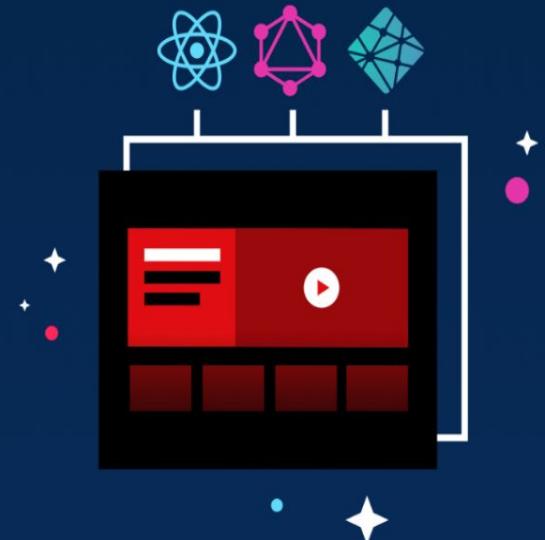


LIVE HANDS-ON WORKSHOP

# Introduction to GraphQL

November 23, Wednesday

8am PT | 4pm GMT | 5pm CET | 8.30pm IST



DataStax

# 01



## Housekeeping Live and Hands-on

# 02



## Database Setup Data model & Astra DB

# 03



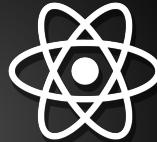
## GraphQL Design & Toolings

# 04



## Backend Expose GQL endpoint

# 05



## Frontend Consume GQL Endpoint

# 06



## What's next? Quiz, Homework, ...



Agenda





# Housekeeping

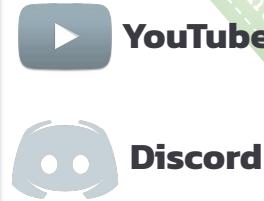
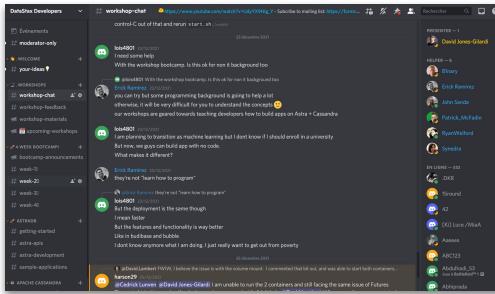
Live and Hands-on



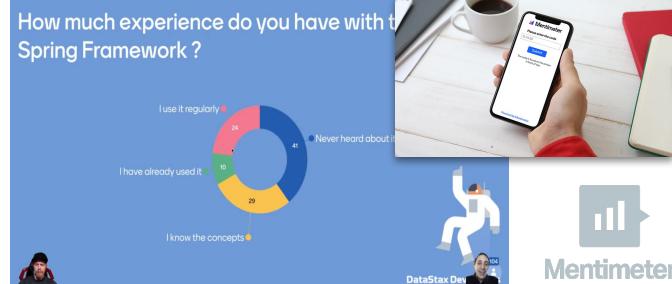
**Livestream:** youtube.com/DataStaxDevs

**Questions:** <https://dtsx.io/discord>

#### Agenda



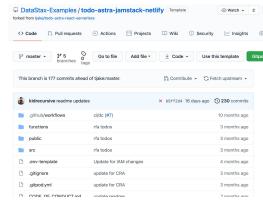
## Games and quizzes: [menti.com](https://menti.com)



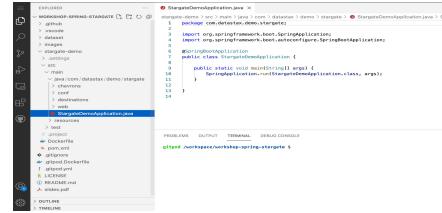
Mentimeter

Nothing to install !

### Source code, exercises, slides



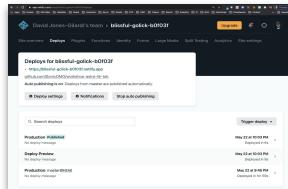
### IDE



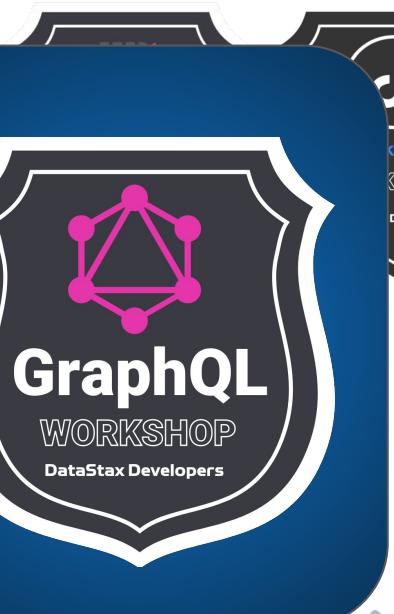
### Database + GraphQL + PlayGround



### Deploy + Run + Host



Hands-On Housekeeping



Get your badge for today

# 01



## Housekeeping Live and Hands-on

# 02



## Database Setup Data model & Astra DB

# 03



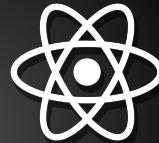
## GraphQL Design & Toolings

# 04



## Backend Expose GQL endpoint

# 05



## Frontend Consume GQL Endpoint

# 06

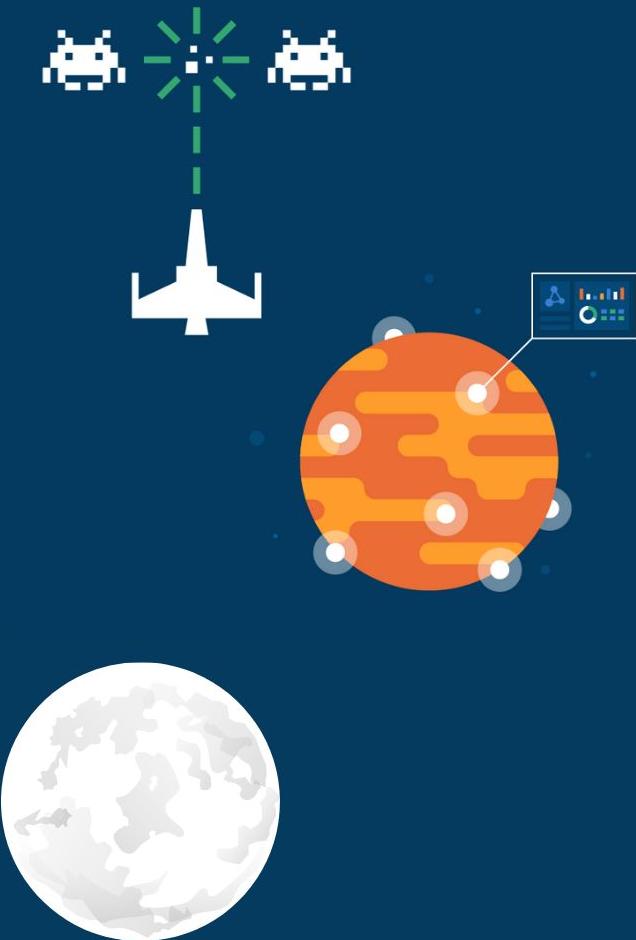


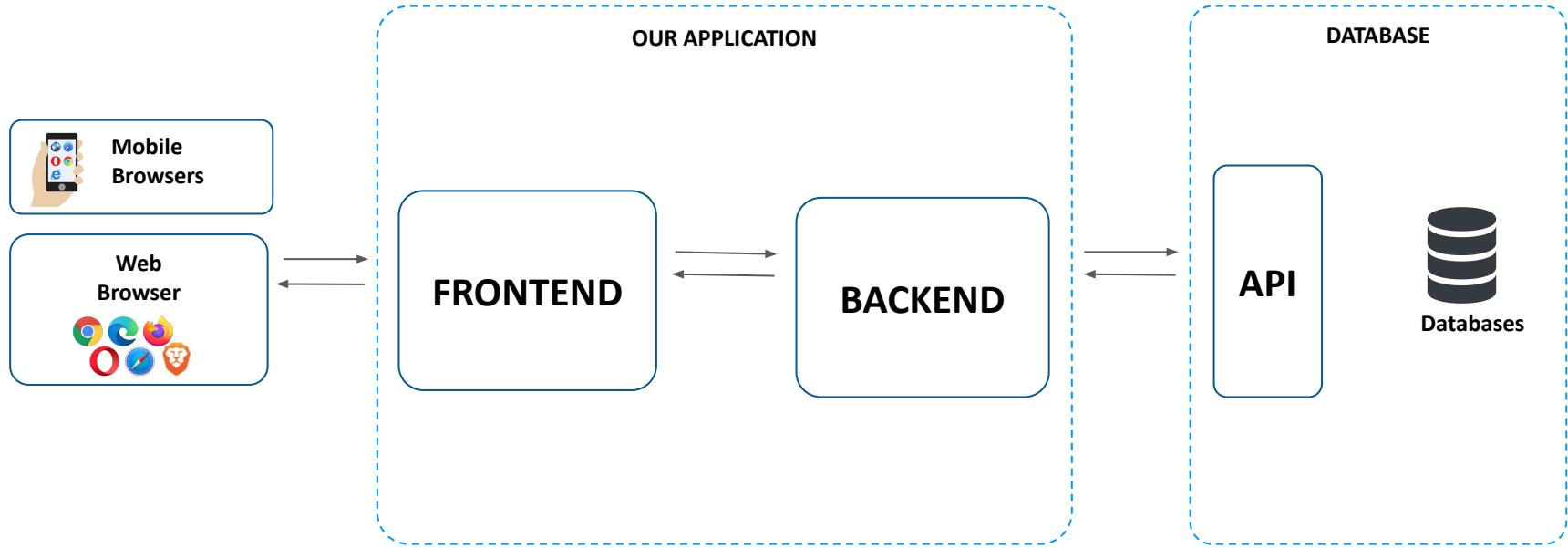
## What's next? Quiz, Homework, ...



