GraphQL with Apollo

airbnb

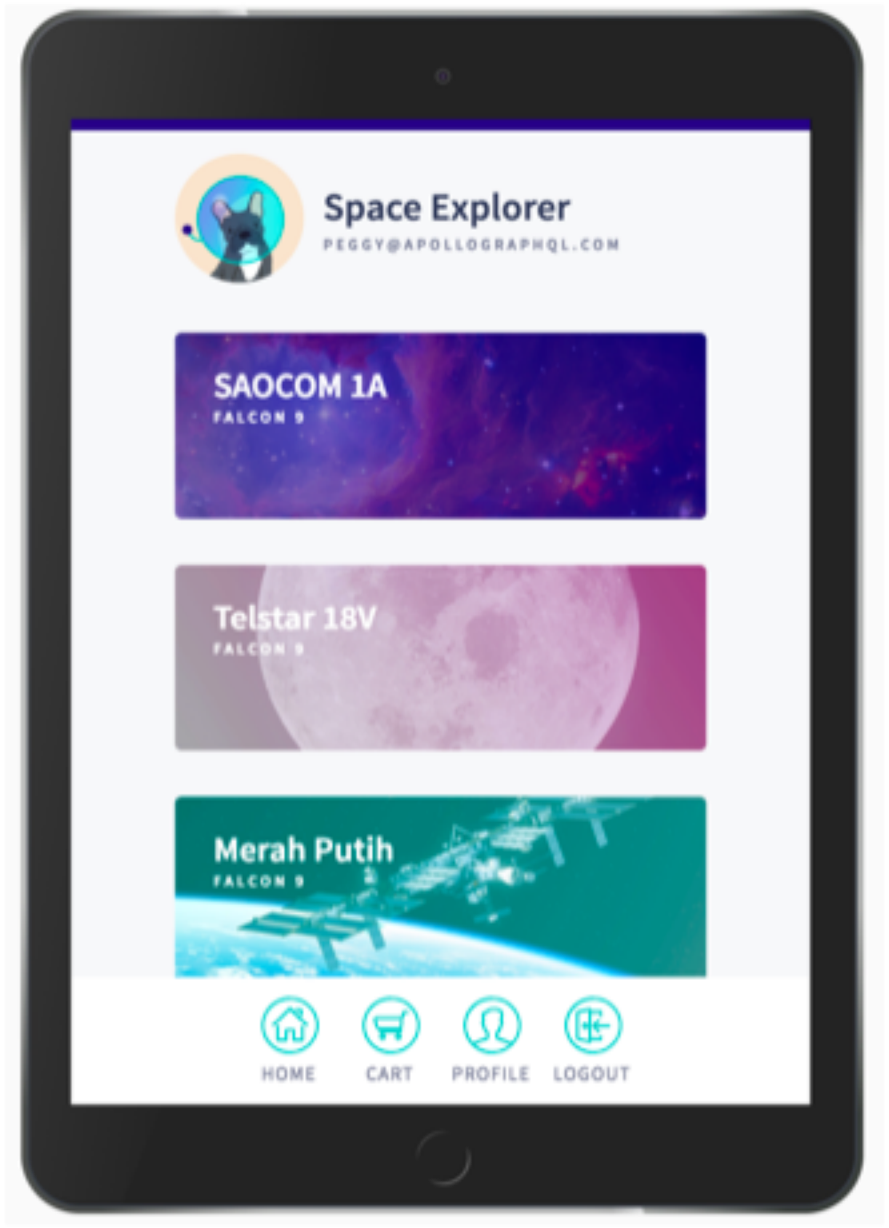
neo4j

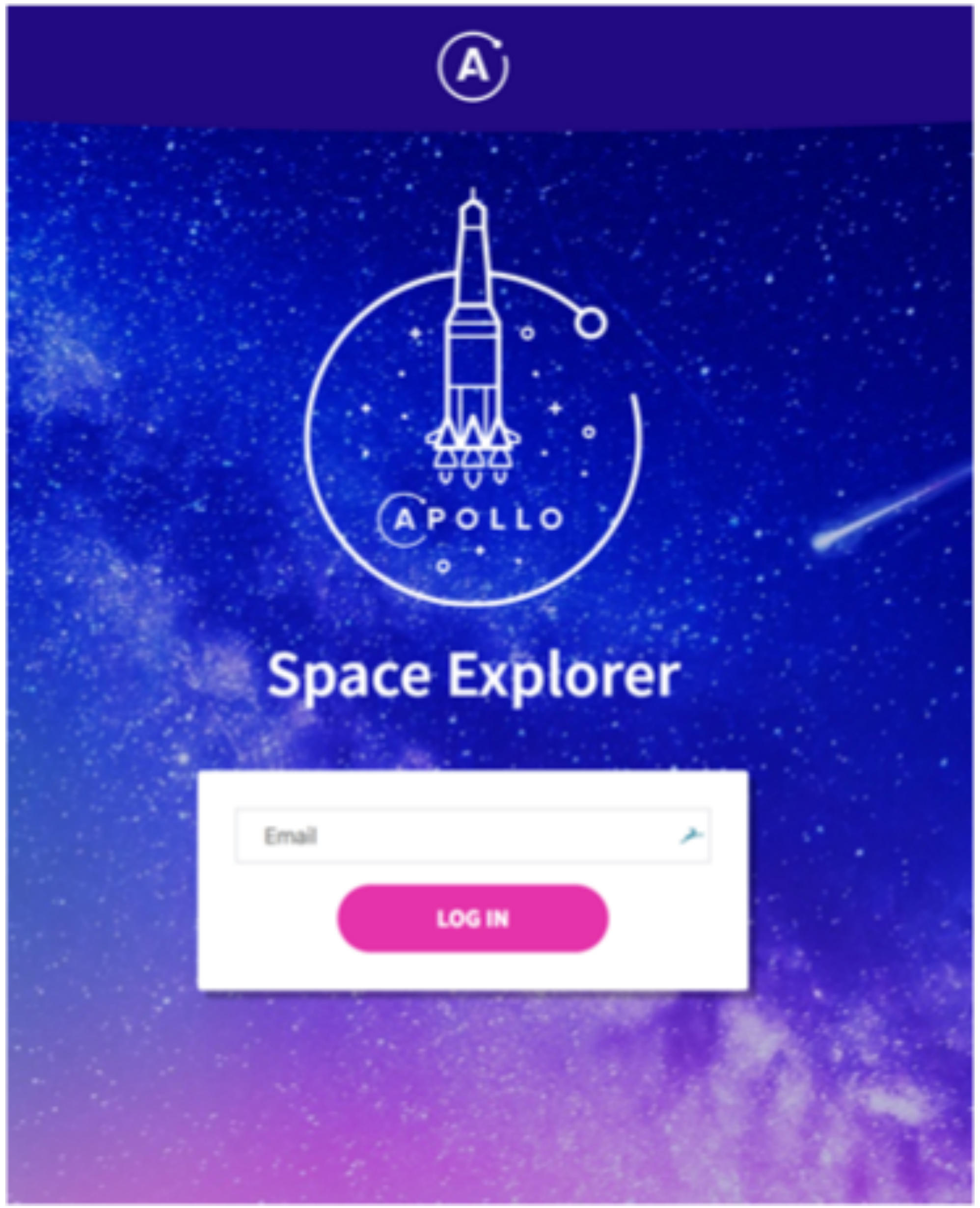
The New York Times

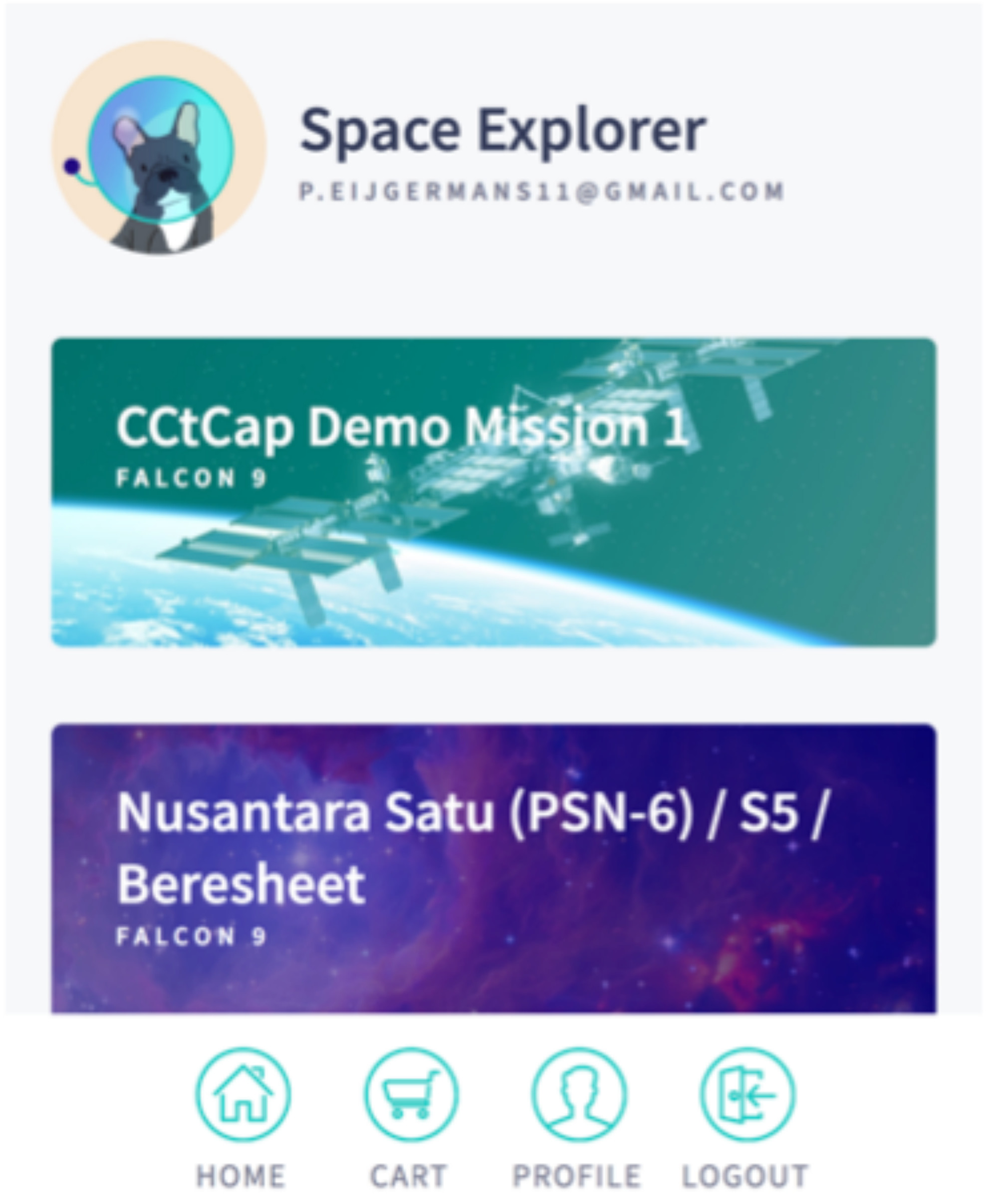
KLM

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



Space Explorer

