GraphQL with Apollo


airbnb

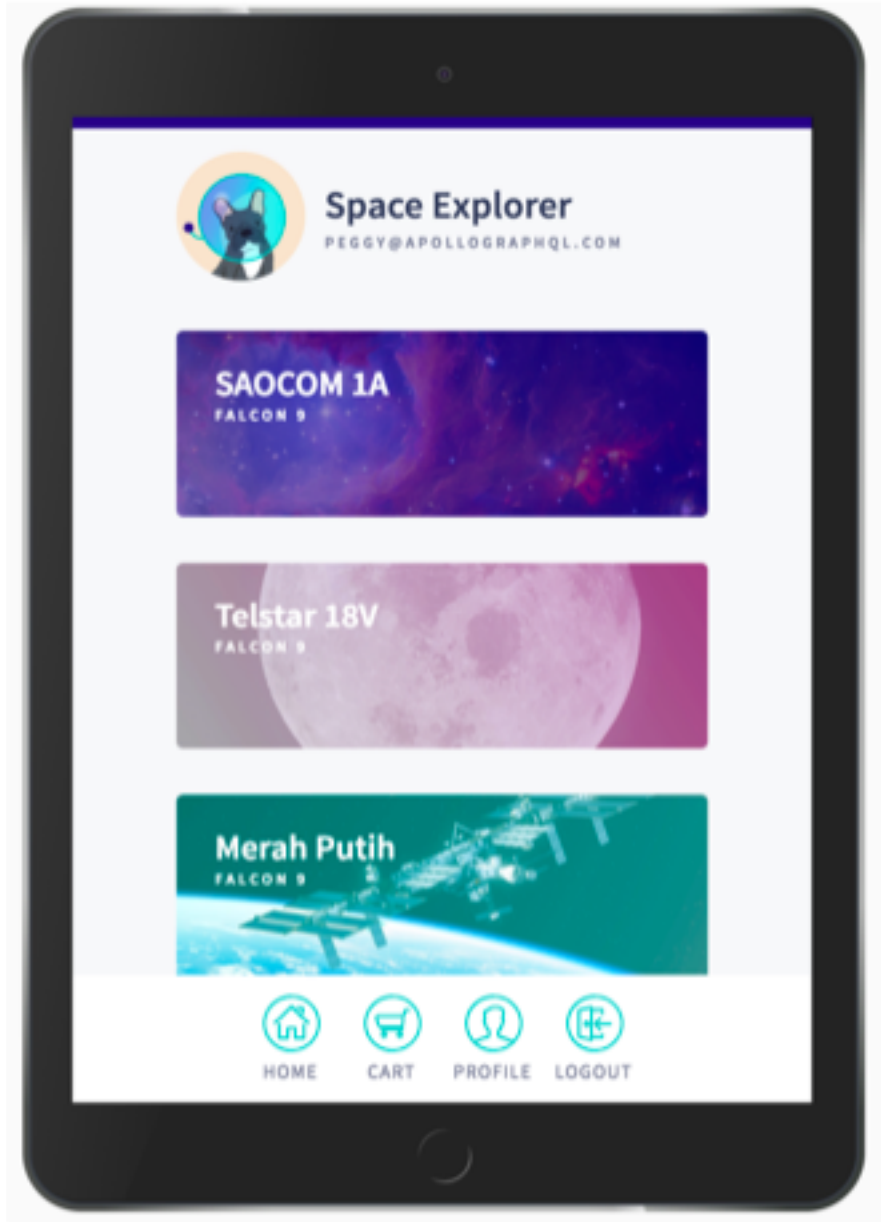

neo4j

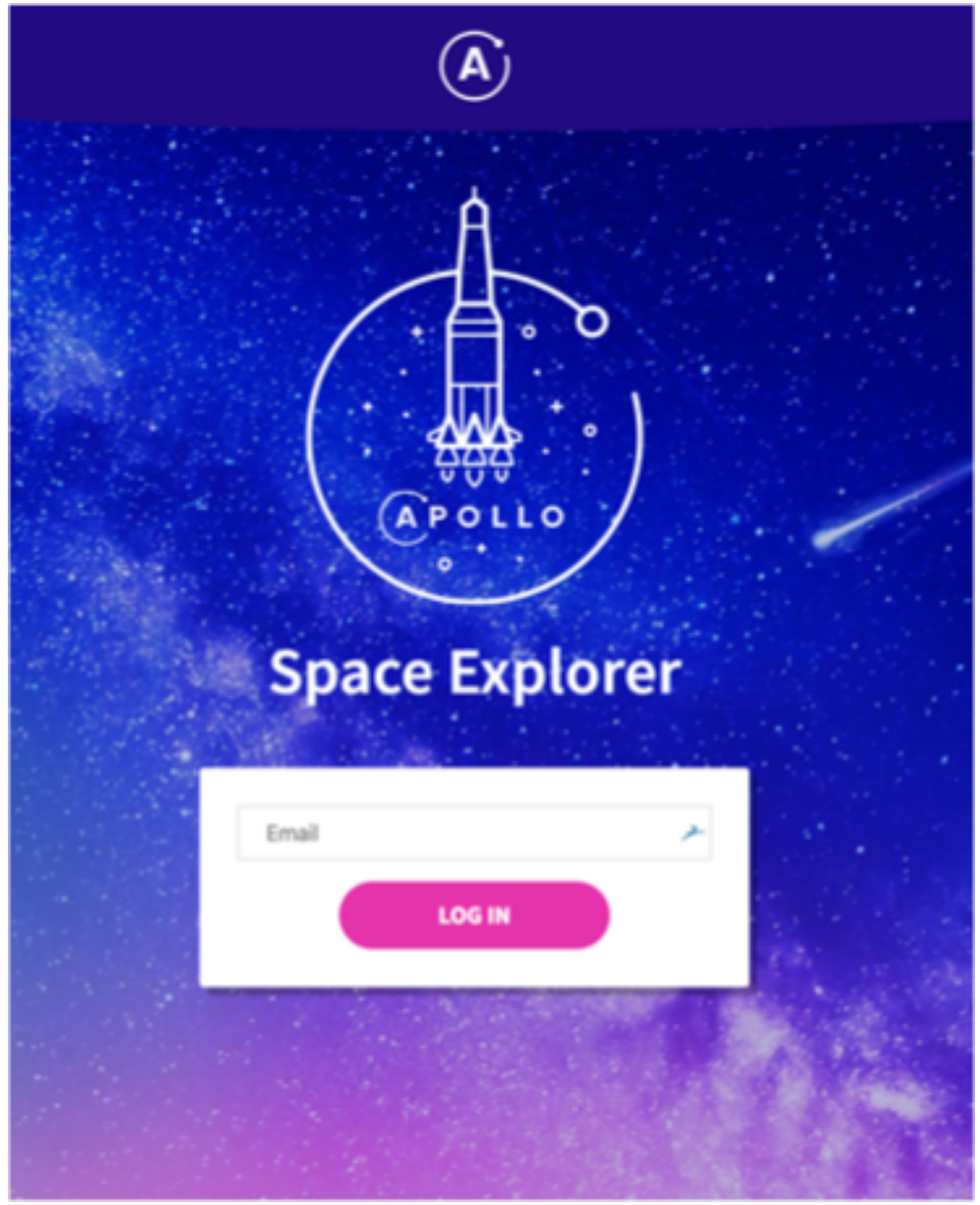

The
New York
Times


KLM


coursera


ATLASSIAN


Space Explorer
PEGGY@APOLLOGRAPHQL.COM
SAOCOM 1A
FALCON 9
Telstar 18V
FALCON 9
Merah Putih
FALCON 9
HOME
CART
PROFILE
LOGOUT


APOLLO
Space Explorer
Email
LOG IN

CCtCap Demo Mission 1

P.EIJGERMANS11@GMAIL.COM

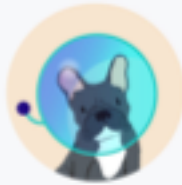