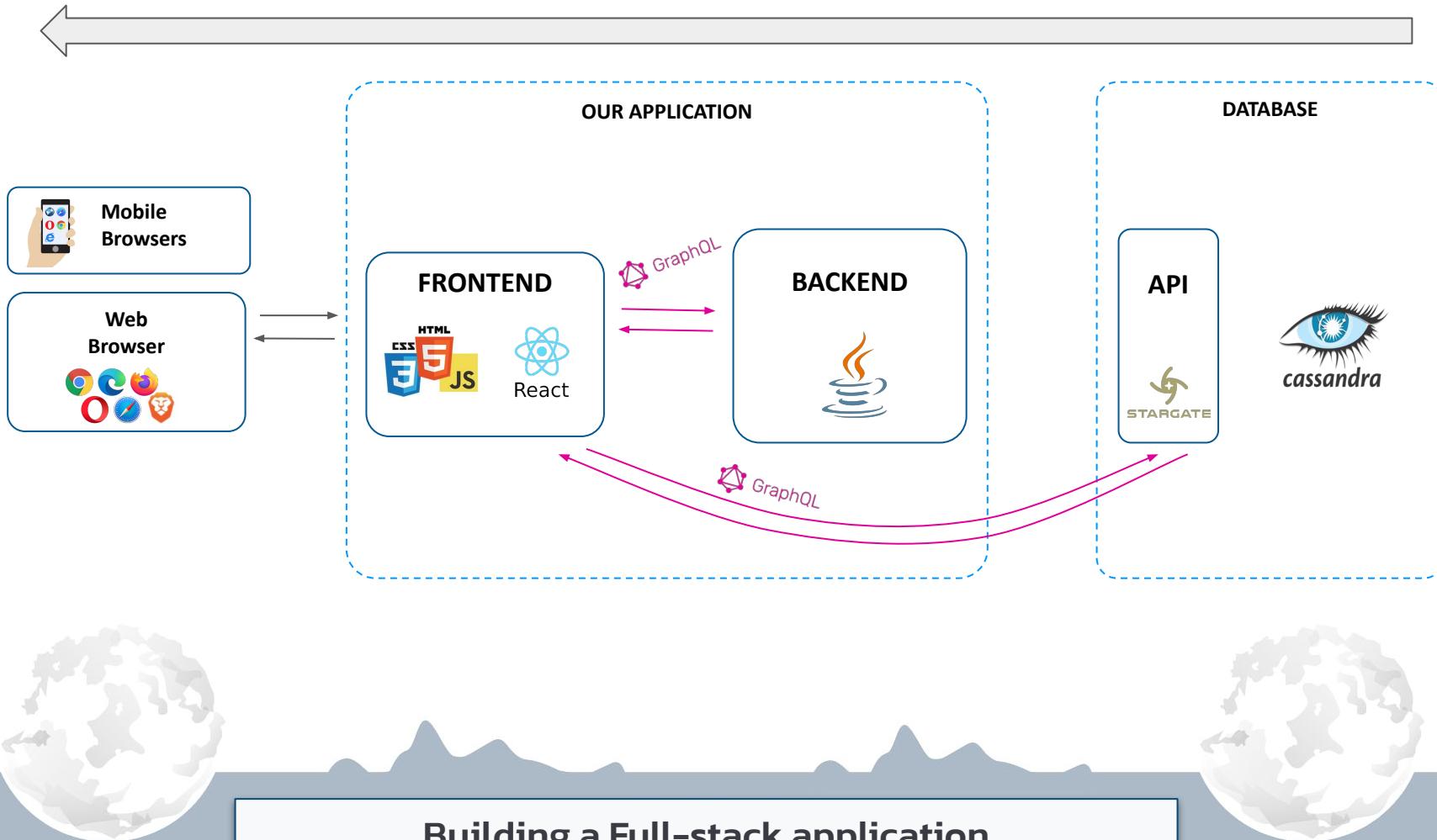
## Agenda







Building a Full-stack application



## Client (React) app



[Intro to GraphQL by DataStax Developers](#)

**Shows(Local)**

Stranger Things: 2016

**Genres(Local)**

Action

**ShowsByName(Astra DB)**

Stranger Things: 2016

**ReferenceList(Astra DB)**

Action

"backend" (local) pathway

## Locally-running DGS GraphQL API

`schema.graphqls`

```
type Show {  
    title: String  
    releaseYear: Int  
}  
  
type Genre {  
    value: String!  
}
```

*hardcoded list in data fetcher*

*hardcoded list in data fetcher*

## Astra DB

(graphQL)  
API



"Astra" pathway

Table "show\_by\_name"