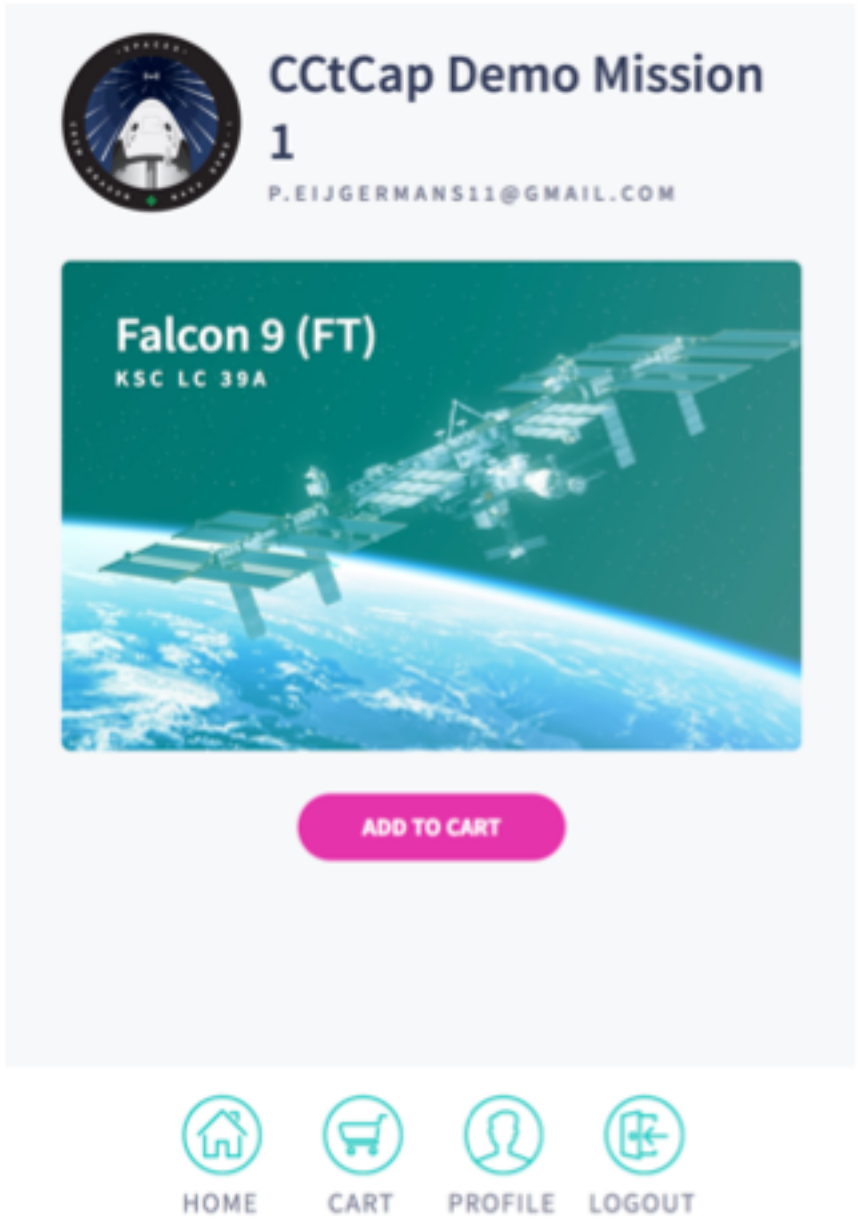
P.EIJGERMANS11@GMAIL.COM

CCtCap Demo Mission 1

FALCON 9

Nusantara Satu (PSN-6) / S5 / Beresheet

FALCON 9

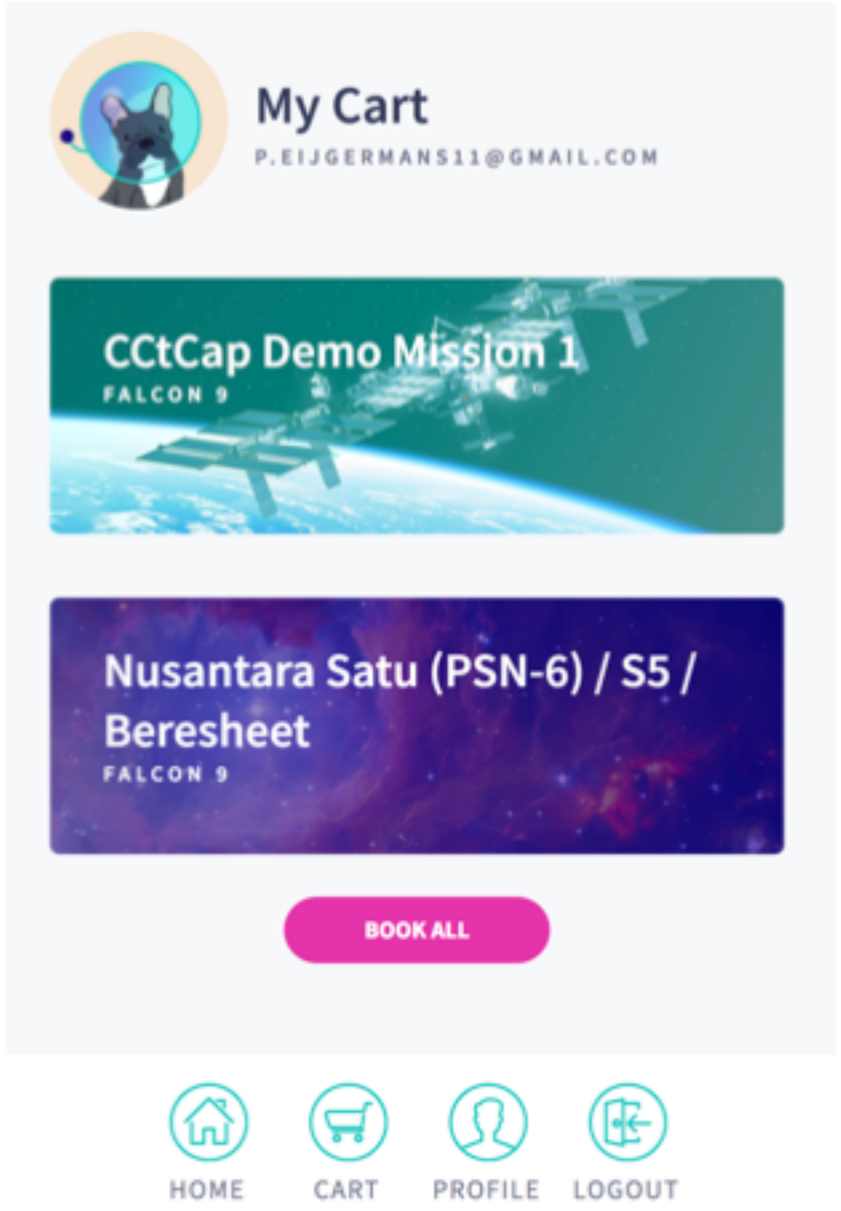


CCtCap Demo Mission 1

P.EIJGERMANS11@GMAIL.COM

Falcon 9 (FT)

