# **GRAPHQL API’daki typelari TYPESCRIPTTEKI TYPElara donusturmek**

Bu sayede biz Typescript icin extradan fazla interface gibi kod yazmak zorunda kalmayacagiz.

Asagidaki kodlari kendimize gore duzenledikten sonra scripts kismina yaziyoruz.

Once Apollo global olarak yuklu degilse npm install -g apollo diyerek yukluyoruz.

client:download-schema => ile schemalari indirmesini istiyoruz.

--endpoint=http://localhost:9000/api", => daha sornasinda yazdigimzi burasi ilede GraphQL serverimizin adresini yaziyoruzki verileri cekebilsin.

"codegen:schema": "npx apollo client:download-schema --endpoint=http://localhost:9000/api",

Ve terminalde npm run codegen:schema yazdiktan sonra root dizininde schema.json olarak GrapQL API mizdan gelen typelari ve schema semalarini gorecegiz.

apollo client:codegen => busefer apolloya clientimizin kodlarini generate etmesini yani donusturmesini soyluyoruz.

 --localSchemaFile=schema.json => Buradada yukarida olusturdugumuz schema.jsonu aliyoruz aslinda. Ana dizinde oldugu icin bu sekilde baska birsey yazmaya gerek yok

includes=src/\*\*/\*.tsx => Buradada ana dizndeki src icindeki tum klasorler ve tsx dosyalari ve tum klasorlerin icindeki tsx dosyalarina bakip gereken type yani interfaceleri bizim icin olusturmasini istiyoruz.

--globalTypesFile=./src/lib/graphql/globalTypes.ts => GlobalTypeslarin yani enum gibi degerlkerin cikacagi yeri belirler.

--target=typescript => Buradada olusturulacak olan typelarin bizim icin typescript icin gerekli oldugu icin Typescript icin olusturulmasini istiyoruz.

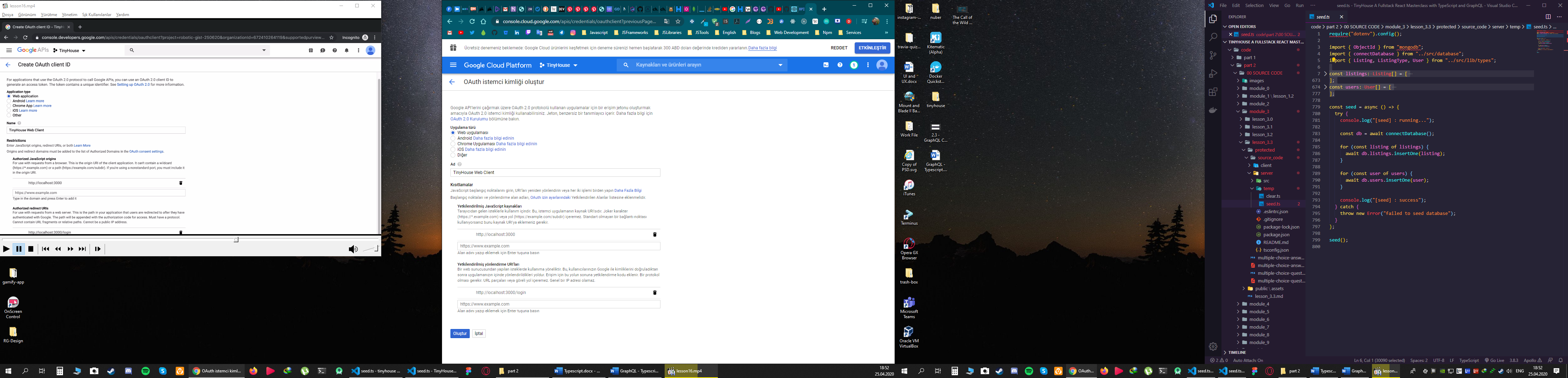
"codegen:generate": "npx apollo client:codegen --localSchemaFile=schema.json --includes=src/\*\*/\*.tsx --globalTypesFile=./src/lib/graphql/globalTypes.ts --target=typescript"

GOOGLE AUTHENTICATION

PROJE OLUSTUR

KIMLIK BILGILERI kismindan KIMLIK BILGISI OLUSTUR

Ve daha sornada OAuth istemci kimliği veya Oauth Client ID yapiyoruz.



Ve sonra yukaridaki gibi yapiyoruz ayarlari.

Sunlara dikkat et yukaridaki ve kendine gore ayarla => <http://localhost:3000> ve <http://localhost:3000/login>

Ve daha sonra aldigimiz Client(Istemci) ID ve secret keyimizi dot.env ye asagidaki gibi ekliyoruz.