title <sup>k</sup>	releaseYear
Stranger Things	2016
Ozark	2017



cassandra

label <sup>k</sup>	value <sup>c</sup>
genre	Action
genre	Anime

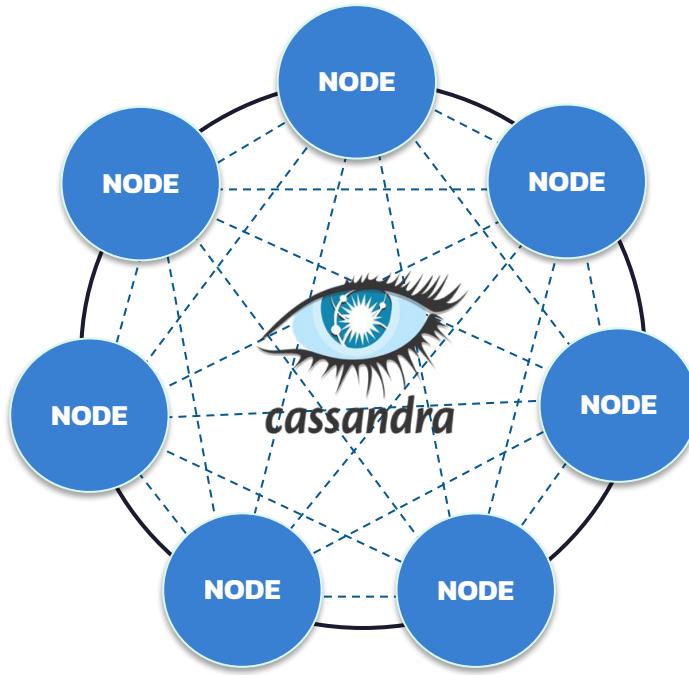
Today's target app

# Database setup



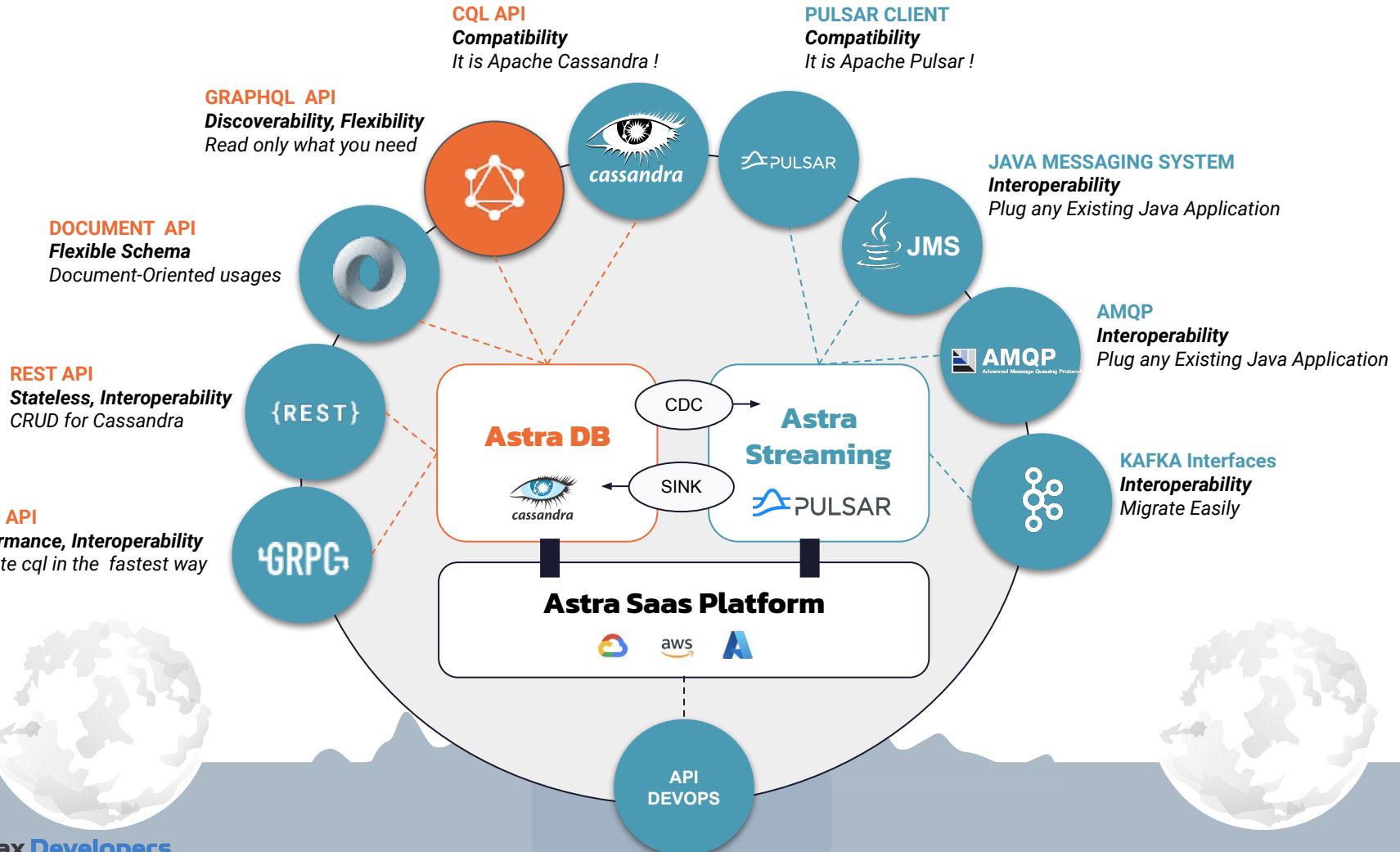
Data model & Astra DB

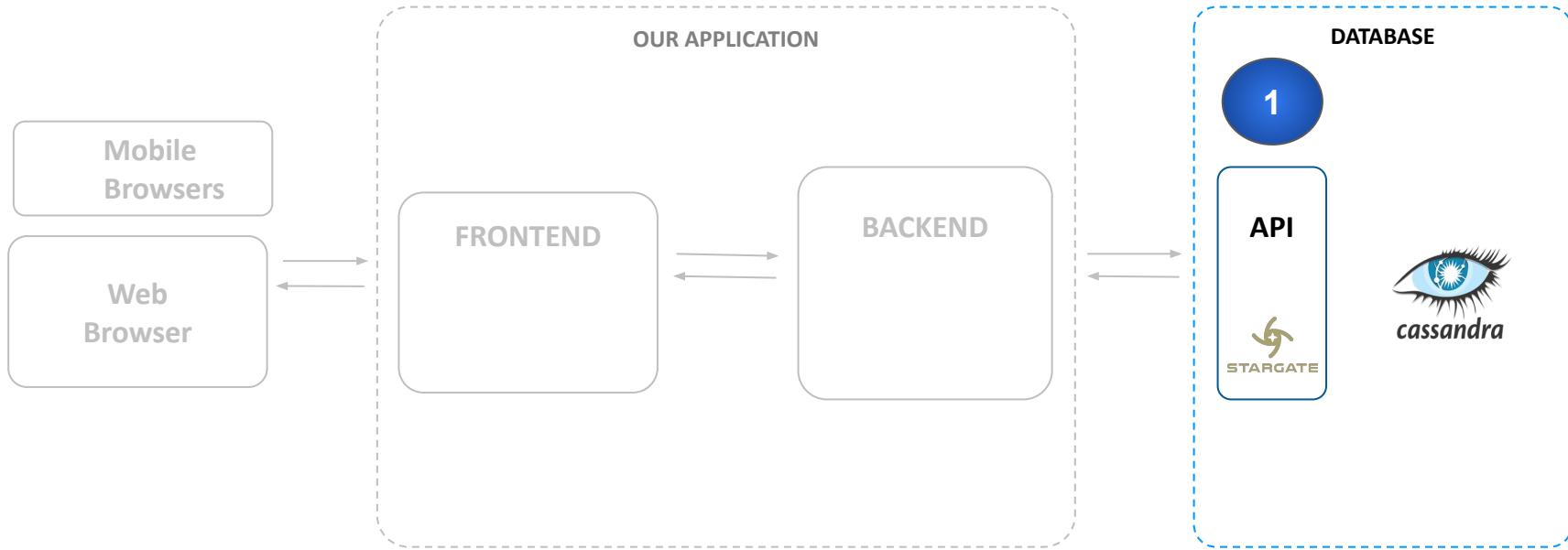




- Big Data Ready
- Read / Write Performance
- Linear Scalability
- Highest Availability
- Self-Healing and Automation
- Geographical Distribution
- Platform Agnostic
- Vendor Independent

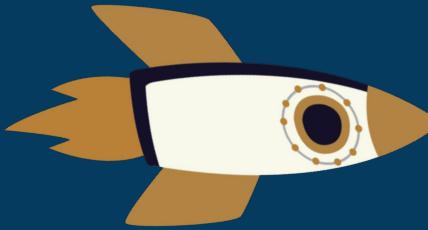
Introduction to Apache Cassandra™





Setup Database





# Hands-on (!github)

## #1 DB SETUP

- ✓ Register & Create your Database
- ✓ Create a security token if needed

# 01



## Housekeeping Live and Hands-on

# 02



## Database Setup Data model & Astra DB

# 03



## Backend Expose GQL endpoint

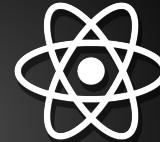
# 04



GraphQL

## GraphQL Design & Toolings

# 05



## Frontend Consume GQL Endpoint

# 06



## What's next? Quiz, Homework, ...



Agenda





 **GraphQL**  
GraphQL  
Learn & Design

- Created by Facebook  
*Used internally for mobile apps*
- 2012
- Facebook give talk at ReactJs Conf
- 2015
- Facebook open sourced GraphQL
- 2015
- Github announced move to GQL
- 2016



GraphQL

## Definition

**GraphQL is an application programming interface (API) query language and server-side runtime that prioritises giving customers precisely the data they request.**



Meet GraphQL



It is a web protocol (HTTP) that specifies the way we build and query remote APIs using a Tree/JSON Syntax.

Based on a **strongly typed** language named GraphQL SDL  
(*Schema Definition Language*)

```
query getShows {  
    shows {  
        title  
        releaseYear  
    }  
}
```

query sample

```
type Query {  
    shows: [Show]  
}  
  
type Show {  
    title: String!  
    blurb: String  
    actors: [Actor]  
    releaseYear: Int  
}
```

schema snippet

<https://graphql.org/learn/schema/#type-language>

What is graphQL ?

A protocol that allows the client to specify exactly what data it needs from a model

It allows one to aggregate data from **multiple relations** in a single query

query sample

```
query getShows {  
  shows {  
    title  
  }  
}
```

query sample

```
query getShows {  
  shows {  
    title  
    blurb  
    actors: {  
      name  
    }  
    releaseYear  
  }  
}
```

<https://graphql.org/learn/schema/#type-language>



What is GraphQL ?

- **Int:** An integer type (example: 10)
  - **Float:** Floating point number, example 3.43
  - **String:** A sequence of characters ("Hello World")
  - **Boolean:** A boolean example true
  - **ID:** An object identifier
- 
- **User Defined types**
    - Create your structure with constraints

```
type Show {  
    title: String!  
    blurb: String  
    actors: [Actor]  
    releaseYear: Int  
}  
  
type Actor {  
    name: String!  
    age: Int  
}
```

schema snippet



## Type System - Built-in scalar types and UDTs



```
{  
  hero {  
    name  
    friends {  
      name  
      homeWorld {  
        name  
        climate  
      }  
      species {  
        name  
        lifespan  
        origin {  
          name  
        }  
      }  
    }  
  }  
}
```

```
type Query {  
  hero: Character  
}  
  
type Character {  
  name: String  
  friends: [Character]  
  homeWorld: Planet  
  species: Species  
}  
  
type Planet {  
  name: String  
  climate: String  
}  
  
type Species {  
  name: String  
  lifespan: Int  
  origin: Planet  
}
```

## Describe what's possible with a type system

GraphQL APIs are organized in terms of types and fields, not endpoints. Access the full capabilities of your data from a single endpoint. GraphQL uses types to ensure Apps only ask for what's possible and provide clear and helpful errors. Apps can use types to avoid writing manual parsing code.

<https://graphql.org>

Schema Definition

- Used for **READ ONLY** requests.
- Used to fetch data from the server using the **GraphQL SDL syntax**
- Describe what data the requester wishes to fetch from whoever is fulfilling the **GraphQL Query**
- **All runnable queries must appear in schema!**

query sample

```
query getSortedShows {  
  shows(titleFilter: "Ozark") {  
    title  
    blurb  
    releaseYear  
  }  
}
```

optional  
params

schema snippet

```
type Query {  
  shows(titleFilter: String): [Show]  
}  
  
type Show {  
  ...  
}
```

```
{  
  hero {  
    name  
    height  
    mass  
  }  
}  
  
{  
  "hero": {  
    "name": "Luke Skywalker",  
    "height": 1.72,  
    "mass": 77  
  }  
}
```

Ask for what you need,  
get exactly that

Send a GraphQL query to your API and get exactly what you need, nothing more and nothing less. GraphQL queries always return predictable results. Apps using GraphQL are fast and stable because they control the data they get, not the server.

<https://graphql.org>

Declarative Query

- Used to change resources data or execute actions on the server
- Client specifies the arguments and the action to be executed and then receives a response or a resource updated
- All runnable mutations must appear in schema!

query sample

```
mutation insertMyFavoriteShow {  
  createShow({  
    title: "Anna Karenina"  
    blurb: "Anna; train; splat!"  
    actors: [  
      {  
        name: "Greta Garbo"  
        age: 30  
      },  
      {  
        name: "Fredric March"  
        age: 38  
      }  
    ]  
    releaseYear: 111  
  })  
  {  
    title  
  }  
}
```

What is a GraphQL Mutation ?