KSC LC 39A



My Cart

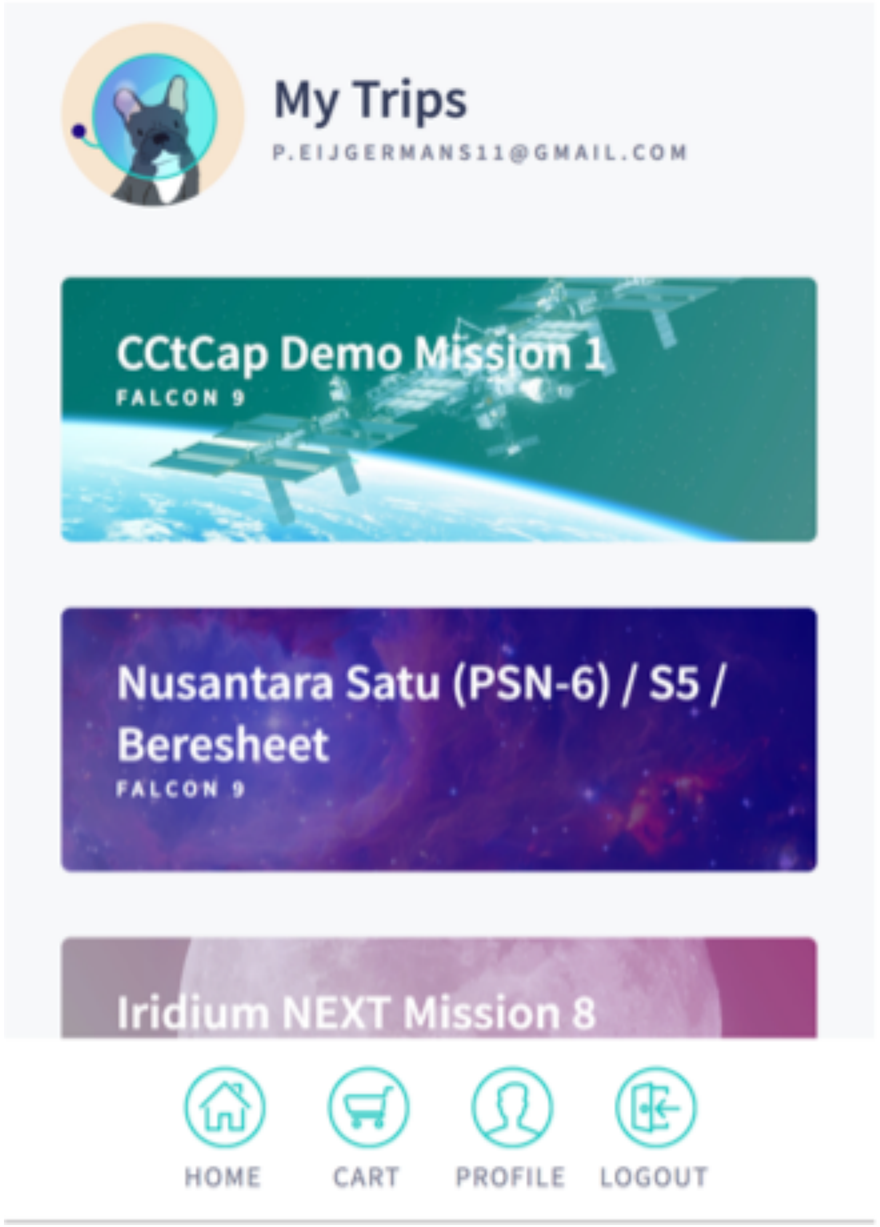
P.EIJGERMANS11@GMAIL.COM

CCtCap Demo Mission 1

FALCON 9

Nusantara Satu (PSN-6) / S5 / Beresheet

FALCON 9



My Trips

P.EIJGERMANS11@GMAIL.COM

CCtCap Demo Mission 1
FALCON 9

Nusantara Satu (PSN-6) / S5 / Beresheet
FALCON 9

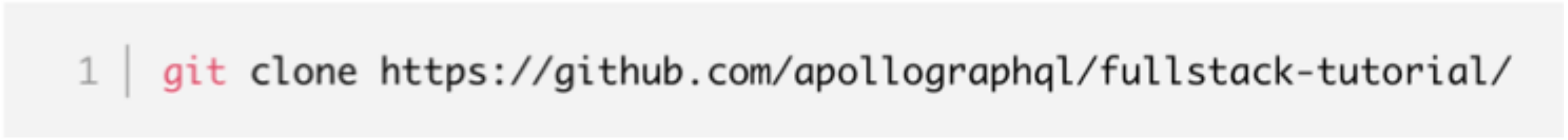
Iridium NEXT Mission 8

Prerequisites

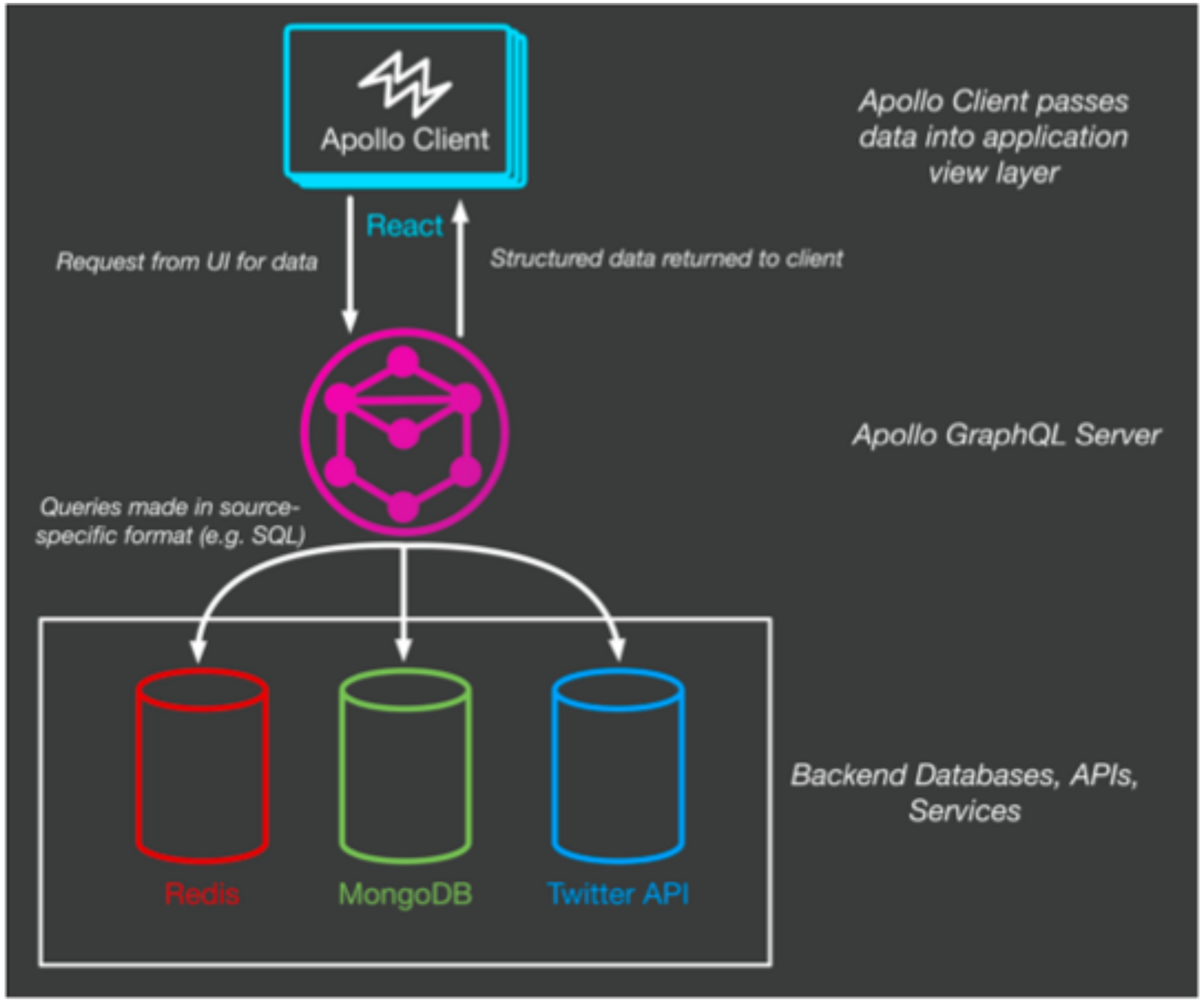
- **System requirements**
- Before we begin, make sure you have:
- Node.js v6.9.0 or greater
- npm 3.10.8 or greater
- git v2.14.1 or greater