ADD TO CART


HOME


CART


PROFILE


LOGOUT

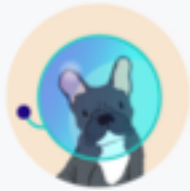


My Cart

P.EIJGERMANS11@GMAIL.COM

BOOK ALL

HOME CART PROFILE LOGOUT



My Trips

P.EIJGERMANS11@GMAIL.COM

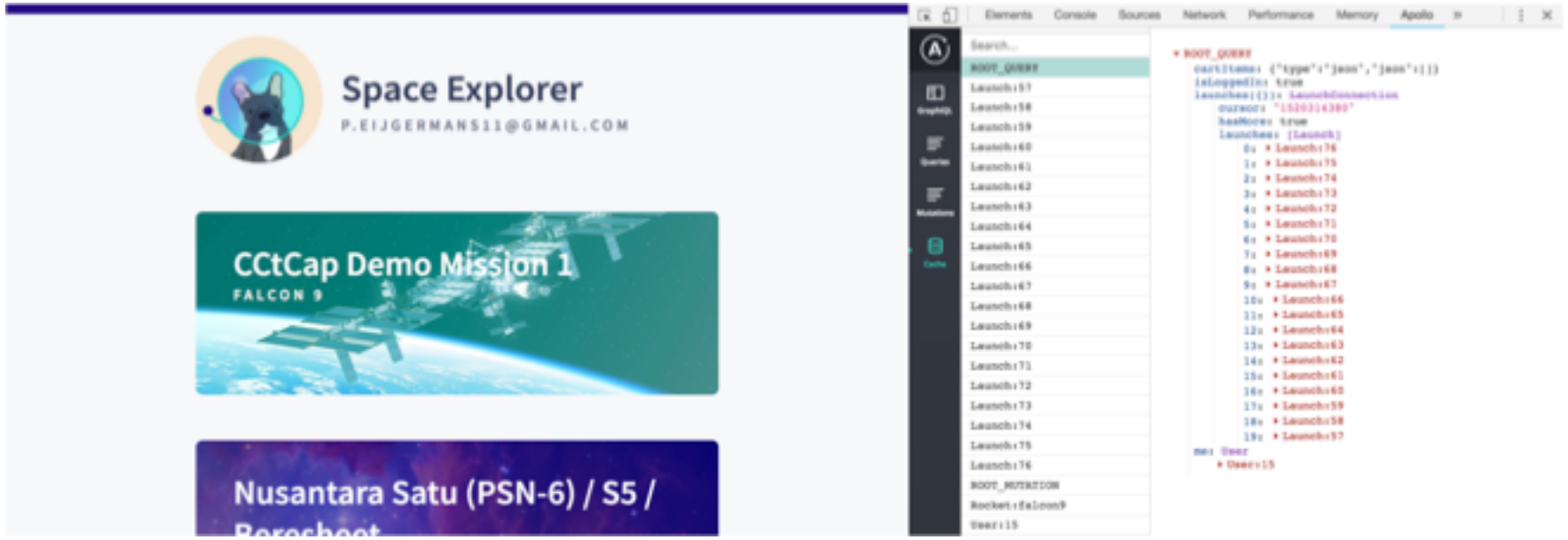
Prerequisites

- **System requirements**
- Before we begin, make sure you have:
- Node.js v6.9.0 or greater
- npm 3.10.8 or greater
- git v2.14.1 or greater