- ❖ Discoverability, introspection
- ❖ Declarative Data Fetching
- ❖ Schema stitching
- ❖ Customize responses
- ❖ Flexible versioning
- ❖ Match standards (Json | Http)



- ❖ Single endpoint (*versioning, monitoring, security*)
- ❖ Complex implementation (*tooling, still young*)
- ❖ Nice for customers, nasty for DB (*N+1 select*)
- ❖ Rate limiting is required
- ❖ Caching is hard (Apollo is a good solution here  
<https://www.apollographql.com>) <- this is what Netflix uses



- ❖ BFF : Backend for frontend
- ❖ Service aggregation | composition, "federation" (*joins*)
- ❖ When bandwidth matters (*mobile phones*)



GraphQL S.W.O.T. analysis



 **GraphQL**  
GraphQL  
**Tooling**

# Graph*QL*

## ENDPOINT URL

## EXECUTE

## QUERY

## RESULTS

The screenshot shows the GraphQL Swapi IDE interface. On the left, a code editor displays a GraphQL query:

```
# Welcome to GraphQL
# GraphQL is an in-browser IDE for writing GraphQL queries.
# Type queries into this side of the pane, and see intelligent typeheads aware of live syntax and validation errors here.
# To bring up the auto-complete at any time, press Tab.
# Press the run button above, or Cmd+Enter, and it will appear in the pane to the right.
{
  allFilms {
    films {
      title
      episodeID
    }
  }
}
```

On the right, the results of the query are shown as JSON:

```
{
  "data": {
    "allFilms": {
      "films": [
        {
          "title": "A New Hope",
          "episodeID": 4
        },
        {
          "title": "The Empire Strikes Back",
          "episodeID": 5
        },
        {
          "title": "Return of the Jedi",
          "episodeID": 6
        },
        {
          "title": "The Phantom Menace",
          "episodeID": 1
        },
        {
          "title": "Attack of the Clones",
          "episodeID": 2
        },
        {
          "title": "Revenge of the Sith",
          "episodeID": 3
        }
      ]
    }
  }
}
```

A blue circle highlights the right-hand panel, which contains the following information:

- FilmsConnection**
- Film**
- Sche**
- A single film.**
- IMPLEMENTS**
- Node**
- FIELDS**
- title: String**
- episodeID: Int**
- openingCrawl: String**
- director: String**
- producers: [String]**
- releaseDate: String**
- speciesConnection(after: String, first: Int, before: String, last: Int): FilmSpeciesConnection**
- starshipConnection(after: String, first: Int, before: String, last: Int): FilmStarshipsConnection**
- vehicleConnection(after: String, first: Int, before: String, last: Int): FilmVehiclesConnection**
- characterConnection(after: String, first: Int, before: String, last: Int): FilmCharactersConnection**
- planetConnection(after: String, first: Int, before: String, last: Int): FilmPlanetsConnection**
- created: String**
- edited: String**

# Schema

# GraphQL Playground

## DDL: DATA DEFINITION

Create Table

Create Schema

## DML: DATA MANAGEMENT

INSERT/RETRIEVE DATA

ENDPOINT URL

EXECUTE

HEADER

The screenshot shows the GraphQL Playground interface. On the left, there's a code editor with a query like `1 { keyspaces { name } }`. Below it are sections for "QUERY VARIABLES" and "HTTP HEADERS". A blue circle highlights the "HTTP HEADERS" section, which contains a header named `iuZspoYB:98936cfa48c7eacc14cf1b0fb4169b394a15b52cb7e2463ece31995bdf10feb4`.

In the center, a large blue arrow points from the "EXECUTE" button towards the results viewer, which displays a JSON response:

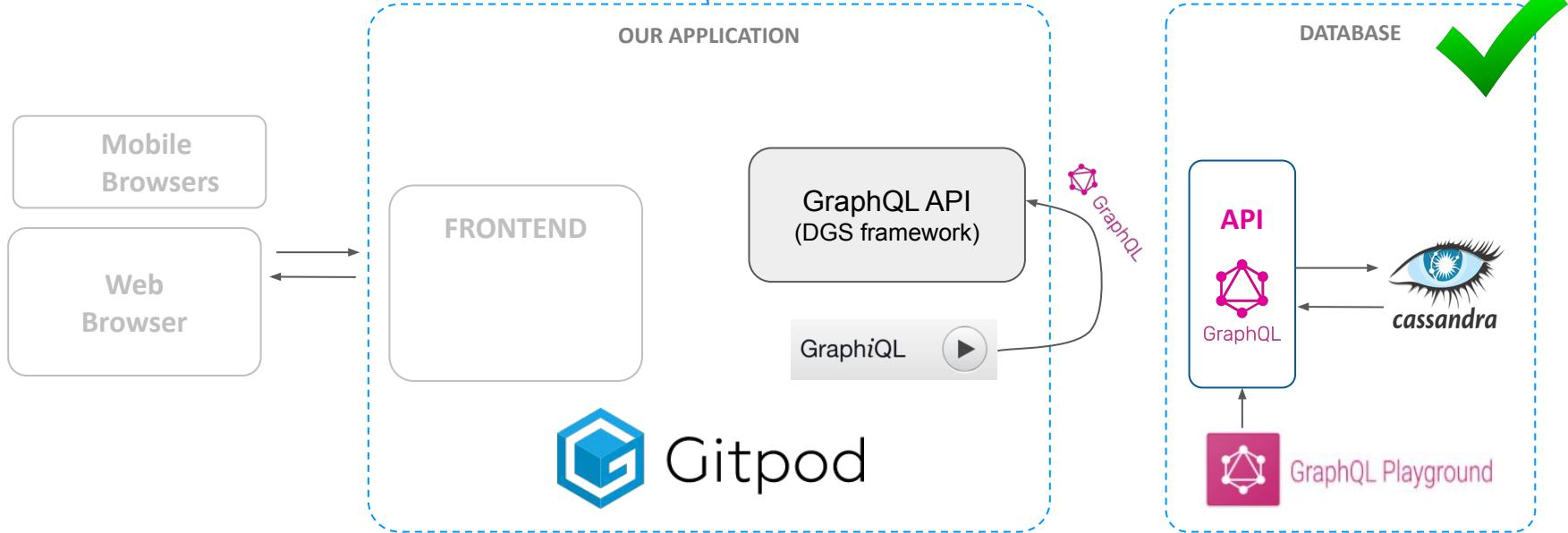
```
{ "data": { "keyspaces": [ { "name": "system_traces" } ] } }
```

On the right, a sidebar titled "Schema" is open, showing the schema structure with tabs for "DOCS" and "SCHEMA". It lists various mutations and their descriptions:

- keyspace(...): Keyspace
- keyspaces: [Keyspace]
- createTable(...): Boolean
- alterTableAdd(...): Boolean
- alterTableDrop(...): Boolean
- dropTable(...): Boolean
- createType(...): Boolean
- dropType(...): Boolean
- createIndex(...): Boolean
- dropIndex(...): Boolean
- createKeyspace(...): Boolean
- dropKeyspace(...): Boolean

Schema

developers



What's coming in next Hands-on ?

The screenshot illustrates the Gitpod UI interface, which is designed for developing and running applications across multiple environments. The interface is divided into several sections:

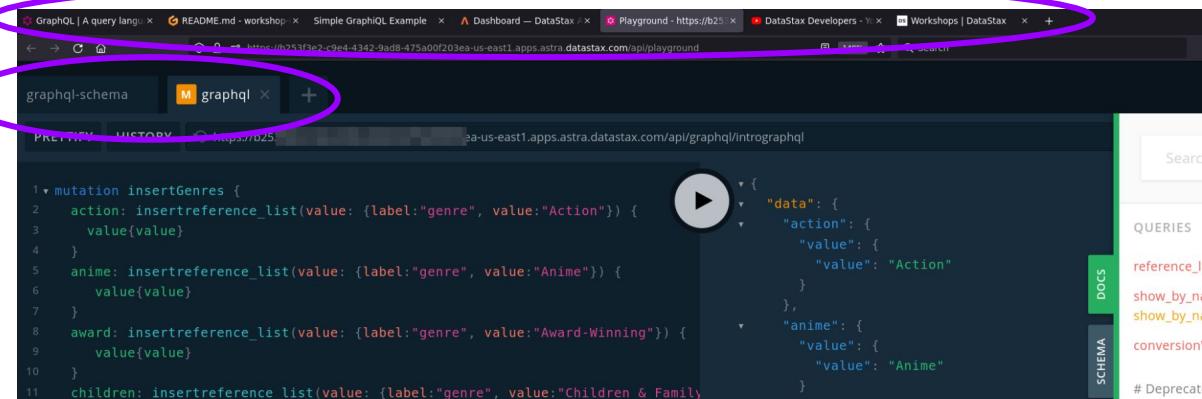
- File explorer** (1): Located on the left side, it shows a tree view of the workspace structure, including subfolders like `code`, `graphql-client-examples`, and `graphql-backend-examples`.
- Editor/viewer** (2): The central area displays a README.md file titled "Introduction to GraphQL + React + Java + Astra DB". It includes a "Get Started" button, a "Tutorial" link, and a "Start Building" button.
- Console switcher** (5): On the right, there is a panel for switching between different terminals or consoles. It currently shows two tabs: `gitpod-host-config: bash` and `graphql-client: bash`.
- "Client" console** (3): This is the leftmost terminal window, showing a command-line session for a GraphQL client example. The user has run commands related to `graphql-client-examples`.
- "Backend" console** (4): This is the rightmost terminal window, showing a command-line session for a Java application running on Tomcat. It displays logs indicating the application has started successfully.