- **Visual Studio Code** installed

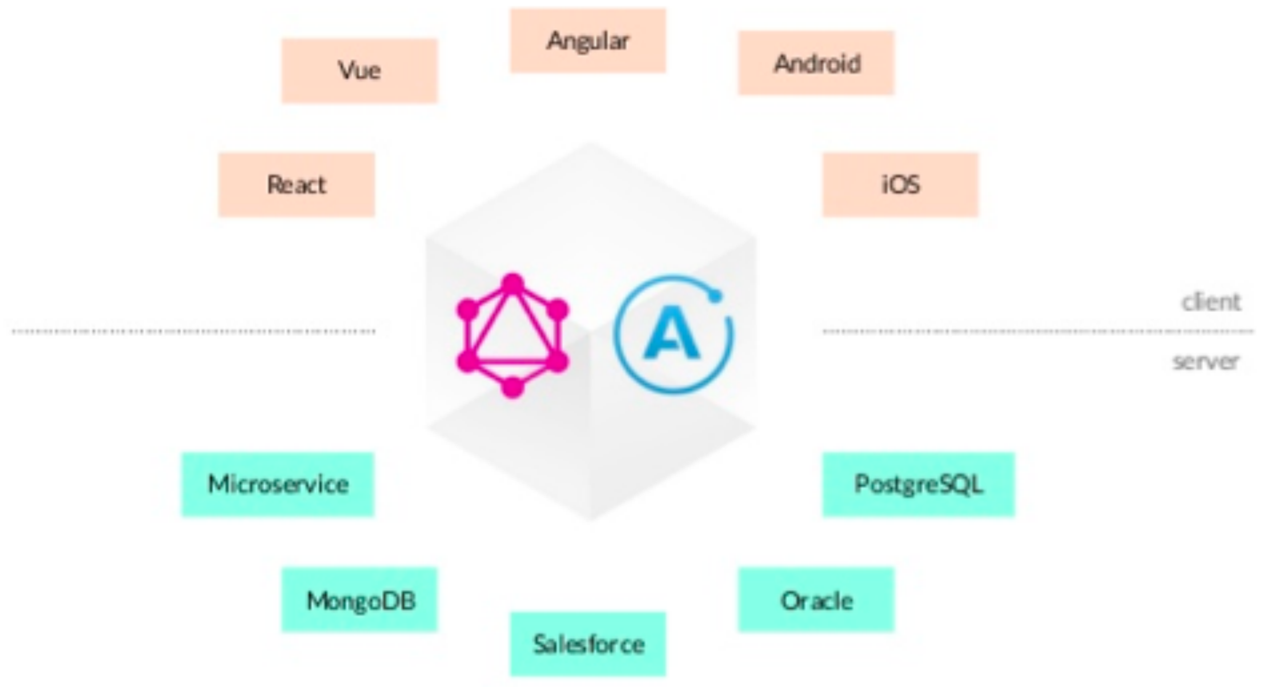
Next, in your terminal, clone this repository:



```
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

Where do GraphQL and Apollo live?



Apollo Client

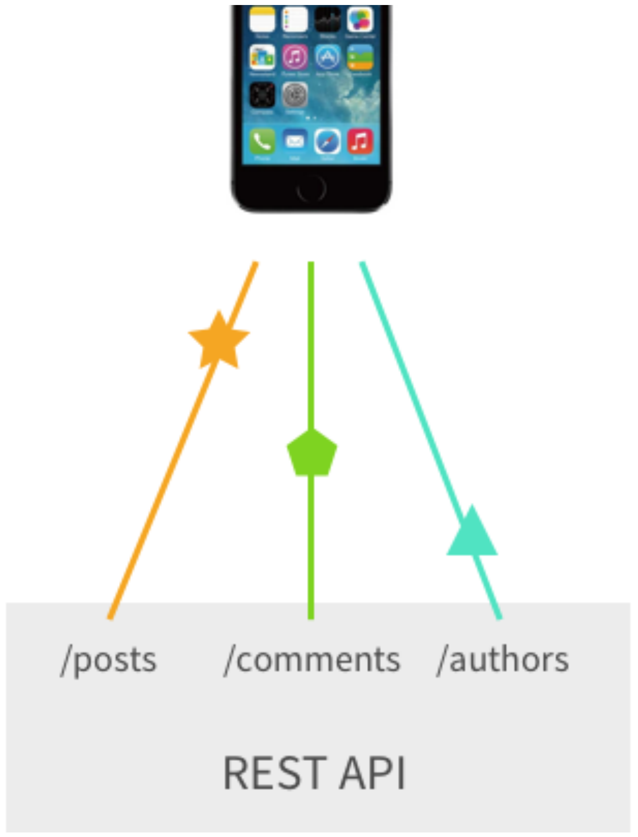
- Trigger **queries** as part of UI components;
- Binding query results to the UI
- Caching
- State management (**no Redux needed !**)
- Integrations for React, React Native, Vue, Angular ect.

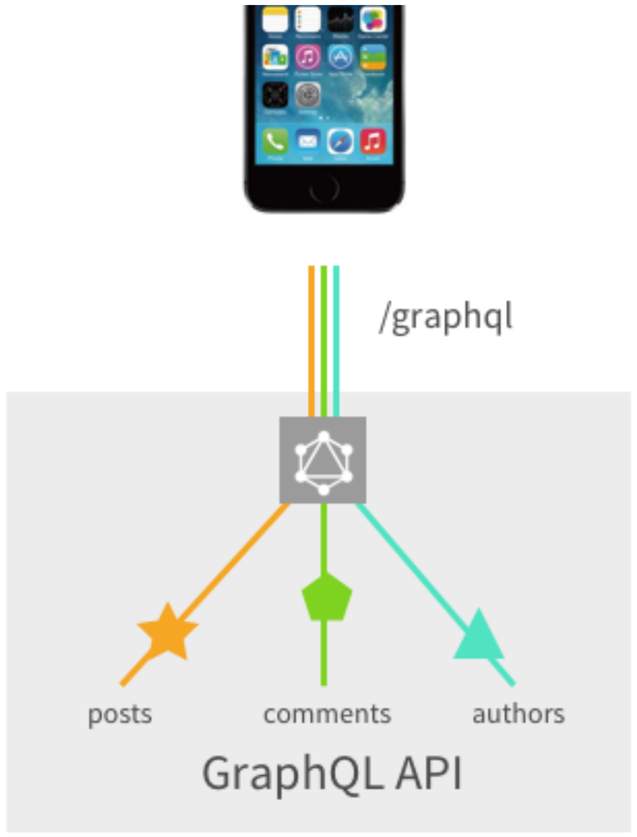
Apollo Server

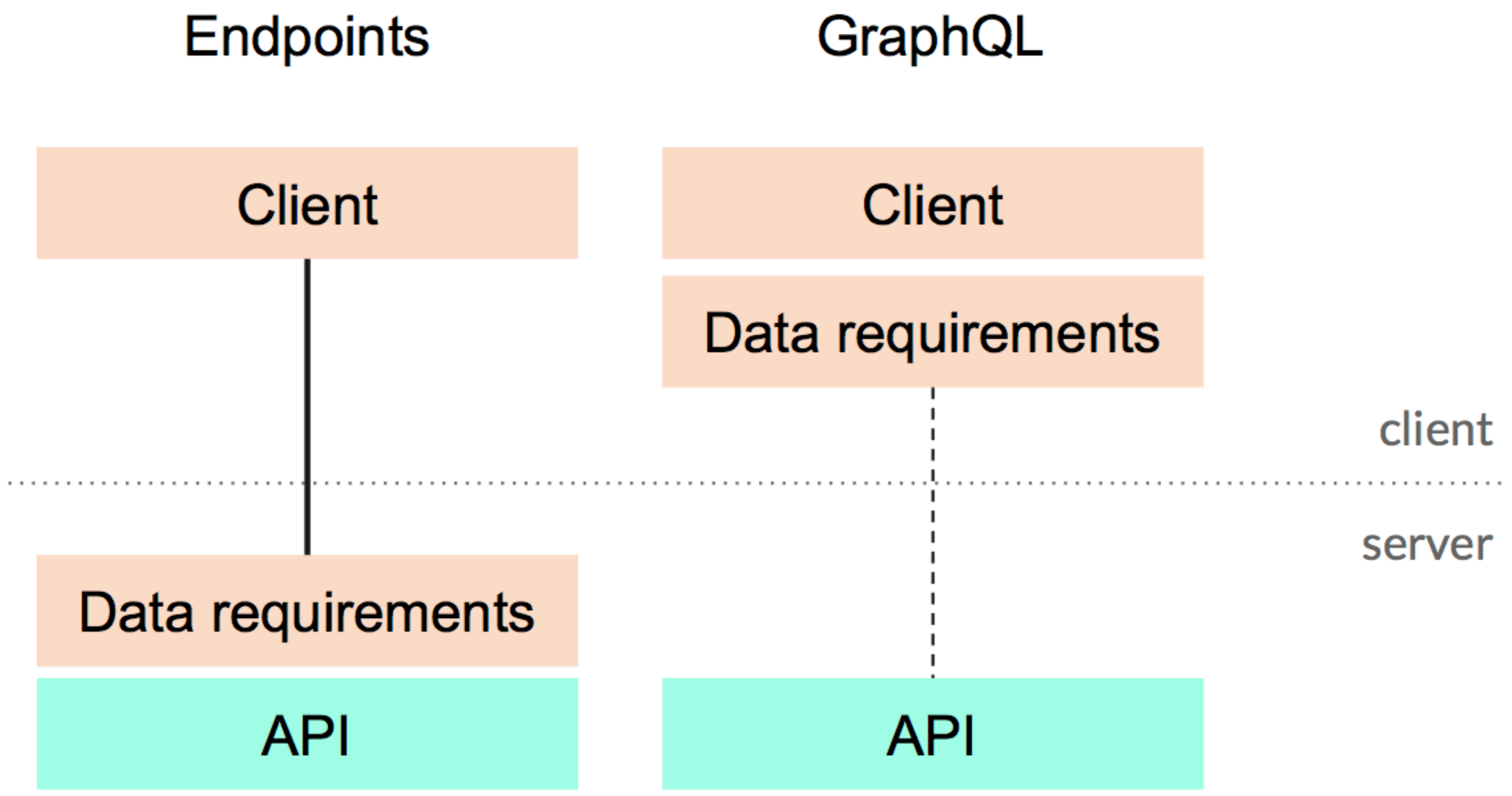