- Visual Studio Code installed

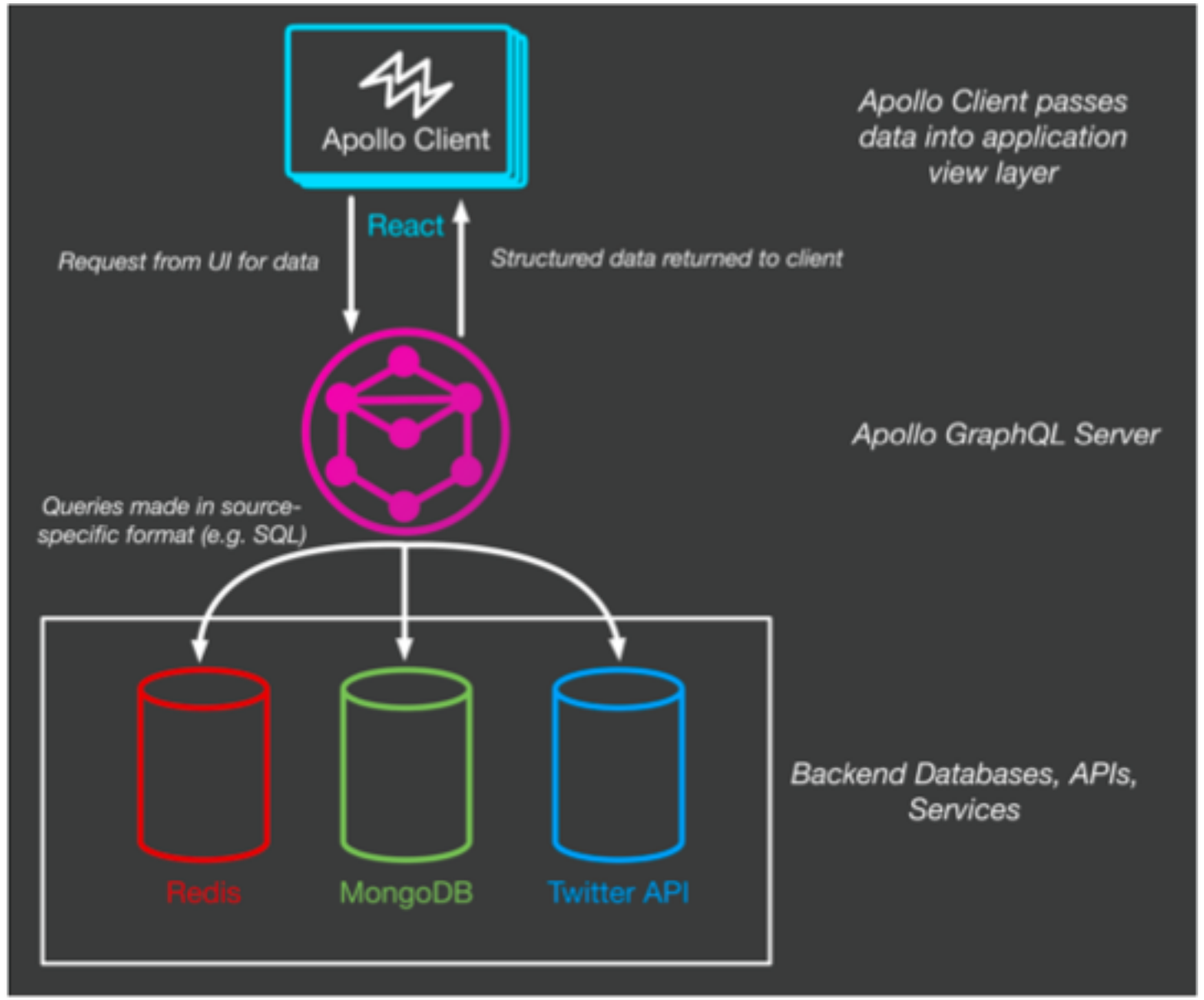
Set up your development environment

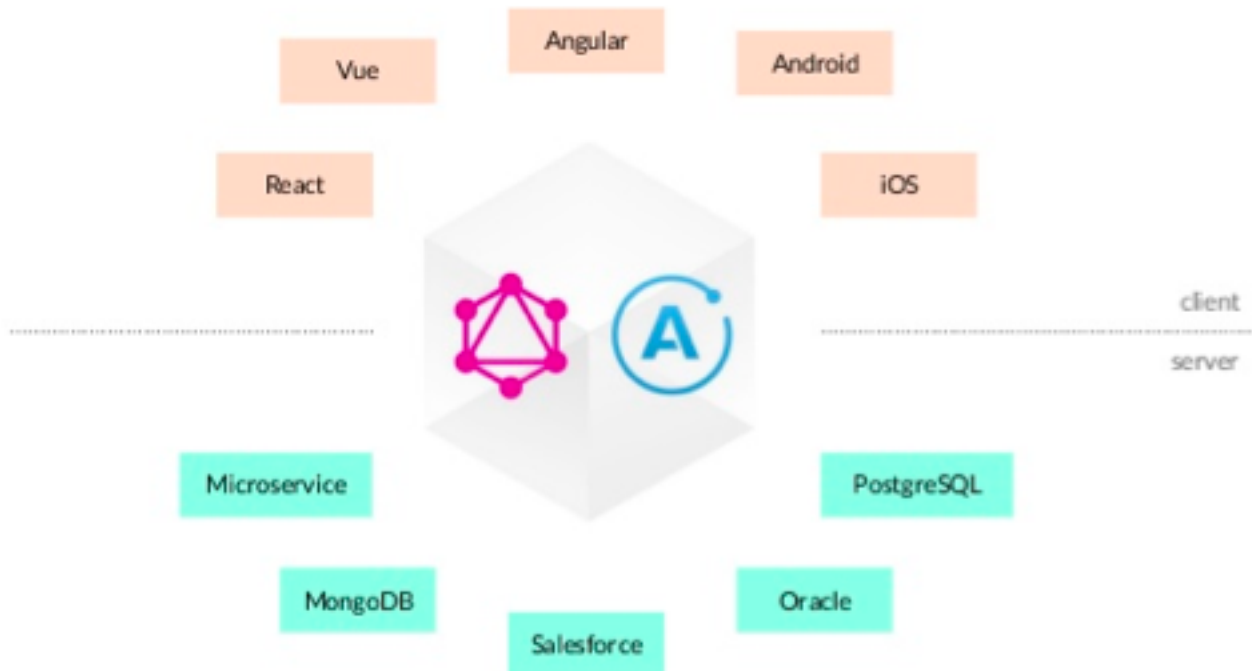
- Apollo DevTools for Chrome (suggested) : Our Chrome extension giving you full visibility into your client.



Next, in your terminal, clone this repository:

```
git clone https://github.com/apollographql/fullstack-tutorial/
```


Apollo Client
Apollo Client passes data into application view layer
React
Request from UI for data
Structured data returned to client
Apollo GraphQL Server
Queries made in source-specific format (e.g. SQL)
Redis
MongoDB
Twitter API
Backend Databases, APIs, Services


Angular
Vue
Android
React
iOS
client
server
Microservice
PostgreSQL
MongoDB
Oracle
Salesforce

Apollo Client

- Define **queries** as part of UI components;
- Binding query results to the UI
- Caching
- State management (**no Redux needed**)
- Integrations for React, React Native, Vue, Angular ect.

Apollo Server

- Encapsulates boilerplate code
- Defining a *schema* and a set of *resolvers*
- Can be **layered over your existing services, including REST APIs and databases**
- Improved Performance → smaller payload
- Caching
- State management