**DataStax Developers**

**Gitpod UI cheat sheet**

## *In your GraphQL Playground ...*

**These are your browser's tabs**

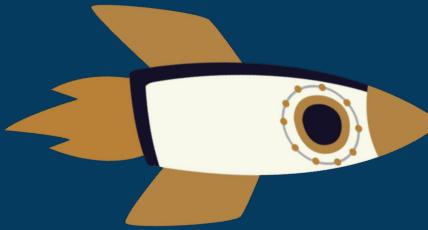


```
1 mutation insertGenres {  
2   action: insertereference_list(value: {label:"genre", value:"Action"}) {  
3     value{value}  
4   }  
5   anime: insertereference_list(value: {label:"genre", value:"Anime"}) {  
6     value{value}  
7   }  
8   award: insertereference_list(value: {label:"genre", value:"Award-Winning"}) {  
9     value{value}  
10  }  
11  children: insertereference_list(value: {label:"genre", value:"Children & Family"}  
12  }  
13  }  
14 }  
15 }  
16 }  
17 }  
18 }  
19 }  
20 }  
21 }  
22 }  
23 }  
24 }  
25 }  
26 }  
27 }  
28 }  
29 }  
30 }  
31 }  
32 }  
33 }  
34 }  
35 }  
36 }  
37 }  
38 }  
39 }  
40 }  
41 }  
42 }  
43 }  
44 }  
45 }  
46 }  
47 }  
48 }  
49 }  
50 }  
51 }  
52 }  
53 }  
54 }  
55 }  
56 }  
57 }  
58 }  
59 }  
60 }  
61 }  
62 }  
63 }  
64 }  
65 }  
66 }  
67 }  
68 }  
69 }  
70 }  
71 }  
72 }  
73 }  
74 }  
75 }  
76 }  
77 }  
78 }  
79 }  
80 }  
81 }  
82 }  
83 }  
84 }  
85 }  
86 }  
87 }  
88 }  
89 }  
90 }  
91 }  
92 }  
93 }  
94 }  
95 }  
96 }  
97 }  
98 }  
99 }  
100 }  
101 }  
102 }  
103 }  
104 }  
105 }  
106 }  
107 }  
108 }  
109 }  
110 }  
111 }  
112 }  
113 }  
114 }  
115 }  
116 }  
117 }  
118 }  
119 }  
120 }  
121 }  
122 }  
123 }  
124 }  
125 }  
126 }  
127 }  
128 }  
129 }  
130 }  
131 }  
132 }  
133 }  
134 }  
135 }  
136 }  
137 }  
138 }  
139 }  
140 }  
141 }  
142 }  
143 }  
144 }  
145 }  
146 }  
147 }  
148 }  
149 }  
150 }  
151 }  
152 }  
153 }  
154 }  
155 }  
156 }  
157 }  
158 }  
159 }  
160 }  
161 }  
162 }  
163 }  
164 }  
165 }  
166 }  
167 }  
168 }  
169 }  
170 }  
171 }  
172 }  
173 }  
174 }  
175 }  
176 }  
177 }  
178 }  
179 }  
180 }  
181 }  
182 }  
183 }  
184 }  
185 }  
186 }  
187 }  
188 }  
189 }  
190 }  
191 }  
192 }  
193 }  
194 }  
195 }  
196 }  
197 }  
198 }  
199 }  
200 }  
201 }  
202 }  
203 }  
204 }  
205 }  
206 }  
207 }  
208 }  
209 }  
210 }  
211 }  
212 }  
213 }  
214 }  
215 }  
216 }  
217 }  
218 }  
219 }  
220 }  
221 }  
222 }  
223 }  
224 }  
225 }  
226 }  
227 }  
228 }  
229 }  
230 }  
231 }  
232 }  
233 }  
234 }  
235 }  
236 }  
237 }  
238 }  
239 }  
240 }  
241 }  
242 }  
243 }  
244 }  
245 }  
246 }  
247 }  
248 }  
249 }  
250 }  
251 }  
252 }  
253 }  
254 }  
255 }  
256 }  
257 }  
258 }  
259 }  
260 }  
261 }  
262 }  
263 }  
264 }  
265 }  
266 }  
267 }  
268 }  
269 }  
270 }  
271 }  
272 }  
273 }  
274 }  
275 }  
276 }  
277 }  
278 }  
279 }  
280 }  
281 }  
282 }  
283 }  
284 }  
285 }  
286 }  
287 }  
288 }  
289 }  
290 }  
291 }  
292 }  
293 }  
294 }  
295 }  
296 }  
297 }  
298 }  
299 }  
300 }  
301 }  
302 }  
303 }  
304 }  
305 }  
306 }  
307 }  
308 }  
309 }  
310 }  
311 }  
312 }  
313 }  
314 }  
315 }  
316 }  
317 }  
318 }  
319 }  
320 }  
321 }  
322 }  
323 }  
324 }  
325 }  
326 }  
327 }  
328 }  
329 }  
330 }  
331 }  
332 }  
333 }  
334 }  
335 }  
336 }  
337 }  
338 }  
339 }  
340 }  
341 }  
342 }  
343 }  
344 }  
345 }  
346 }  
347 }  
348 }  
349 }  
350 }  
351 }  
352 }  
353 }  
354 }  
355 }  
356 }  
357 }  
358 }  
359 }  
360 }  
361 }  
362 }  
363 }  
364 }  
365 }  
366 }  
367 }  
368 }  
369 }  
370 }  
371 }  
372 }  
373 }  
374 }  
375 }  
376 }  
377 }  
378 }  
379 }  
380 }  
381 }  
382 }  
383 }  
384 }  
385 }  
386 }  
387 }  
388 }  
389 }  
390 }  
391 }  
392 }  
393 }  
394 }  
395 }  
396 }  
397 }  
398 }  
399 }  
400 }  
401 }  
402 }  
403 }  
404 }  
405 }  
406 }  
407 }  
408 }  
409 }  
410 }  
411 }  
412 }  
413 }  
414 }  
415 }  
416 }  
417 }  
418 }  
419 }  
420 }  
421 }  
422 }  
423 }  
424 }  
425 }  
426 }  
427 }  
428 }  
429 }  
430 }  
431 }  
432 }  
433 }  
434 }  
435 }  
436 }  
437 }  
438 }  
439 }  
440 }  
441 }  
442 }  
443 }  
444 }  
445 }  
446 }  
447 }  
448 }  
449 }  
450 }  
451 }  
452 }  
453 }  
454 }  
455 }  
456 }  
457 }  
458 }  
459 }  
460 }  
461 }  
462 }  
463 }  
464 }  
465 }  
466 }  
467 }  
468 }  
469 }  
470 }  
471 }  
472 }  
473 }  
474 }  
475 }  
476 }  
477 }  
478 }  
479 }  
480 }  
481 }  
482 }  
483 }  
484 }  
485 }  
486 }  
487 }  
488 }  
489 }  
490 }  
491 }  
492 }  
493 }  
494 }  
495 }  
496 }  
497 }  
498 }  
499 }  
500 }  
501 }  
502 }  
503 }  
504 }  
505 }  
506 }  
507 }  
508 }  
509 }  
510 }  
511 }  
512 }  
513 }  
514 }  
515 }  
516 }  
517 }  
518 }  
519 }  
520 }  
521 }  
522 }  
523 }  
524 }  
525 }  
526 }  
527 }  
528 }  
529 }  
530 }  
531 }  
532 }  
533 }  
534 }  
535 }  
536 }  
537 }  
538 }  
539 }  
540 }  
541 }  
542 }  
543 }  
544 }  
545 }  
546 }  
547 }  
548 }  
549 }  
550 }  
551 }  
552 }  
553 }  
554 }  
555 }  
556 }  
557 }  
558 }  
559 }  
560 }  
561 }  
562 }  
563 }  
564 }  
565 }  
566 }  
567 }  
568 }  
569 }  
570 }  
571 }  
572 }  
573 }  
574 }  
575 }  
576 }  
577 }  
578 }  
579 }  
580 }  
581 }  
582 }  
583 }  
584 }  
585 }  
586 }  
587 }  
588 }  
589 }  
590 }  
591 }  
592 }  
593 }  
594 }  
595 }  
596 }  
597 }  
598 }  
599 }  
600 }  
601 }  
602 }  
603 }  
604 }  
605 }  
606 }  
607 }  
608 }  
609 }  
610 }  
611 }  
612 }  
613 }  
614 }  
615 }  
616 }  
617 }  
618 }  
619 }  
620 }  
621 }  
622 }  
623 }  
624 }  
625 }  
626 }  
627 }  
628 }  
629 }  
630 }  
631 }  
632 }  
633 }  
634 }  
635 }  
636 }  
637 }  
638 }  
639 }  
640 }  
641 }  
642 }  
643 }  
644 }  
645 }  
646 }  
647 }  
648 }  
649 }  
650 }  
651 }  
652 }  
653 }  
654 }  
655 }  
656 }  
657 }  
658 }  
659 }  
660 }  
661 }  
662 }  
663 }  
664 }  
665 }  
666 }  
667 }  
668 }  
669 }  
670 }  
671 }  
672 }  
673 }  
674 }  
675 }  
676 }  
677 }  
678 }  
679 }  
680 }  
681 }  
682 }  
683 }  
684 }  
685 }  
686 }  
687 }  
688 }  
689 }  
690 }  
691 }  
692 }  
693 }  
694 }  
695 }  
696 }  
697 }  
698 }  
699 }  
700 }  
701 }  
702 }  
703 }  
704 }  
705 }  
706 }  
707 }  
708 }  
709 }  
710 }  
711 }  
712 }  
713 }  
714 }  
715 }  
716 }  
717 }  
718 }  
719 }  
720 }  
721 }  
722 }  
723 }  
724 }  
725 }  
726 }  
727 }  
728 }  
729 }  
730 }  
731 }  
732 }  
733 }  
734 }  
735 }  
736 }  
737 }  
738 }  
739 }  
740 }  
741 }  
742 }  
743 }  
744 }  
745 }  
746 }  
747 }  
748 }  
749 }  
750 }  
751 }  
752 }  
753 }  
754 }  
755 }  
756 }  
757 }  
758 }  
759 }  
760 }  
761 }  
762 }  
763 }  
764 }  
765 }  
766 }  
767 }  
768 }  
769 }  
770 }  
771 }  
772 }  
773 }  
774 }  
775 }  
776 }  
777 }  
778 }  
779 }  
780 }  
781 }  
782 }  
783 }  
784 }  
785 }  
786 }  
787 }  
788 }  
789 }  
790 }  
791 }  
792 }  
793 }  
794 }  
795 }  
796 }  
797 }  
798 }  
799 }  
800 }  
801 }  
802 }  
803 }  
804 }  
805 }  
806 }  
807 }  
808 }  
809 }  
810 }  
811 }  
812 }  
813 }  
814 }  
815 }  
816 }  
817 }  
818 }  
819 }  
820 }  
821 }  
822 }  
823 }  
824 }  
825 }  
826 }  
827 }  
828 }  
829 }  
830 }  
831 }  
832 }  
833 }  
834 }  
835 }  
836 }  
837 }  
838 }  
839 }  
840 }  
841 }  
842 }  
843 }  
844 }  
845 }  
846 }  
847 }  
848 }  
849 }  
850 }  
851 }  
852 }  
853 }  
854 }  
855 }  
856 }  
857 }  
858 }  
859 }  
860 }  
861 }  
862 }  
863 }  
864 }  
865 }  
866 }  
867 }  
868 }  
869 }  
870 }  
871 }  
872 }  
873 }  
874 }  
875 }  
876 }  
877 }  
878 }  
879 }  
880 }  
881 }  
882 }  
883 }  
884 }  
885 }  
886 }  
887 }  
888 }  
889 }  
890 }  
891 }  
892 }  
893 }  
894 }  
895 }  
896 }  
897 }  
898 }  
899 }  
900 }  
901 }  
902 }  
903 }  
904 }  
905 }  
906 }  
907 }  
908 }  
909 }  
910 }  
911 }  
912 }  
913 }  
914 }  
915 }  
916 }  
917 }  
918 }  
919 }  
920 }  
921 }  
922 }  
923 }  
924 }  
925 }  
926 }  
927 }  
928 }  
929 }  
930 }  
931 }  
932 }  
933 }  
934 }  
935 }  
936 }  
937 }  
938 }  
939 }  
940 }  
941 }  
942 }  
943 }  
944 }  
945 }  
946 }  
947 }  
948 }  
949 }  
950 }  
951 }  
952 }  
953 }  
954 }  
955 }  
956 }  
957 }  
958 }  
959 }  
960 }  
961 }  
962 }  
963 }  
964 }  
965 }  
966 }  
967 }  
968 }  
969 }  
970 }  
971 }  
972 }  
973 }  
974 }  
975 }  
976 }  
977 }  
978 }  
979 }  
980 }  
981 }  
982 }  
983 }  
984 }  
985 }  
986 }  
987 }  
988 }  
989 }  
990 }  
991 }  
992 }  
993 }  
994 }  
995 }  
996 }  
997 }  
998 }  
999 }
```