- Defining a ***generic*** **API** = *schema (model) for the client*
- API **layered over your existing services, including REST APIs and databases**
- Improved Performance → smaller payload
- Caching

Why GraphQL?

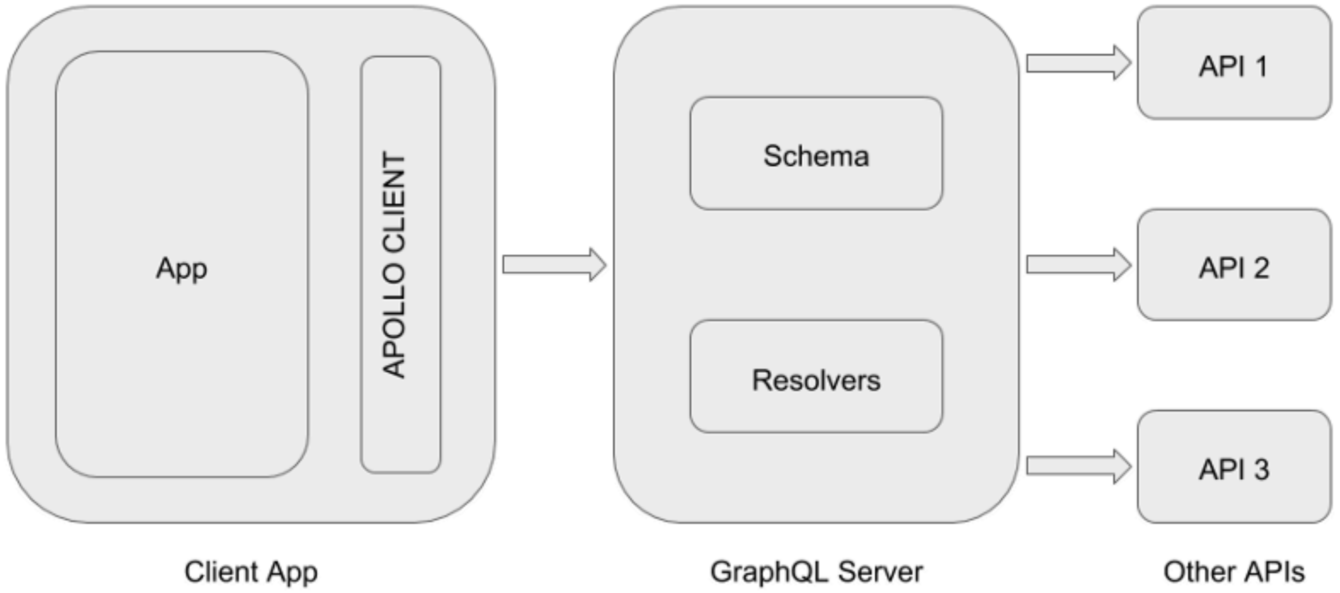
Get many resources in a single request






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

GraphQL traditional approach



Step 1: Build a schema = API

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

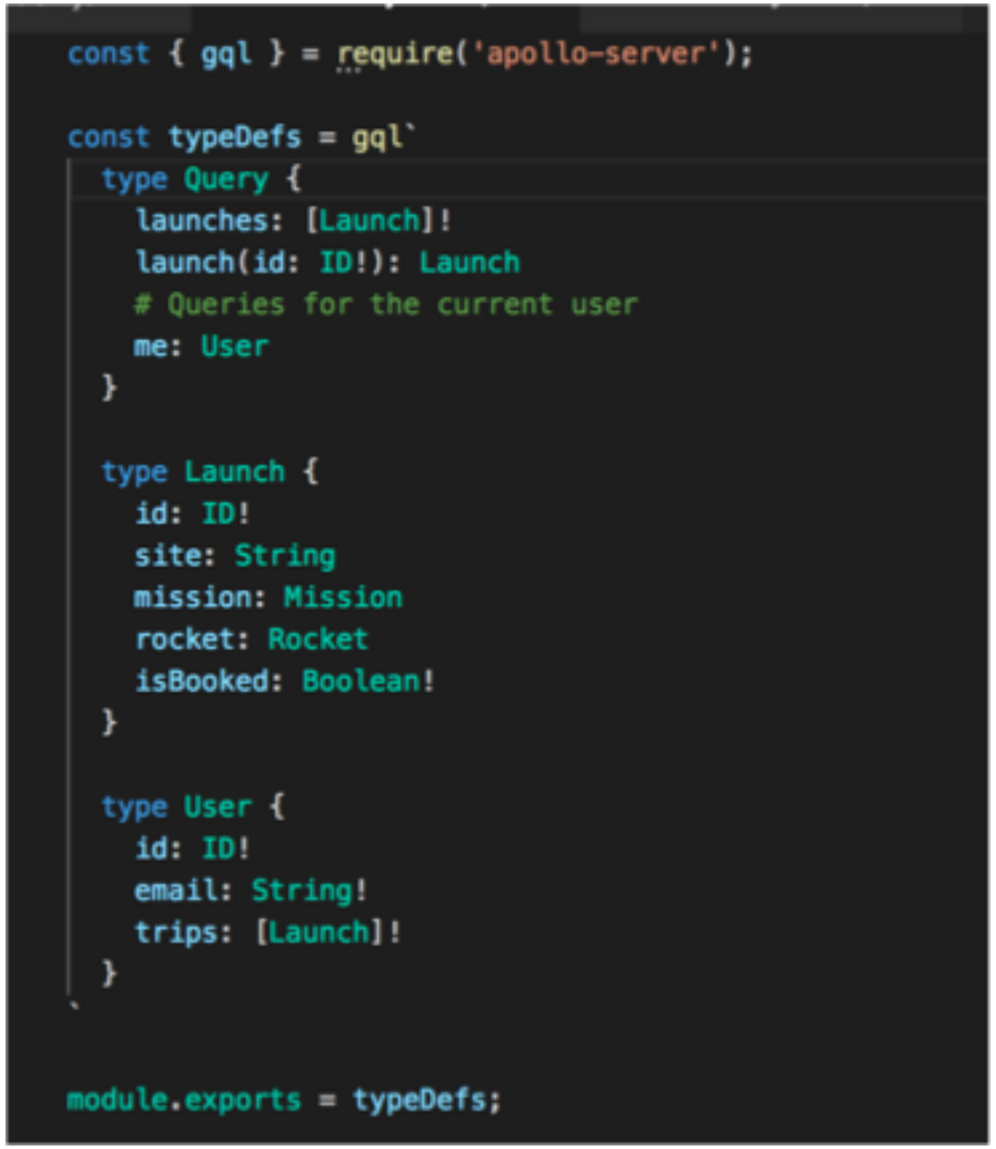
Mutations

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;
```

