetry between types and interfaces already.
- Matt: these are both abstract types.
- Lee: really this is a constraint on the union - the union itself doesn't implement the interface.
- Michael: indeed, it's like a generic constraint. Members in the set have to implement the interface, not the union itself.
- Lee: this seems like a nice to have rather than a need to have. My biggest question is: what's the precedent? Can we find other languages with similar shaped type systems that have a similar constraint? For a similarly shaped diamond dependency problem, how do those languages solve it?
- Roman: comparing to programming languages is not exactly the way to go, PLs have their own needs, we're special. I see the expressiveness benefit. Server-side it just feels like an extra validation step.
- Lee: Would still like to see precedent from other languages. Agreed that general purpose programming languages and GraphQL are different.
- Lee: Swift, Kotlin, TypeScript have similar constructs around union types/type classes/interfaces. The HousePetEdge / HousePet diamond dependency framing - how do other languages implement this dependency? It's okay to do it in a unique way if we can justify it.
- Yaacov: we could pull out the knowledge by introspecting the types and seeing they implement the interfaces, but it was suggested in #518 that if we don't declare that they implement those interfaces then someone could rely on it and it could break queries.
- Michael: could it be that issue #518 was created before interfaces-implement-interfaces was implemented?
- Yaacov: I think the explicit ability to limit the types allowed is still useful.
- Lee: it falls into that grey area between desiring simplicity vs desiring power.

- Matt: The question with doing this implicitly if that it makes schema change much more flexible. It would be unexpected if adding a type to a union broke the schema because the new type doesn't implement Node, so we should definitely state the constraint.
- Michael: an alternative would be if this constraint is only valid on this one field. Google doesn't seem to come up on a clear precedent for this.
- Lee: I've not heard of this kind of constraint on a union before, so I'm super curious how this type system problem has been solved. Explicitly stating the constraint is more useful and more stable.
- Roman: Question, which languages implement types like unions?
- [general] F#, TypeScript, Swift, Kotlin, Haskell, C#
- [someone] Objective-C has unions but not Swift.
- Michael: Usually you would have generic constraints on the field, but here it moves to the union type
- Ivan: question regarding execution behaviour; spec proposal doesn't change executor, as I understand in client queries if you want to use `id` you still need to do inline fragment/fragment, whereas when an interface implements an interface you don't need to do an inline fragment. In current version from DX it's better to do intermediate interface. I don't have a strong opinion on this proposal, but if it's merged then execution behaviour should also be changed.
- Yaacov: I follow, and I agree. I think I can add that. So it effectively becomes an empty interface whilst becoming a union. (Or at least no additional fields.)
- Ivan: in the current version you cannot request any fields at all without an inline fragment/fragment.
- Yaacov: maybe that does confuse the differences between unions/interfaces more.
- Lee: This is going to have cascading effects on introspection, this would extend introspection to now also be for unions.
- Lee: if you know it implements Node and you can't query 'id' then that would be strange.
- Michael: I was expecting this to be the intent already, was surprised it wasn't.
- Yaacov: sounds like there's consensus.
- Stephen: a full algebraic type system would support this - combining union/intersection type in the same type. TypeScript would allow this. Not sure how often it's used. Another thought regarding usefulness; we implicitly already have this concept - if you want to spread on Node, you can only do it if at least one of the types in the union implements Node; so there's an implicitness there that this could make more explicit.
- Lee: Yeah, that's a good point.
- Lee: next actions:
    - Better understand impact on execution. Does it change the way local selection sets work without needing fragment?
    - Look for precedent from other languages
    - Confusion of union implementing something (it being a constraint rather than union itself implementing) - perhaps it needs a different keyword - "contains"? I don't know.
- Yaacov: "constrainted by"?
- Roman: we already have interfaces implementing interfaces, why more keywords?