Tabs, tabs, tabs: know the difference





# Hands-on (!github)

## #2 First Touch

 Launch Gitpod

 Experiment with Graph*QL*

 Demo with GraphQL Playground

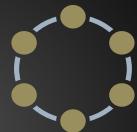


# 01



## Housekeeping Live and Hands-on

# 02



## Database Setup Data model & Astra DB

# 03



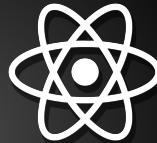
## GraphQL Design & Toolings

# 04



## Backend Expose GQL endpoint

# 05



## Frontend Consume GQL Endpoint

# 06

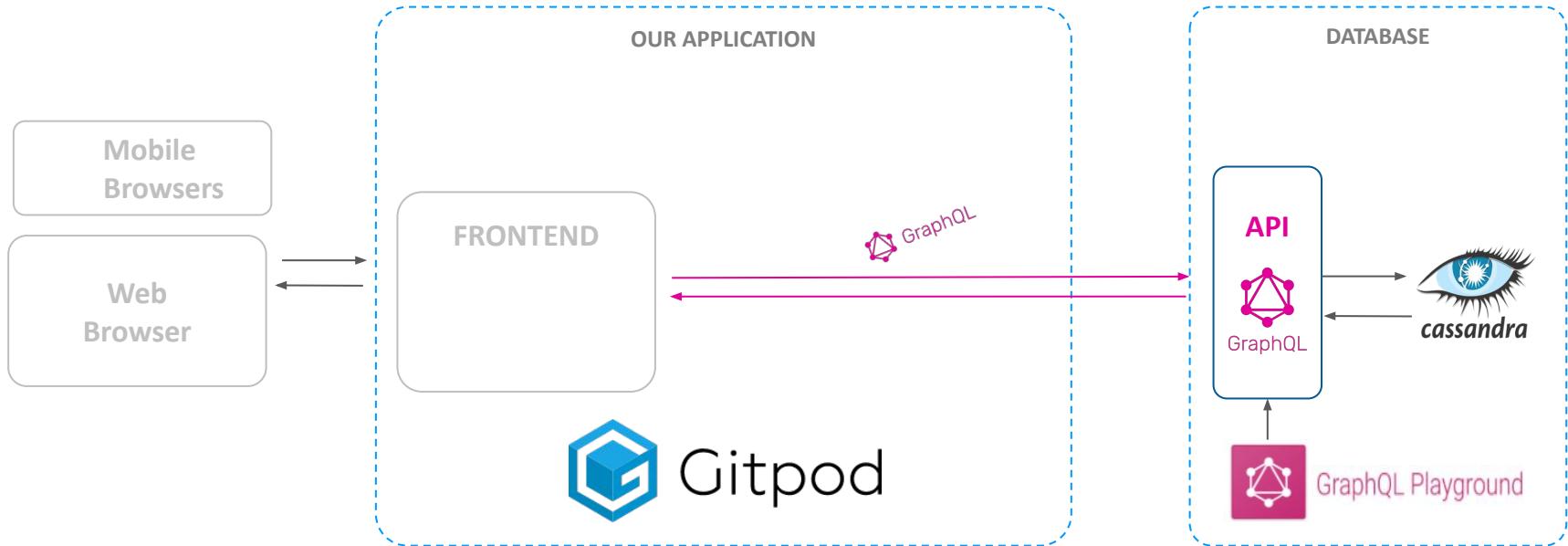


## What's next? Quiz, Homework, ...

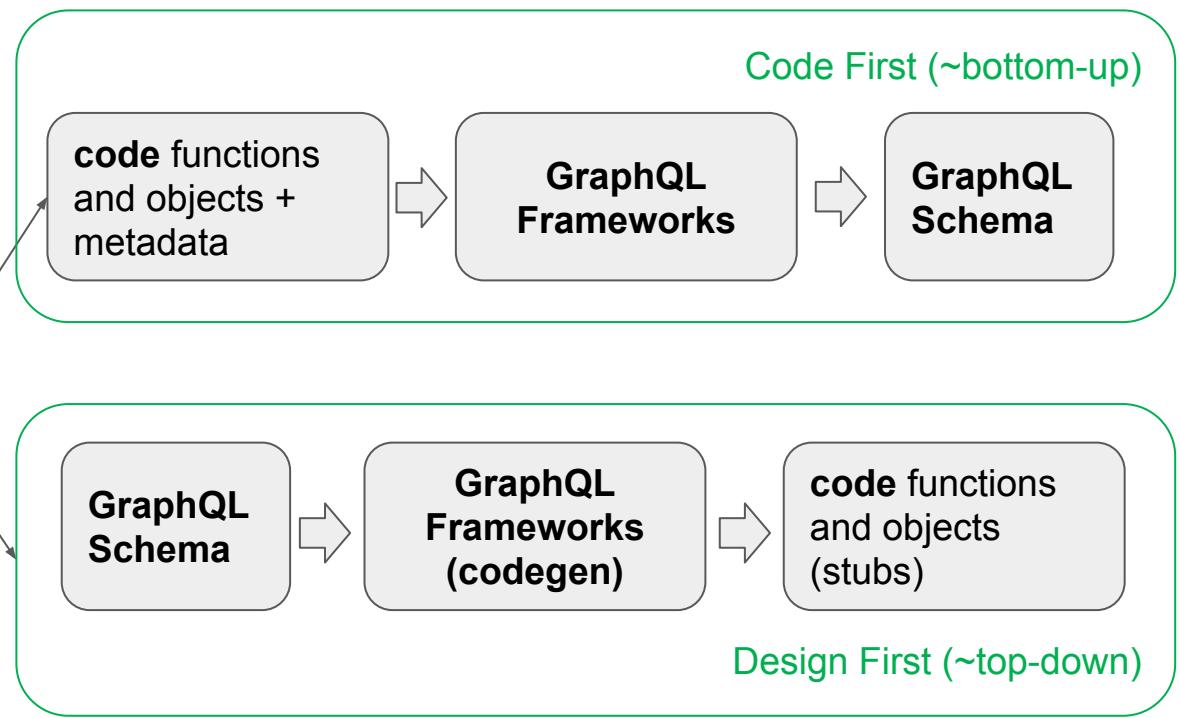
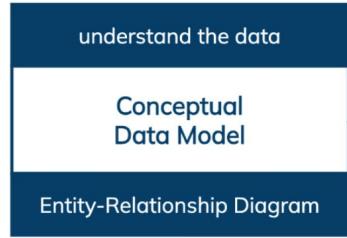