What’s awesome about GraphQL

The shape of your query will match the shape of your response.

Our schema will be based on these features:

Queries

Fetch all upcoming rocket launches

Fetch a specific launch by its ID

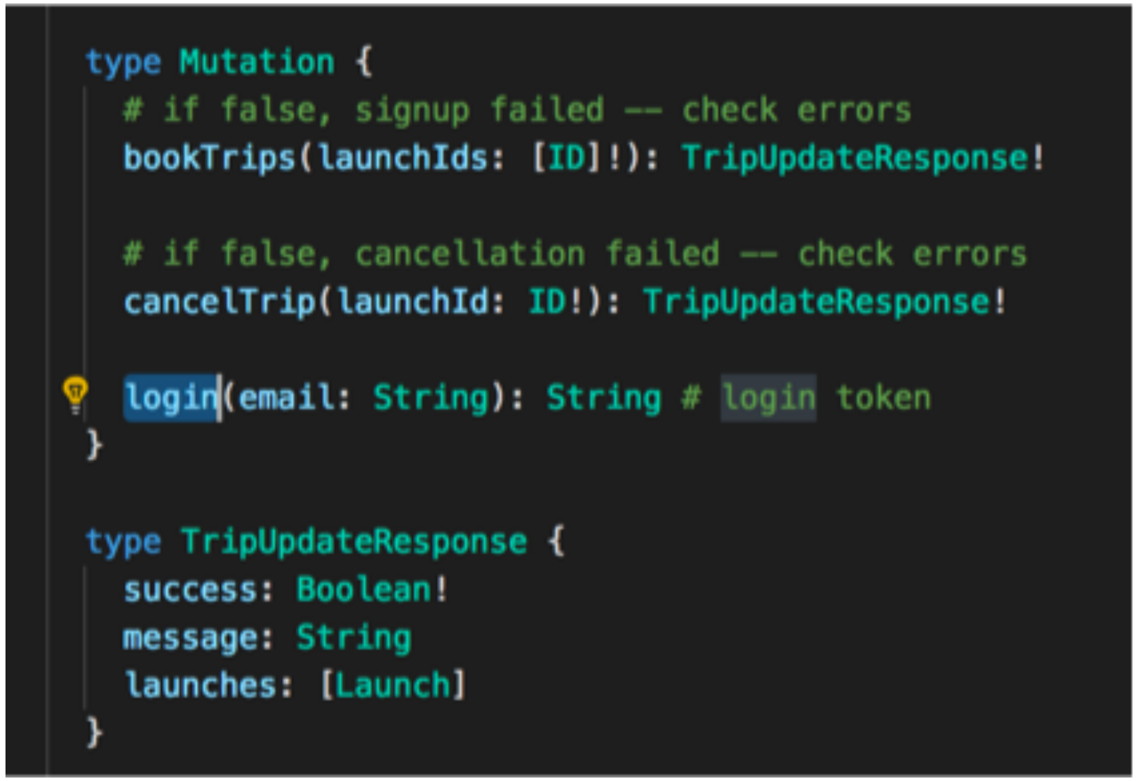
Mutations

Book launch trips if the user is logged in

Cancel launch trips if the user is logged in

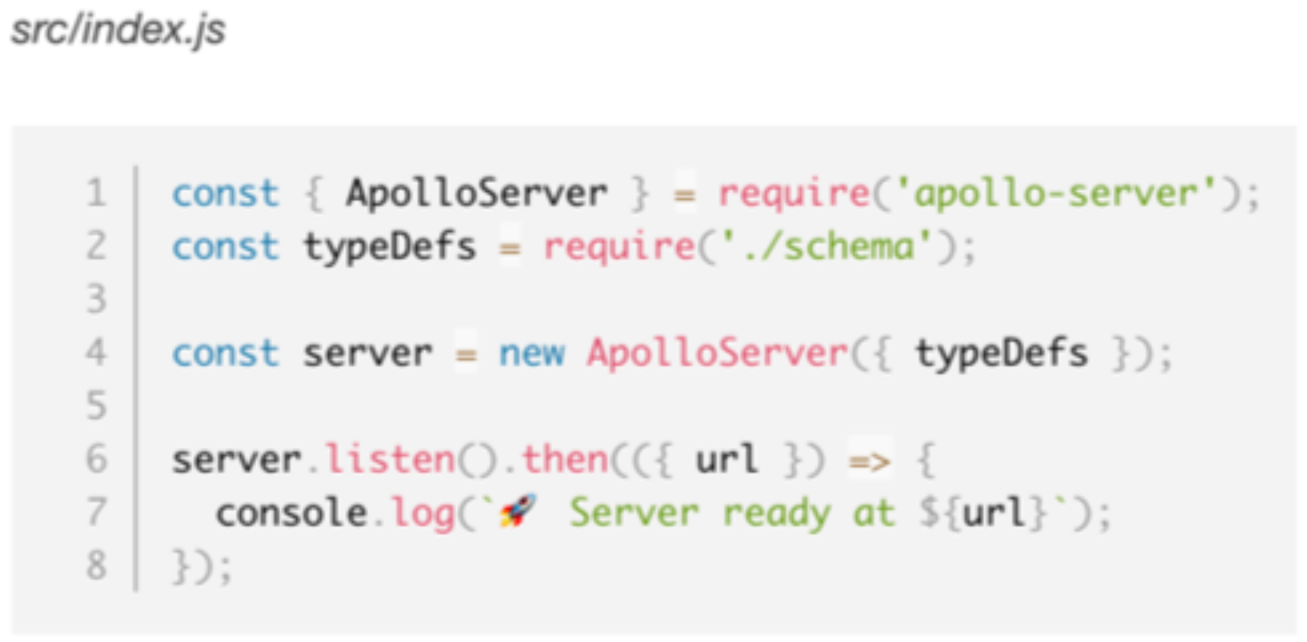
Login the user

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

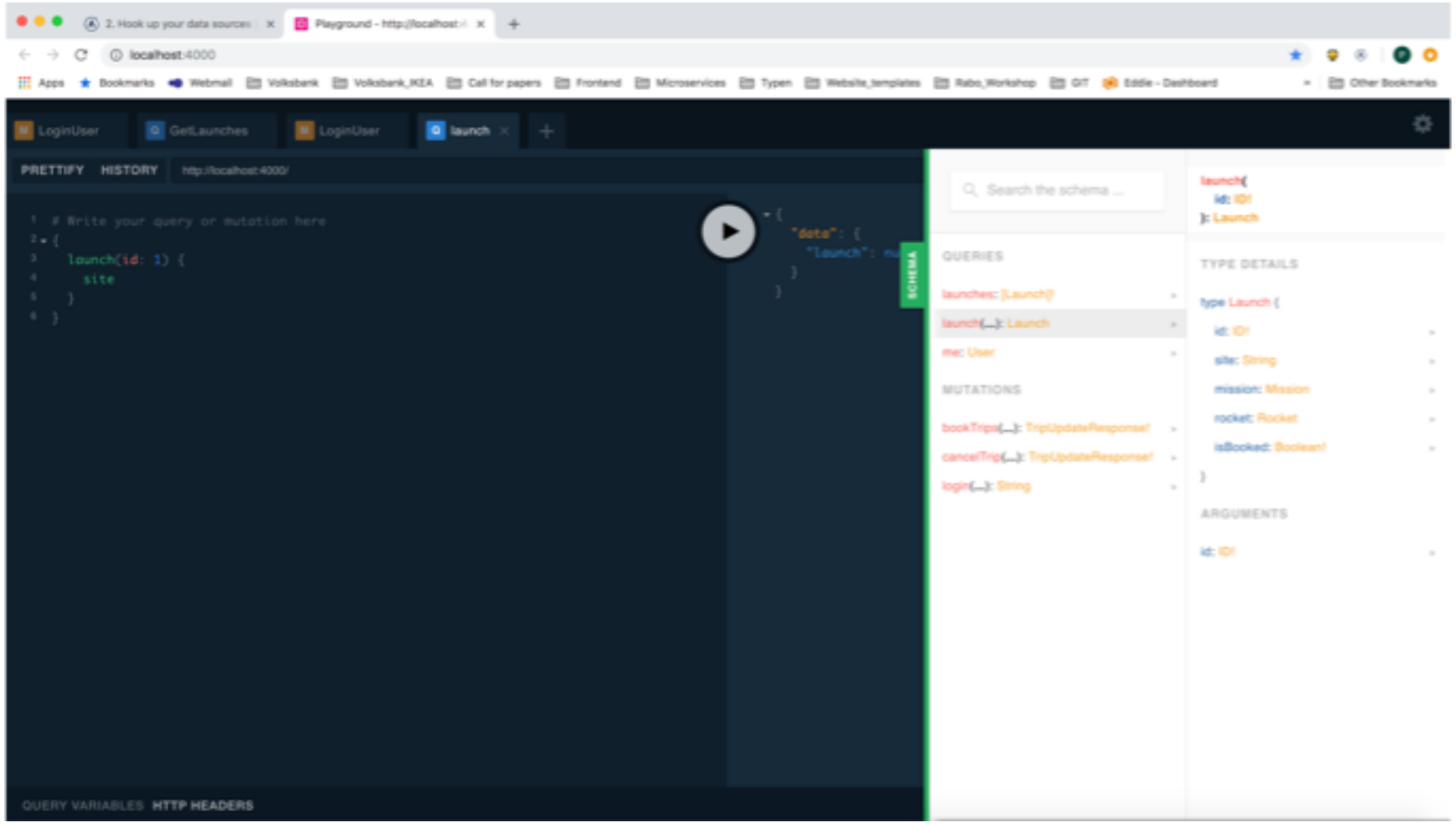


src/index.js

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

const server = new ApolloServer({ typeDefs });

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

Test your schema

in GraphQL Playground



```
# Write your query or mutation here
{
  launch(id: 1) {
    site
  }
}
```

```
{
  "data": {
    "launch": nu
  }
}
```

QUERIES

- launches: [Launch]!
- launch(...): Launch
- me: User

MUTATIONS

- bookTrips(...): TripUpdateResponse!
- cancelTrip(...): TripUpdateResponse!
- login(...): String

```
launch(
  id: ID!
): Launch
```

TYPE DETAILS

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

ARGUMENTS

id: ID!

Step 2: Hook up your data sources

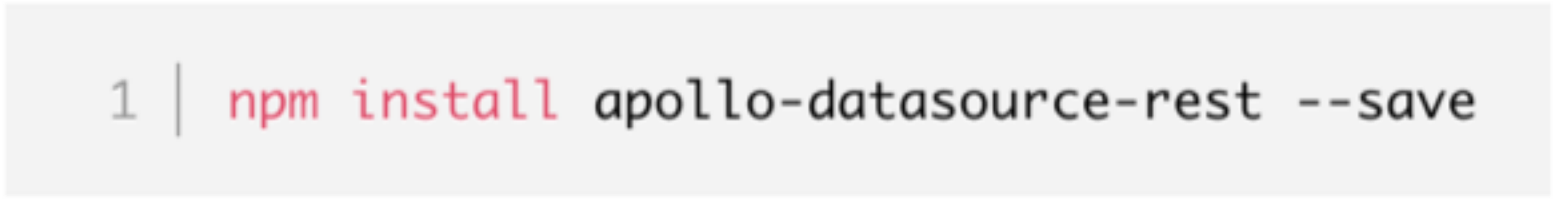
Connect REST data to your graph/schema

Connect a REST API Data Source

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

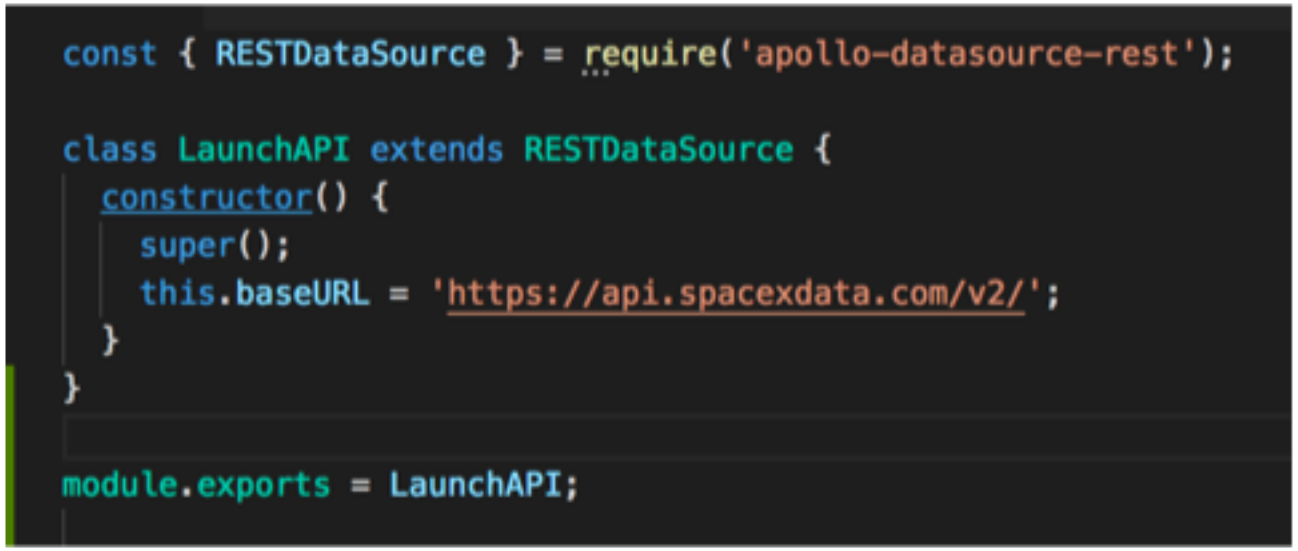
Connect a REST API Data Source

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



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

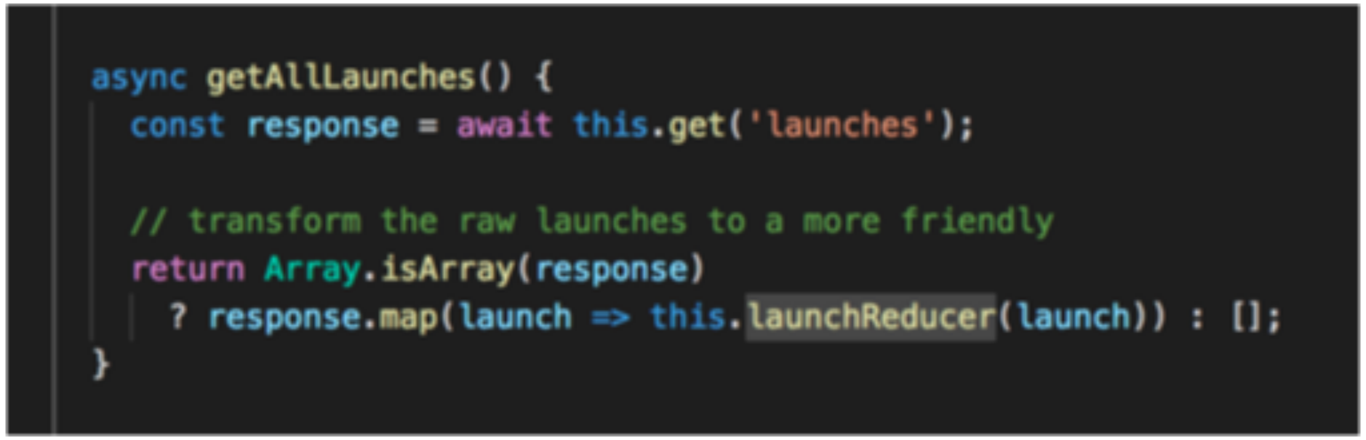
Define DataSource (*launch.js*)