- Lee: just wanted to flag it.
- Yaacov: this was a confusion whilst implementing it as well!
- Lee: RFC1 for now. Let's keep people enthused.
- Chat:
  - Benjie: This is making it harder for me to try and deprecate unions :(
  - :D
  - From Michael Staib - to Everyone 08:22 PM
  - haha 😄
  - From Laurin to Everyone 08:22 PM
  - 😅
  - From Lee Byron (GraphQL Foundation) to Everyone 08:22 PM
  - unions should have been "one of" from the beginning
  - From Matt Mahoney to Everyone 08:22 PM
  - > This is making it harder for me to try and deprecate unions :(
  - Agree
  - From Laurin to Everyone 08:22 PM
  - > unions should have been "one of" from the beginning
  - 💯

Defer/Stream - *NOTE: below are the potentially final two discussion items prior to advancement (!!!)*

- Return arrays from stream payloads (15m, Rob)
  - Background: robrichard/defer-stream-wg#32
  - Goal: Confirm stream payload format.
- Asynchronous iterators of iterables versus of items (15 min, Yaacov)
  - Background: robrichard/defer-stream-wg#38
  - Goal 1: Confirm whether AsyncGenerator payload in reference implementation should be an iterable or a single item.
  - Goal 2: Confirm whether the spec must specify the AsyncGenerator payload type used.
- Rob: you can't go back in time and null something out, so we decided null should bubble to the boundary of the payload. Error versus null; can they be distinguished? For defer they're always objects, so null is explicit.
- For stream it was trickier, the data was the actual object. Suggestion was to wrap it in an array, when the element in it was null… [todo]
- I don't want to include batching in this proposal; we could add it later on. I'd like to move forward with data being an array with a single list element inside of it. I'd like consensus on this.
- Additionally we considered moving the number out of the path and into its own property. I'm less clear on the benefits. We talked about sending things out of order, but currently we require payloads are in order.
- I Suggest we move forward with data wrapped in an array and keep path as it is.

- Lee: I remember atIndex being pulled out differentiated from the full value versus the index within it - it gave you a fairly easy way to tell the difference between a stream into a list versus a deferred chunk.
- Rob: we already would have that with the array wrapping around the data - it dictates it's a stream payload.
- Benjie: also do you get an empty array in that initial payload `[]`?
- Rob: you do.
- Roman: the server can reuse the same stream for many clients; so the index number doesn't make sense, or it has to keep track of where each client started it.
- Michael: This is about the @stream directive and is single client.
- Roman: but the server might want to share the original stream (like stock quotes), and this would give a lot of headache for the server rather than just streaming the quotes it has to add the index. Is that useful for the client?
- Another thing; you want to avoid the batching? But I think batching is inevitably coming for efficiency, and if it's inevitable then we should address the potential implementation issues now rather than discovering them later. So re indexing, effectively it prevents reusing these streams.
- Michael: I don't see it that way - you have local state in the execution engine anyway.
- Yaacov: I have an implementation up at graphql-executor and it has different behaviour when batching is requested (parallel streaming); it only sends atIndex when parallel streaming is enabled. Building on that the client is going to have to specify if they want batching/out of order/parallel streaming. The default is sequential, if we enable batching later we can add the additional field. Since it's guarantee to be sequential - do we need to include the index? Especially now we've solved the null problem.
- Lee: Is there a case where some kind of data integrity issue could cause something to be dropped and this being used as parity check?
- Yaacov: stock quotes would be a subscription example; the "top 1000 songs of the nineties" is more of a @stream request. I would support putting it back in the path field.
- Lee: There is something nice about, if we want to include it, a client […] could use the index to insert it at the right path
- Lee: the parity check seems compelling, to help debugging, and allowing out of order payloads later.
- Rob: there will be numbers in the path for non-stream payloads, streams inside of streams, etc.
- Michael: if we agree that it's always an append then we can remove the number at the end.
- Lee: I'm always in favour of adding more data rather than less so that it doesn't bite us as the feature evolves. If it was a nested stream then the parents one would be in the path.
- Roman: we send array in slices; this array of items is sometimes a set; neither client nor server care about order. When the object arrives it has the index inside. It's not left to the protocol - they need this kind of thing everywhere, so the index is already there explicit. It should not be a big concern for us here. If order's important then it should be inside the object.