## Agenda

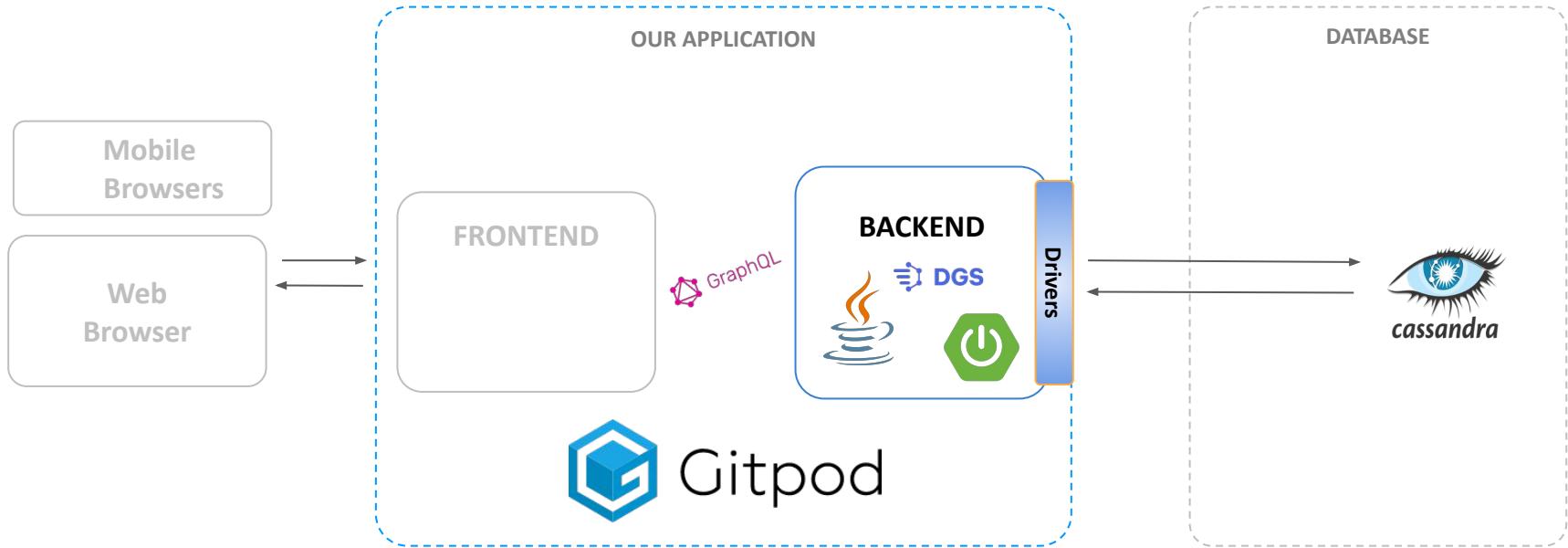




Sooo Connect the Front end to Graphql



Build a GraphQL API



## Introduction to DGS by Netflix



```
public class Genre {  
    private final String value;  
  
    public Genre(String value) {  
        this.value = value;  
    }  
  
    public String getValue() {  
        return value;  
    }  
}
```



```
@DgsComponent  
public class GenresDatafetcher {  
    Logger logger = Logger.getLogger(GenresDatafetcher.class.getName());  
  
    private final List<Genre> genres = List.of(  
        new Genre("Action"),  
        new Genre("Anime"),  
        new Genre("Award-Winning"),  
        new Genre("Children & Family"),  
        new Genre("Comedies")  
);  
  
@DgsQuery  
public List<Genre> genres(@InputArgument String labelFilter) {  
    logger.info("QUERY executed - Genres value is: " + genres.get(0).getValue());  
  
    if(labelFilter == null) {  
        return genres;  
    }  
  
    return genres.stream().filter(s -> s.getValue().contains(labelFilter)).collect(Collectors.toList());  
}
```

## DataFetcher as @DgsComponent

## Expose query with @DgsQuery



Exposing a GQL Endpoint with

# 01



## Housekeeping Live and Hands-on

# 02



## Database Setup Data model & Astra DB

# 03



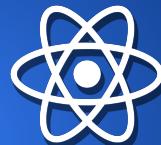
## GraphQL Design & Toolings

# 04



## Backend Expose GQL endpoint

# 05



## Frontend Consume GQL Endpoint

# 06

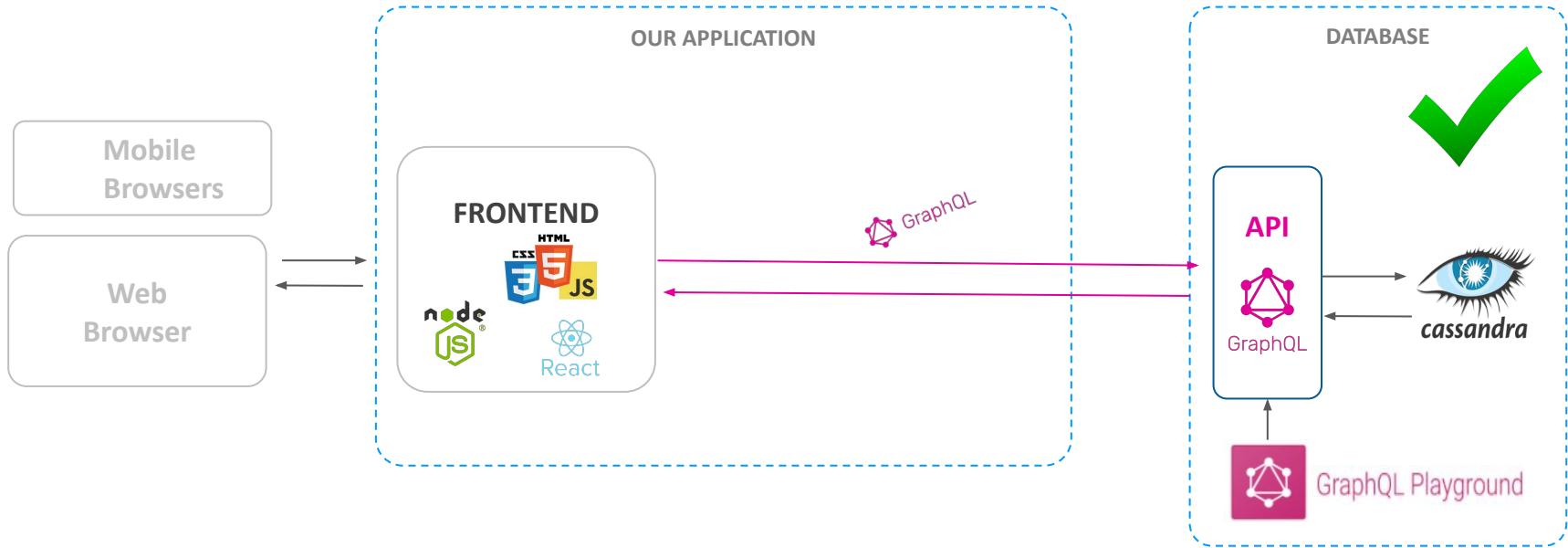


## What's next? Quiz, Homework, ...

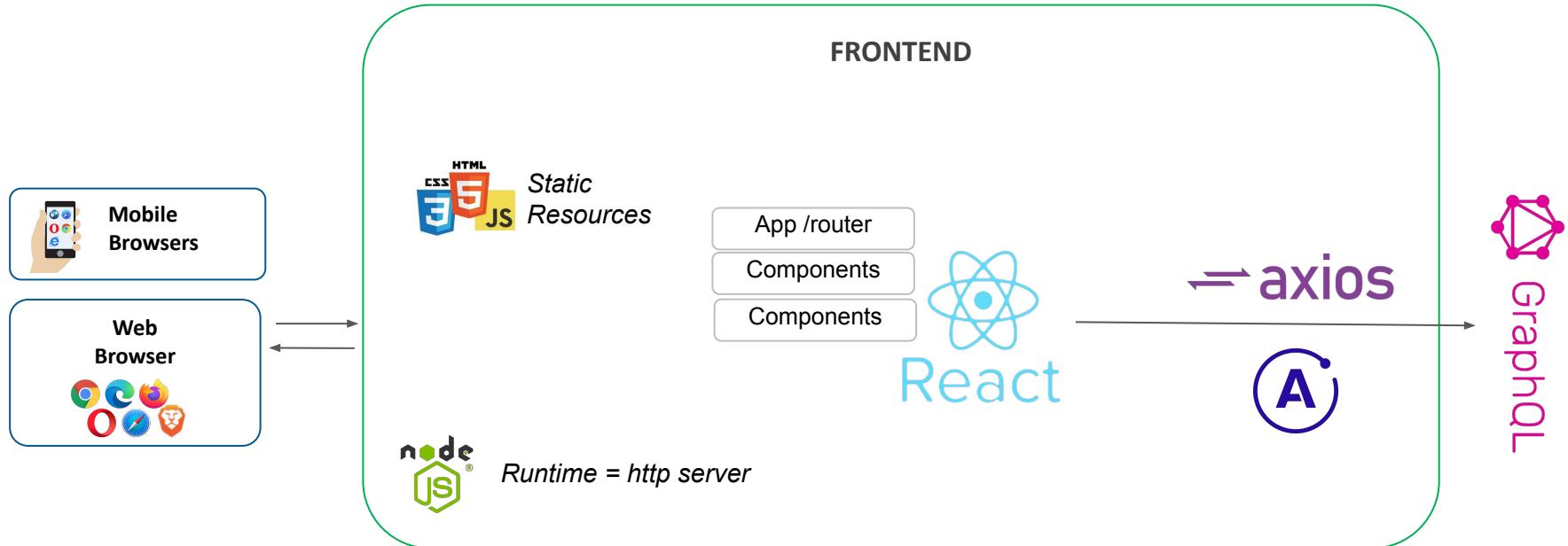


## Agenda





Sooo Connect the Front end to Graphql

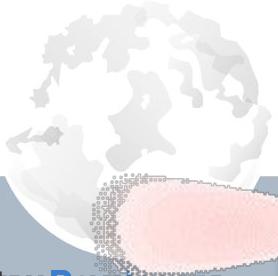


(\*) Check out Apollo's `@apollo/client` library as well ...