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

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

module.exports = LaunchAPI;
```

launch.js

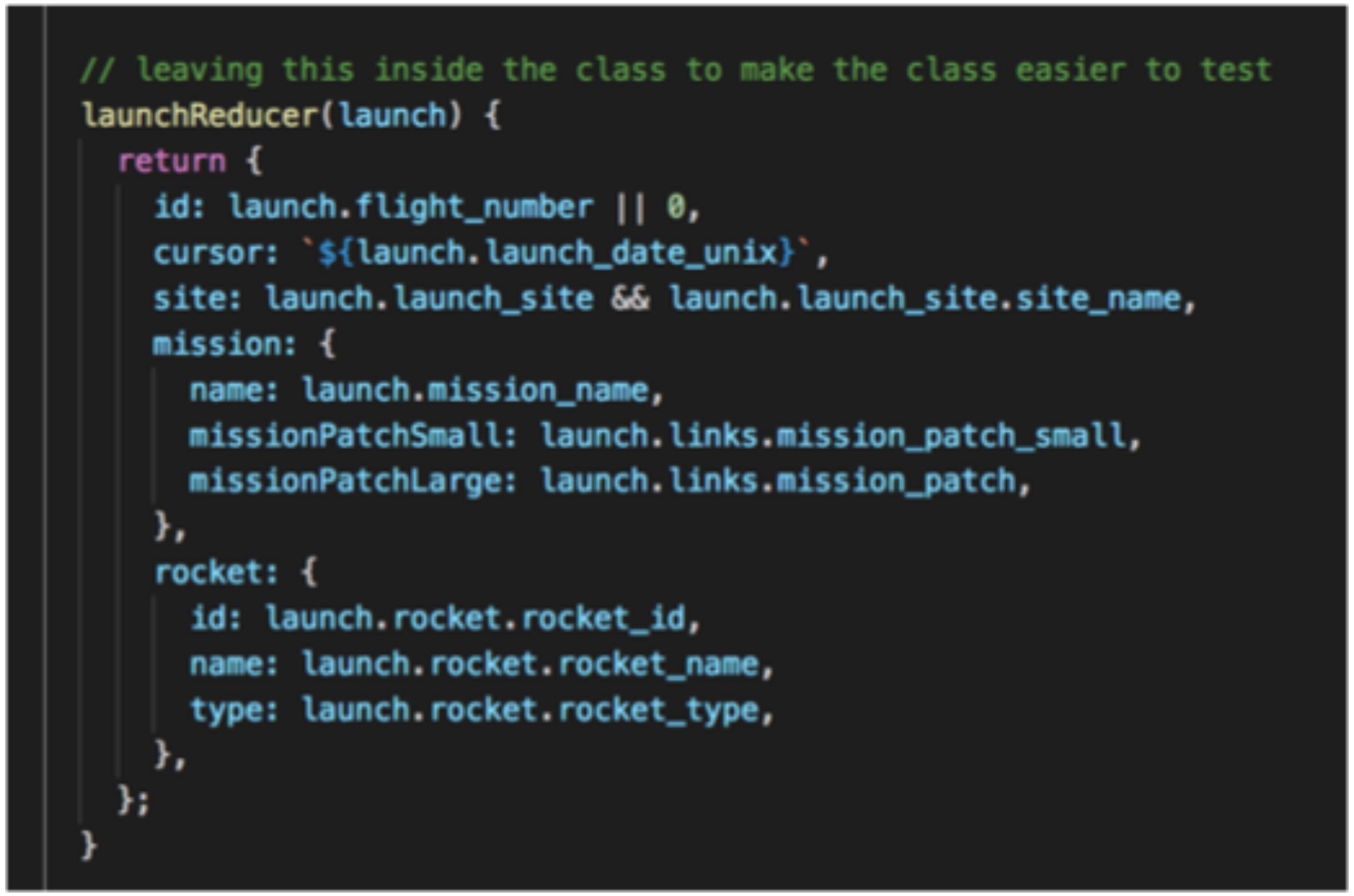
this = the REST data sources with baseURL



```
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)) : [];
}
```

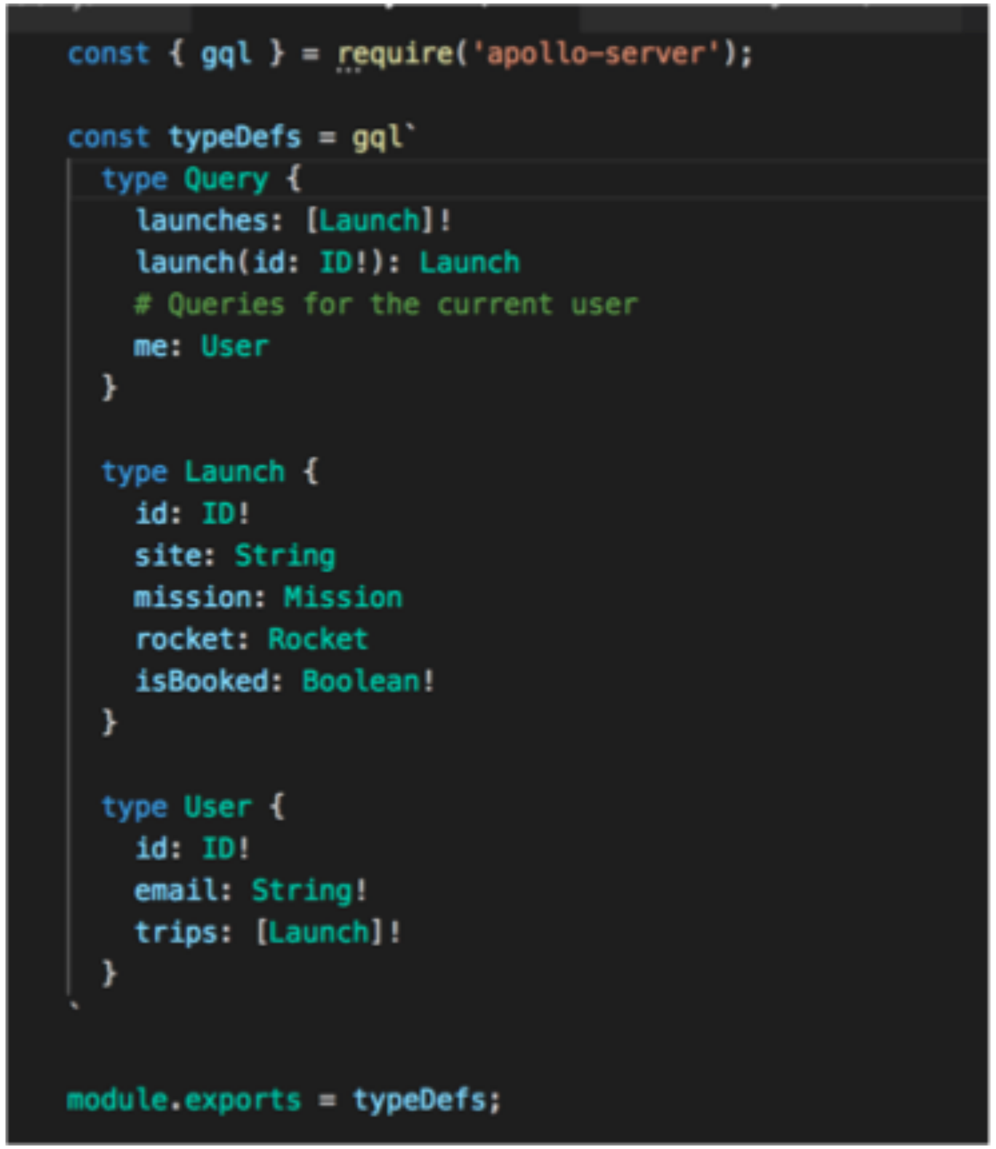
https://api.spacexdata.com/v2/launches

Add launchReducer in launch.js



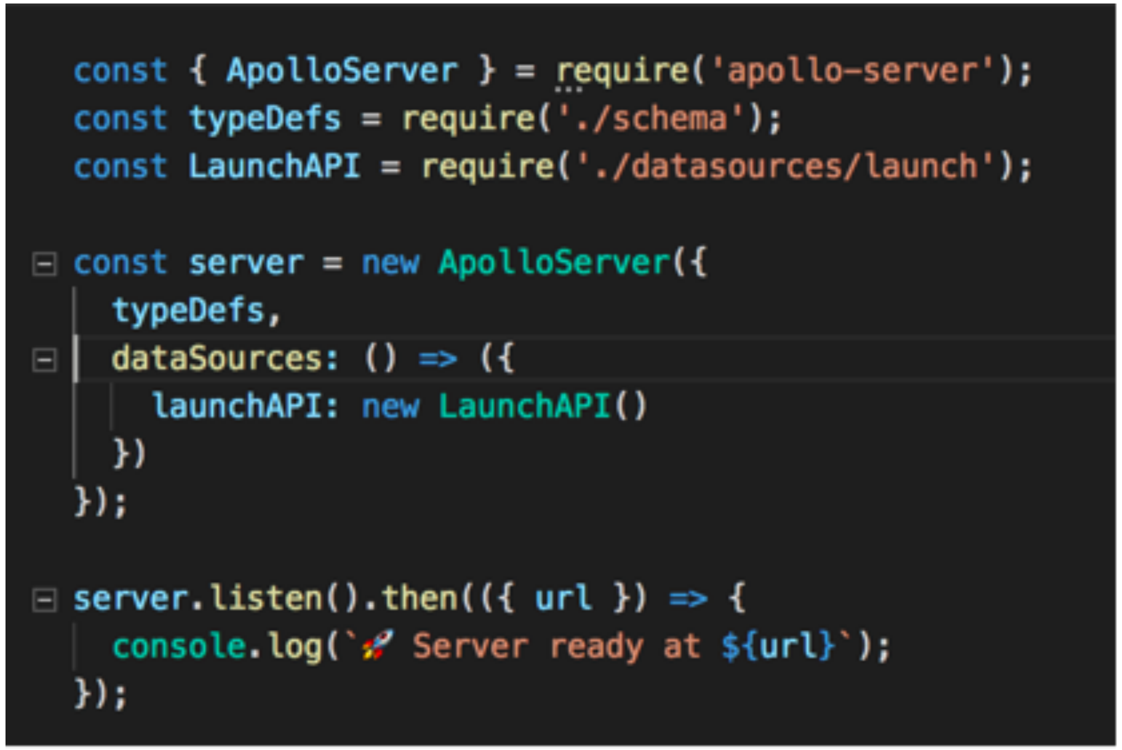
```javascript
// leaving this inside the class to make the class easier to test
launchReducer(launch) {
  return {
    id: launch.flight_number || 0,
    cursor: `${launch.launch_date_unix}`,
    site: launch.launch_site && launch.launch_site.site_name,
    mission: {
      name: launch.mission_name,
      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,
    },
  };
}
```

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 Datasource 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}`);
});
```

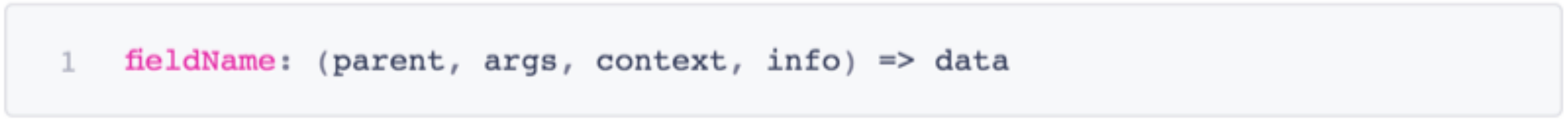
Step 3: Write your graph’s *Resolvers*

Learn how a GraphQL query fetches data

What is a Resolver?

Resolvers voor het daadwerkelijk ophalen van de service.