Apollo Server

1. CACHING

Load less data and save resources

2. TRANSPARENCY

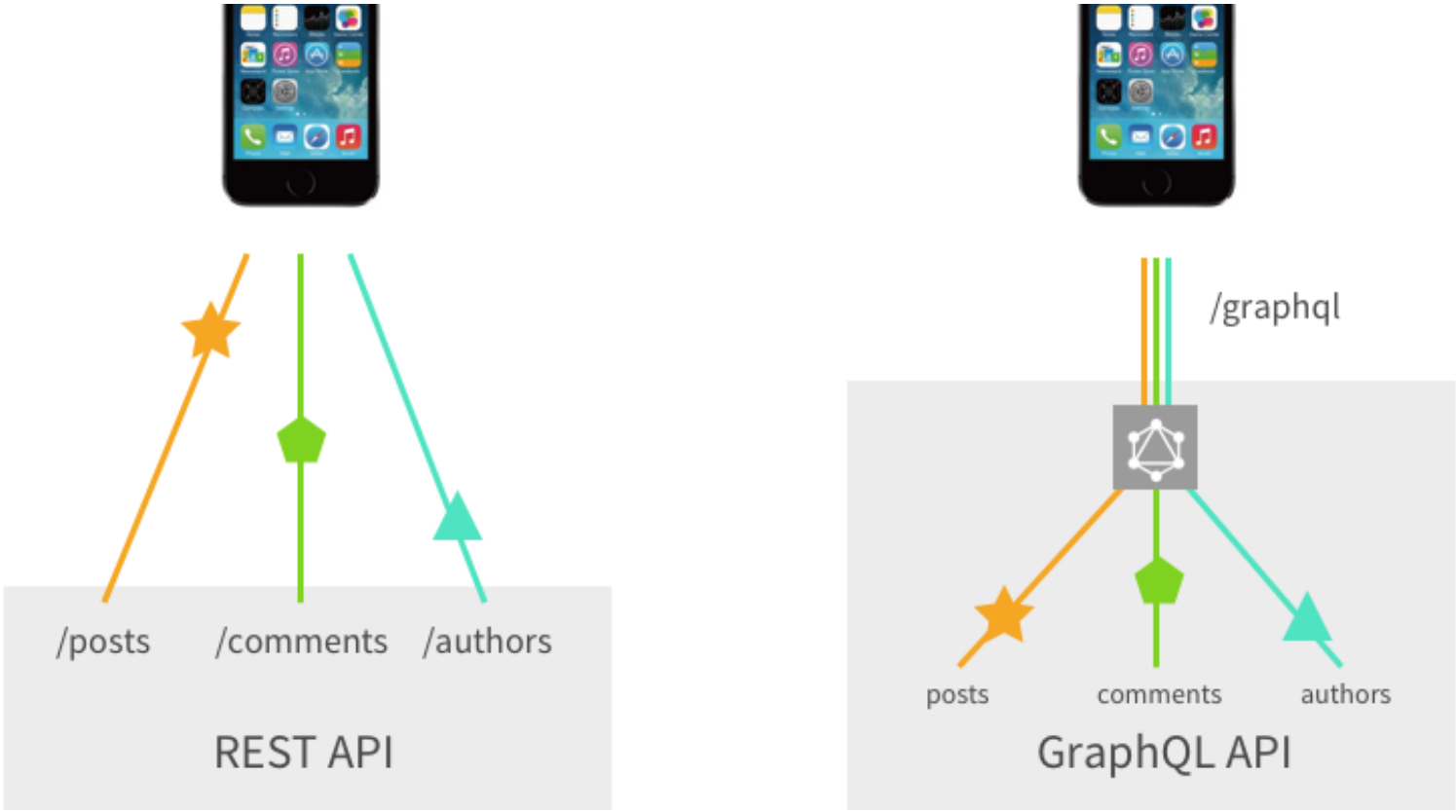
Understand what's happening

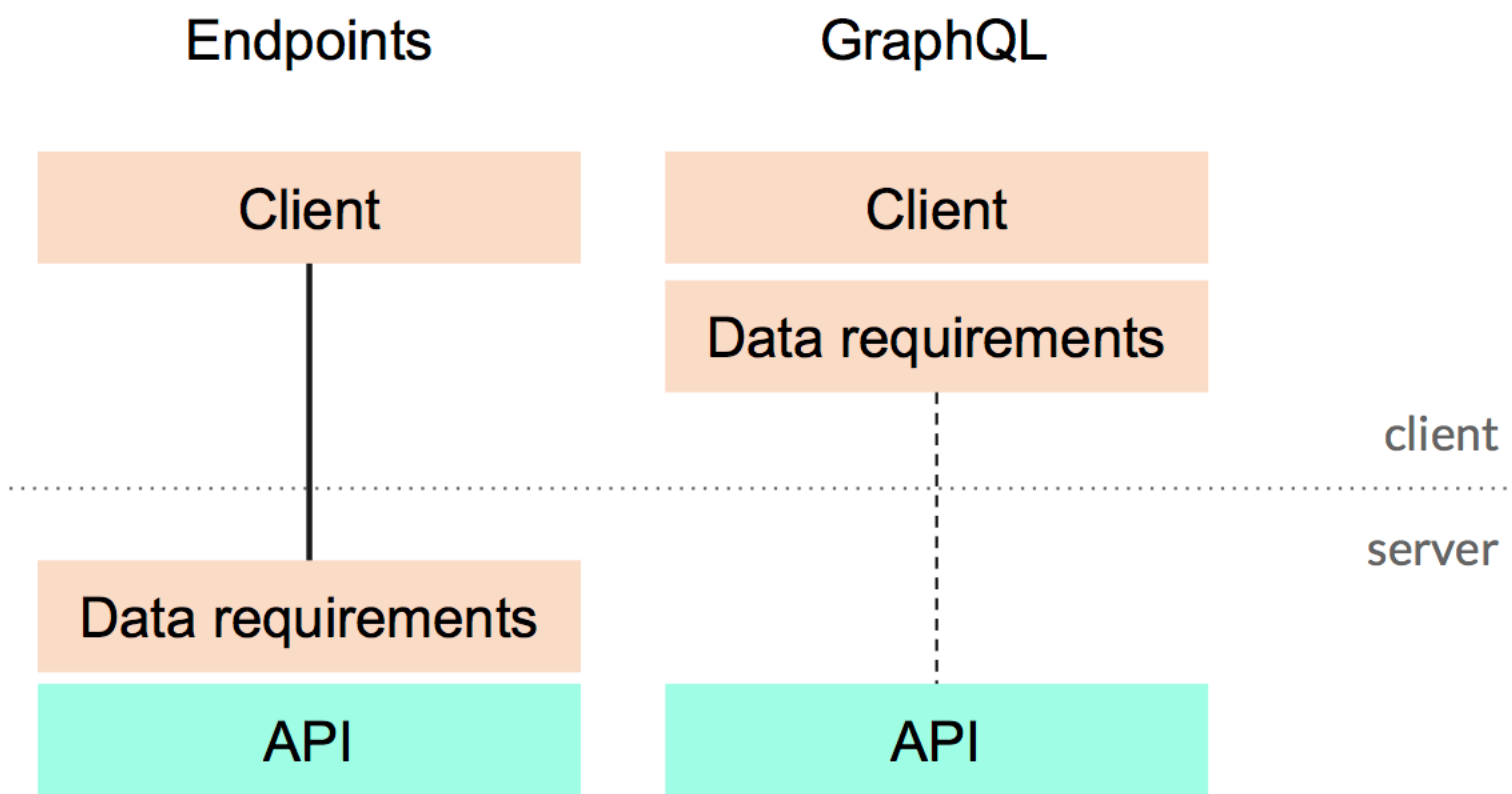
3. MODULARITY

Combine parts into a seamless whole

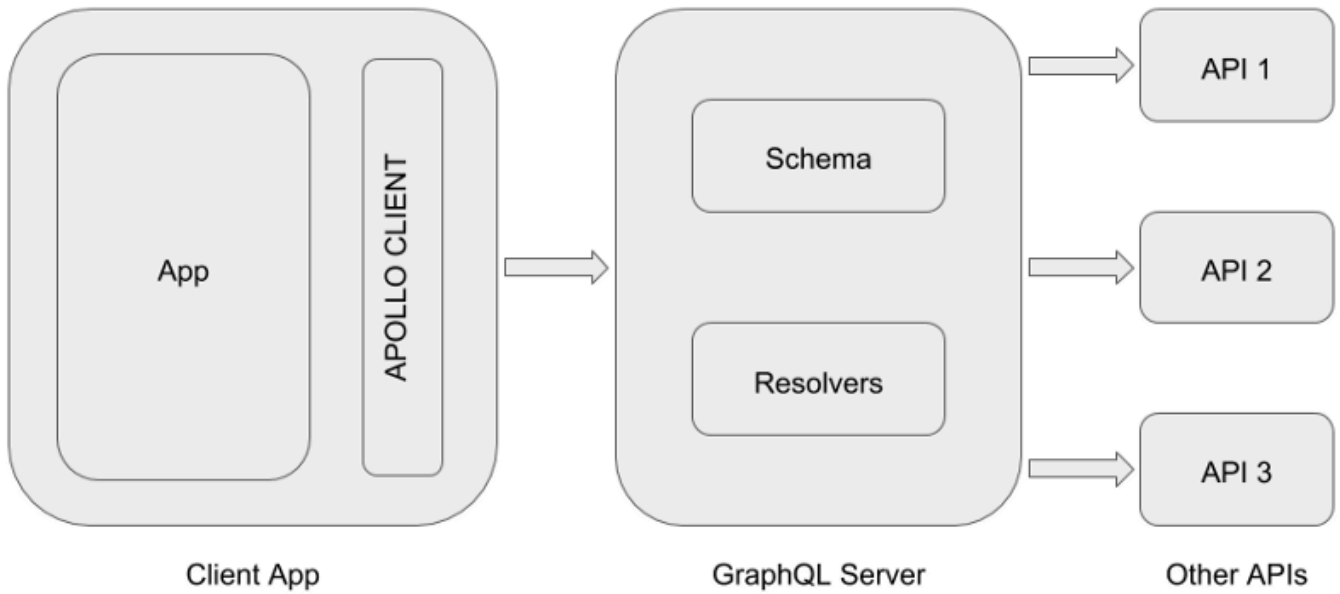
Why GraphQL?

Get many resources in a single request




Endpoints
GraphQL
Client
Client
Data requirements
client
server
Data requirements
API
API

GraphQL traditional approach



Step 1: Build a schema

Create a blueprint for your graph's data

Schema concepts

A GraphQL schema is the center
of any GraphQL server

Our schema will be based on these features:

Queries

Fetch all upcoming rocket launches

Fetch a specific launch by its ID

Mutations

Book launch trips if the user is logged in

Cancel launch trips if the user is logged in

Login the user

schema.js

```
const { gql } = require('apollo-server');

const typeDefs = gql`
  type Query {
    launches: [Launch]!
    launch(id: ID!): Launch
    # Queries for the current user
    me: User
  }

  type Launch {
    id: ID!
    site: String
    mission: Mission
    rocket: Rocket
    isBooked: Boolean!
  }

  type User {
    id: ID!
    email: String!
    trips: [Launch]!
  }
`

module.exports = typeDefs;
```

Our schema will be based on these features:

Queries

Fetch all upcoming rocket launches

Fetch a specific launch by its ID

Mutations

Book launch trips if the user is logged in

Cancel launch trips if the user is logged in

Login the user

Mutation

```
type Mutation {
  # if false, signup failed -- check errors
  bookTrips(launchIds: [ID]!): TripUpdateResponse!

  # if false, cancellation failed -- check errors
  cancelTrip(launchId: ID!): TripUpdateResponse!

  login(email: String): String # login token
}

type TripUpdateResponse {
  success: Boolean!
  message: String
  launches: [Launch]
}
```