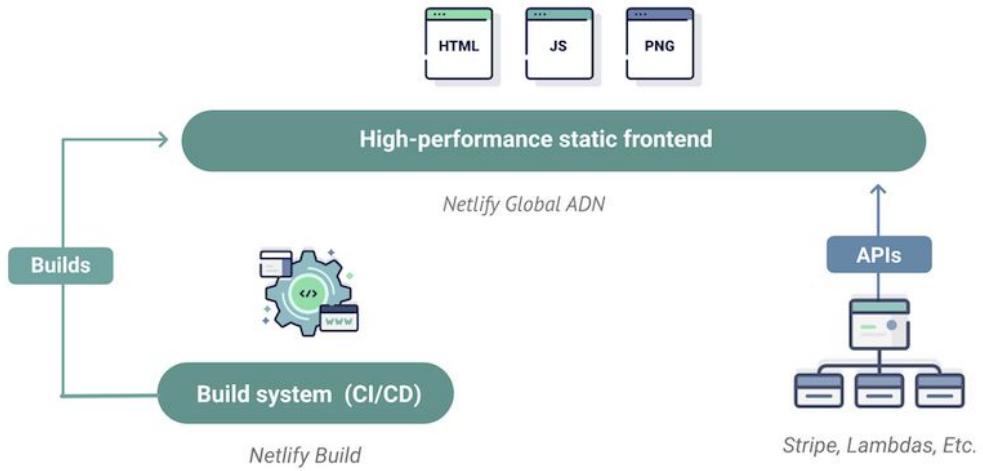
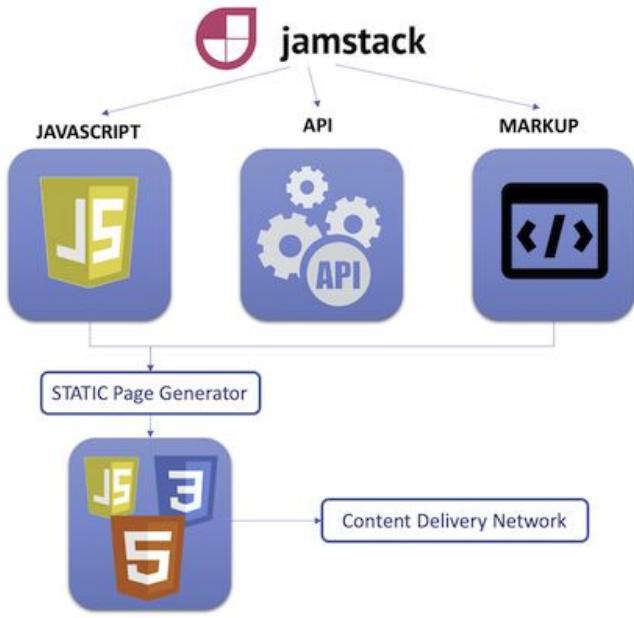
Our Front End



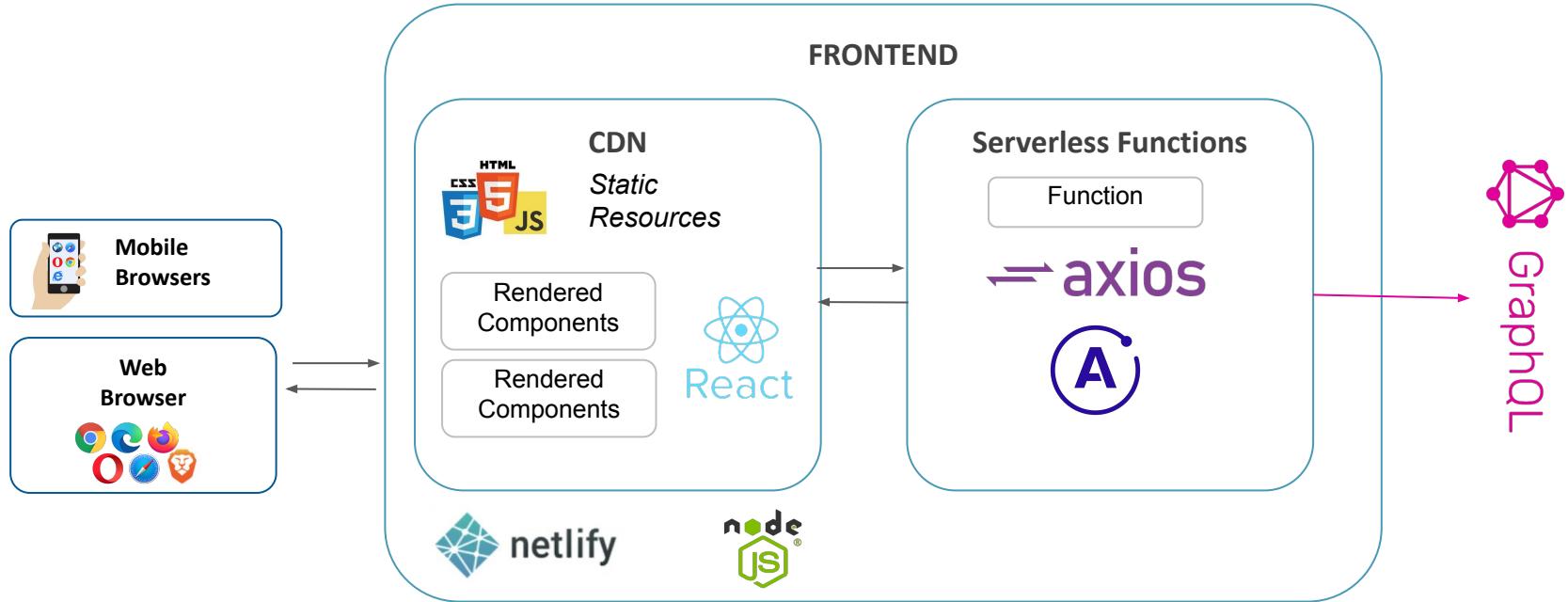


Cops are coming for you

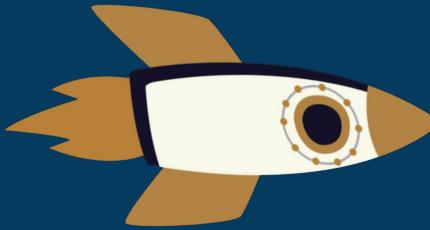




## Introduction to Netlify



Our Front End



# Hands-on (!github)

## #3 Run the app

- ✓ React (start client, examine code)
- ✓ DB Connectivity (link app and DB)
- ✓ Run the final app (connected to DB, last touch)

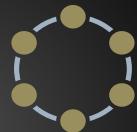


# 01



## Housekeeping Live and Hands-on

# 02



## Database Setup Data model & Astra DB

# 03



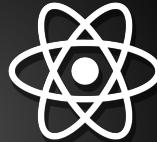
## GraphQL Design & Toolings

# 04



## Backend Expose GQL endpoint

# 05



## Frontend Consume GQL Endpoint

# 06



## What's next? Quiz, Homework, ...



## Agenda



# What's next?



Quiz, Homework, ...



- **Homework**

- Complete the practice steps after step #9
- Insert (mutate) a show or genre of your choice in the database.
- Submit screenshot to us ... and wait for your **badge!**



- Optionally

- Watch "Netflix Clone GraphQL" workshop recording

- Do reach us for anything: [dtsx.io/discord](https://dtsx.io/discord) !



All info on Github – !homework



Assignment and Badge

The screenshot shows the DataStax Developers Discord server interface. On the left, there's a sidebar with categories like Événements, moderator-only, WELCOME, start-here, code-of-conduct, introductions, upcoming-events, useful-resources, memes, your-ideas, @the-stage, WORKSHOPS, workshop-chat, workshop-feedback, workshop-materials, upcoming-workshops, ASTRADB, getting-started, astra-apis, astra-development, sample-applications, and APACHE CASSANDRA. The main area is the #workshop-chat channel, which has a video player showing a presentation slide with a green cross logo and several names (Taymireya, Tegimberia, Nefelous, Artyebelona, Drivobekas). Below the video, a message from RIGGITYREKT at 21:14 discusses mixed version testing for DSE 5.1.20 nodes. Another message from RIGGITYREKT at 23:39 asks how to fix a configuration issue. A message from Erick Ramirez at 02:19 explains the issue. Cedrick Lunen at 09:01 provides a solution involving parameters -k, -g, and -s. The right side of the screen lists PRESENTER — 1 (David Jones-Gilardi), HELPER — 7 (012345, AaronP, B1nary, Chelsea Navo, Jeremy Hanna, John Sanda, Patrick\_McFadin), and EN LIGNE — 560 users (-samu-, 6304-42JB, Aahlya, Abdurahim, abhi3pathi, Abhiis.s, Abhineet, Abirsh). At the bottom, there's a button to "Envoyer un message dans #workshop-chat".

Datastax Developers Discord (19k+)

!discord

dtsx.io/discord

# Subscribe



# Subscribe



How to create an Authentication Token in...

37 views • 4 weeks ago

How to use the Data Loader in Astra DB

62 views • 4 weeks ago

Astra DB Sample App Gallery

36 views • 4 weeks ago

How to use Secure Connect in Astra DB

42 views • 4 weeks ago

Cassandra Day India: CL Room (Workshops)

2.4K views • Streamed 4 weeks ago

Cassandra Day India: RF Room (Talks)

1.3K views • Streamed 1 month ago

# Thank You!