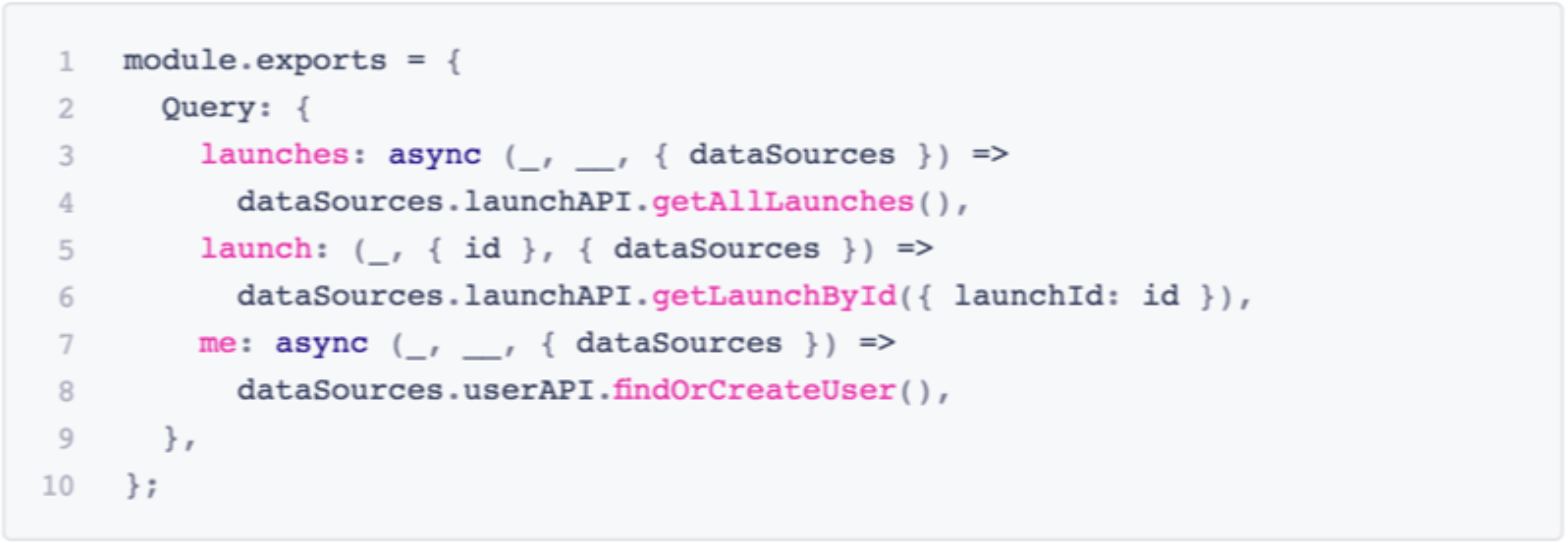
Syntax Resolver



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

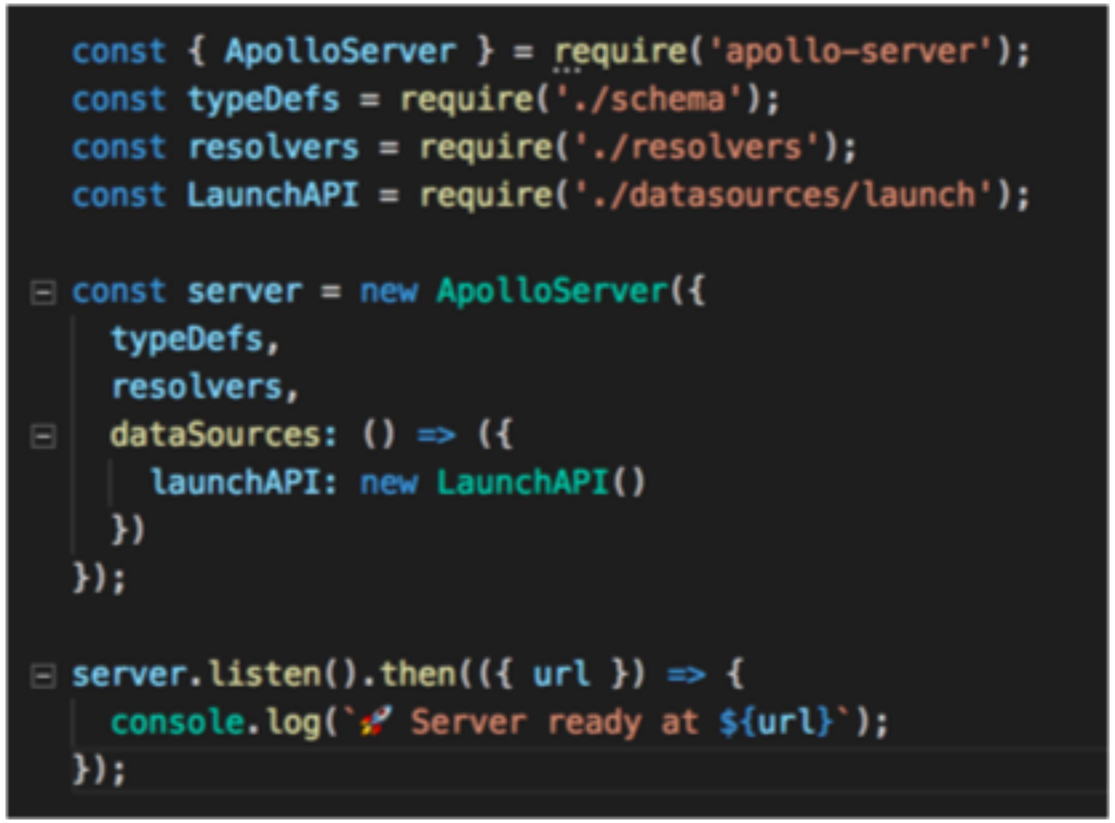
Write Query Resolvers (*resolvers.js*)

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(),
  },
};
```

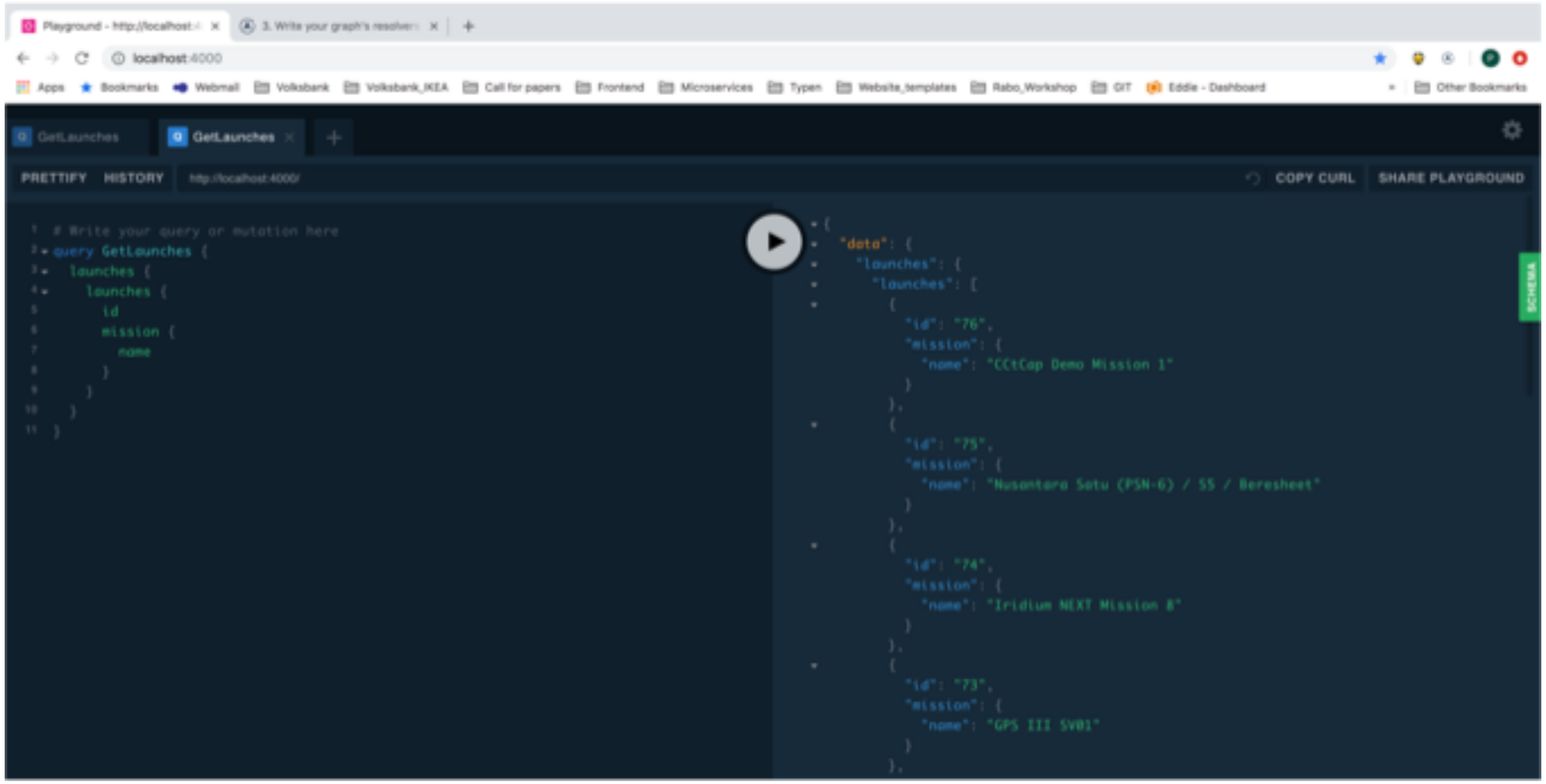
Connecting Resolvers to Apollo Server



```
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}`);
});
```

Run Queries

in GraphQL Playground



```
# Write your query or mutation here
query GetLaunches {
  launches {
    launches {
      id
      mission {
        name
      }
    }
  }
}
```

```
{
  "data": {
    "launches": {
      "launches": [
        {
          "id": "76",
          "mission": {
            "name": "CCtCap Demo Mission 1"
          }
        },
        {
          "id": "75",
          "mission": {
            "name": "Nusantara Satu (PSN-6) / S5 / Beresheet"
          }
        },
        {
          "id": "74",
          "mission": {
            "name": "Iridium NEXT Mission 8"
          }
        },
        {
          "id": "73",
          "mission": {
            "name": "GPS III SV01"
          }
        },
```

Final flow

Resolver → Data Source → Reducer → Schema

→ FE

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.