- Yaacov: It sounds like you agreeing that we could use the number if the order is important?
- Roman: we should not use it; we're getting into the application area. We promise to deliver and that's it. Server messing up order by pulling from different sources, server has to fix.
- Yaacov: I believe Lee's point was about sanity checking, not about order being wrong.
- Lee: separate but related concerns
    - Given the constraints we've already set up what if there's a reliability issue - we can't differentiate this error condition versus a safe condition; even if most people won't use it
    - Todays constraint says that they must be returned in order, but that won't be the case forever into the future. While including the index now seems superfluous, it seems like a cheap investment to future proof.
- Ivan: I might not have the full context, but my concern about the data in general (not just atIndex) is that we have very different payload for initial response versus subsequent responses. If we're addressing advanced use cases we shouldn't try to split these use cases like batching into a payload form that was designed for initial simple response.
- Chat:

    -
      ```json
      {
        "incremental": [
              {
              "label": "filmsStream",
              "path": ["person", "films", 2],
              "data": [{"title": "Return of the Jedi"}],
              "hasNext": false
              },
              {
              "label": "filmsStream",
              "path": ["person", "films", 3],
              "data": [{"title": "Some title for item #3"}],
              "hasNext": false
              }
        ],
        "hasNext": false,
      }
      ```
- Another thing people have asked, is identifying how a stream has finished vs the whole async operation
- hasNext is currently global, so I can't differentiate which stream finished. Maybe I'm missing something? Can we explore a different response shape?
- Lee: we talked about this before regarding hasNext… actually nevermind the initial payload has this too.

- Rob: Do you suggest that if we send 1 payload it would be wrapped in that incremental property?
- Ivan: I'm not good at naming things, I normally as Benjie for better names. Server can ignore stream/defer - I don't like that part too much, it's weird for the server to ignore the directive. Reason was optimisation: what if server can return stuff immediately. If we do top level field, server is required to split query but can return it in the same response. We could batch on the number of payloads under the "incremental" field.
- Rob: We spoke how the FB server has a way to batch by returning multiple payloads in an array, so we suggested to return all incremental payloads in an array, even when not batched.
- Matt: at Meta, we don't wrap in an array, we just return two JSON objects back-to-back in the same HTTP response. The parse needs to break at JSON object chain.
- Lee: there's a subtle difference in client side behaviour - client could process first object and then trigger render and then process next and trigger render - could cause CPU thrashing. Wrapping it in an array tells the parser to handle everything before rendering.
- Matt: To be clear, I'm not arguing that FB's approach was necessarily correct. Even in GraphiQL this would surface as an issue, as it doesn't know how to handle the back to back JSON payloads.
- Michael: from an efficiency POV you're always sending more data down.
- Yaacov: we're probably hitting on the next point of discussion now. Are we done with the discussion about path? Consensus seems to be around leaving off atIndex and putting it in path.
- Lee: sounds good.
- Yaacov: when the backend is supplying a very long list, or the server is using an async iterator for the data source. The suggestion is that rather than using an async iterator of values, we use an async iterator of iterables - this means we don't have to go back to the event loop on every item, we can do things faster and we can send payloads at the same time to the client. This is a breaking change in the GraphQL.js implementation. Introducing this change later is going to cause problems for people who've written resolvers that return async iterables of values directly.
- This dovetails nicely with what Ivan was saying before. Entire result can be wrapped in an array because they're all available at the same time.
- [Please see YouTube video)
- Lee: I think this makes a lot of sense; I'm going to have to read more, but preserving compatibility for having one streamed list whilst maintaining the ability… [couldn't keep up - sorry!]
- Ivan: I've done experiments, your proposal is to support [something] batching and data batching. In data batching we don't have to specify the fields. In other compression formats; since we have defined order of fields, if we have path, etc; during payload the difference is non-existent
- Lee: compression solves all problems
- Ivan: so it's not a problem if we're duplicating. I expect people will only use a couple defer. Complexity overhead is not justified.
- Yaacov: the key thing is do you agree with using arrays to batch entire payloads.

- Lee: there’s a lot of feedback to give here; we’ll address it in the GitHub discussions since we’re over time. Thanks everyone!