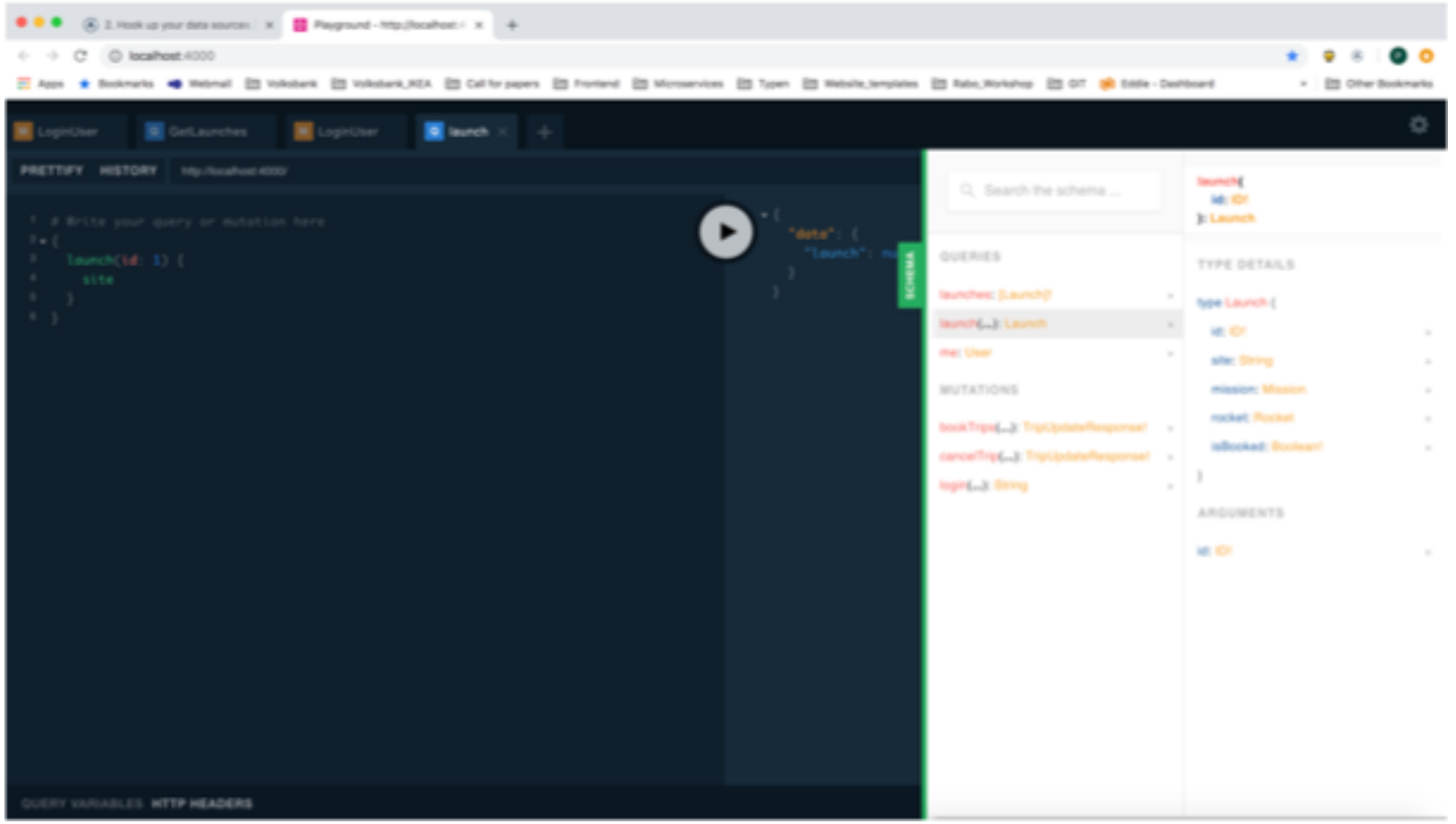
Define your Apollo server
+ npm start

src/index.js

```
1  const { ApolloServer } = require('apollo-server');
2  const typeDefs = require('./schema');
3
4  const server = new ApolloServer({ typeDefs });
5
6  server.listen().then(({ url }) => {
7    console.log(`🚀  Server ready at ${url}`);
8  });
```

Explore your schema

in GraphQL Playground



Step 2: Hook up your data sources

Connect REST data to your graph/schema

Connect a REST API Data Source

- First, let's connect the Space-X v2 REST API to our schema.

- Install the **apollo-datasource-restpackage**:

```
1 | npm install apollo-datasource-rest --save
```

Define DataSource (*launch.js*)

```
const { RESTDataSource } = require('apollo-datasource-rest');

class LaunchAPI extends RESTDataSource {
  constructor() {
    super();
    this.baseURL = 'https://api.spacexdata.com/v2/';
  }
}

module.exports = LaunchAPI;
```

launch.js

```
async getAllLaunches() {
  const response = await this.get('launches');

  // transform the raw launches to a more friendly
  return Array.isArray(response)
    ? response.map(launch => this.launchReducer(launch)) : [];
}
```

schema.js

```
const { gql } = require('apollo-server');

const typeDefs = gql`
  type Query {
    launches: [Launch]!
    launch(id: ID!): Launch
    # Queries for the current user
    me: User
  }

  type Launch {
    id: ID!
    site: String
    mission: Mission
    rocket: Rocket
    isBooked: Boolean!
  }

  type User {
    id: ID!
    email: String!
    trips: [Launch]!
  }
`

module.exports = typeDefs;
```

Add launchReducer in launch.js

```
// leaving this inside the class to make the class easier to test
launchReducer(launch) {
  return {
    id: launch.flight_number || 0,
    cursor: `${launch.launch_date_unix}`,
    site: launch.launch_site && launch.launch_site.site_name,
    mission: {
      name: launch.mission_name,
      missionPatchSmall: launch.links.mission_patch_small,
      missionPatchLarge: launch.links.mission_patch,
    },
    rocket: {
      id: launch.rocket.rocket_id,
      name: launch.rocket.rocket_name,
      type: launch.rocket.rocket_type,
    },
  };
}
```

Add Data Source to Apollo Server (*index.js*)

```
const { ApolloServer } = require('apollo-server');
const typeDefs = require('./schema');
const LaunchAPI = require('./datasources/launch');

const server = new ApolloServer({
  typeDefs,
  dataSources: () => ({
    launchAPI: new LaunchAPI()
  })
});

server.listen().then(({ url }) => {
  console.log(`🚀 Server ready at ${url}`);
});
```

Step 3: Write your graph's *Resolvers*

Learn how a GraphQL query fetches data

What is a Resolver?

Resolvers provide the instructions for turning a GraphQL operation (a query, mutation, or subscription) into data.

Connecting Resolvers to Apollo Server

```
const { ApolloServer } = require('apollo-server');
const typeDefs = require('./schema');
const resolvers = require('./resolvers');
const LaunchAPI = require('./datasources/launch');

const server = new ApolloServer({
  typeDefs,
  resolvers,
  dataSources: () => ({
    launchAPI: new LaunchAPI()
  })
});

server.listen().then(({ url }) => {
  console.log(`🚀 Server ready at ${url}`);
});
```

Syntax Resolver

```
1   fieldName: (parent, args, context, info) => data
```

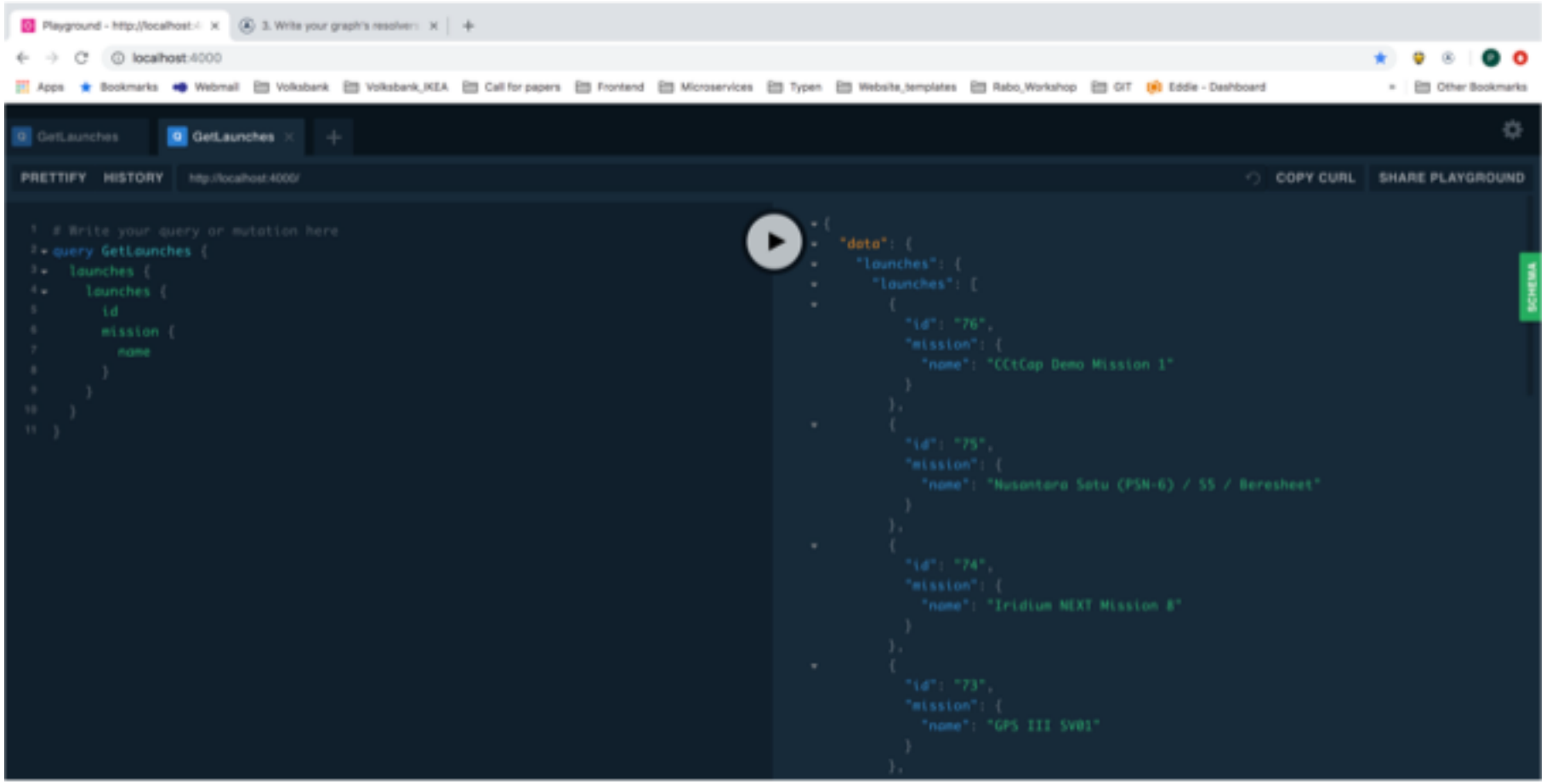
Write Query Resolvers (*resolvers.js*)

src/resolvers.js

```
module.exports = {
  Query: {
    launches: async (_, __, { dataSources }) =>
      dataSources.launchAPI.getAllLaunches(),
    launch: (_, { id }, { dataSources }) =>
      dataSources.launchAPI.getLaunchById({ launchId: id }),
    me: async (_, __, { dataSources }) =>
      dataSources.userAPI.findOrCreateUser(),
  },
};
```

Run Queries

in GraphQL Playground



Final flow

Resolver → Data Source → Reducer → Schema

Start Tutorial

- https://www.apollographql.com/docs/tutorial/schema.html

Skip step 4: “Run your graph in production”

Instructions

- There are two folders: one for the **starting point** (start/) and one for the **final version** (final).
- Within each directory are two folders: one for the **server** and one for the **client**.
- We will be working in the **server folder** first.