G\_CLIENT\_ID=344827543625-c1b101v91t8v8lf8smp9l2t2fg1jh8ed.apps.googleusercontent.com

G\_CLIENT\_SECRET=oIJajnLPJhJGKSbKh2dYg71F

Tabi url mizide ekleyelim projemizin

PUBLIC\_URL=http://localhost:3000

Daha sonra typeDefs ve resolverslarimizi asagidaki gibi ayarliyoruz.

###typeDefs

import { gql } from 'apollo-server-express';

export const typeDefs = gql`

  type Query {

    authUrl: String!

  }

  type Mutation {

    logIn: String!

    logOut: String!

  }

`;

###Resolvers

import { IResolvers } from 'apollo-server-express';

export const viewerResolvers: IResolvers = {

  Query: {

    authUrl: () => {

      return "Query.authUrl"

    }

  },

  Mutation: {

    logIn: () => {

      return "Mutation logIn"

    },

    logOut: () => {

      return "Mutation logOut"

    },

  }

}

npm install googleapis

Daha sorna yukaridaki gibi googleapis i yukluyoruz.

Daha sonra api.ts veya Google.ts veya auth.ts adinda bir dosya olusturup asagidaki gibi ayarliyoruz.

import { google } from 'googleapis';

const auth = new google.auth.OAuth2(

  process.env.G\_CLIENT\_ID,

  process.env.G\_CLIENT\_SECRET,

  `${process.env.PUBLIC\_URL}/login`

)

*//access\_type: 'online', => Buda default olarak online yazilir*

*//SCOPELAR icin detayla liste icin bu linke gidin => https://developers.google.com/identity/protocols/oauth2/scopes*

*//Yani asagida bizOAuth2 ile erisebilecegimiz scopelardan sectigimiz email ve profile i istiyoruz.*

export const Google = {

  authUrl: auth.generateAuthUrl({

*// eslint-disable-next-line @typescripot-exlint/camelcase*

    access\_type: 'online',

    scope: [

      "https://www.googleapis.com/auth/userinfo.email ",

      "https://www.googleapis.com/auth/userinfo.profile "

    ]

  }),

  logIn: async (*code*: string) => {

*//Bu sekildede  access tokeni ve refesh tokenin ikisinide aliriz.*

    const { tokens } = await auth.getToken(code);

*//Ve credential olarak ayarliyoruz.*

    auth.setCredentials(tokens);

*//Peopleun ustune gelindiginde current versionu gorebilirsiniz.*

*//Bu sekildede authenticated olan kisinin bilgilerini aliyoruz.*

    const {data} = await google.people({version: "v1"}).people.get({

      resourceName: 'people/me',

      personFields: 'emailAddresses,names,photos'

    });

    return { user: data };

  }

}

Ve google hizmetlerinden People Api i etkinlestiriyoruz.

COOKIE PARSER KULLANIMI

Npm install cookie-parser

Npm install @types/cookie-parser

.env dosyasina ekle

SECRET=this-is-a-secret

Sonra serverin kurulu oldugu index.js e gelip

import cookieParser from 'cookie-parser';

daha sorna da server fonksiyonunun icine secret keyimizi .envden alarak eyaziyoruz.

app.use(cookieParser(process.env.SECRET));

daha sorna graphql ile asagidaki gibi request ve response objemizi context ile gondeririz

context: ({ *req*, *res* }) => ({

      db,

      req,

      res

    })

Daha sonra .env dosyasina asagidaki satiri ekliyoruz.

NODE\_ENV=development

Ve cookie parseri kullanacagimiz yerde asagidaki satiri ekliyoruz

const cookieOptions = {

  httpOnly: true,

  sameSite: true,

  signed: true,

  secure: process.env.NODE\_UNV === 'development' ? false : true

}

Kullanacagimiz yerde response ulasmak icin asagidaki gibi responsemizi ekliyoruz

const logInViaGoogle = async (

*…*

*res*: Response

): Promise<User | undefined> => { }

Dahasonra asagidaki gibi cookiemizin omrunu ayarliyoruz.

  res.cookie("viewer", userId, {

    ...cookieOptions,

    maxAge: 365 \* 24 \* 60 \* 60 \* 1000

  });

Cookiyi temizlemek icinde logouttan sorna asagidaki kodu yaziyoruz. Ilki yukarida tanimladigimiz cookie adi 2. Si ozellikleri

res.clearCookie("viewer", cookieOptions);

daha sonra login dosyamizda asagidaki kodu olusturuyopruz

const logInViaCookie = async (*token*: string, *db*: Database, *req*: Request, *res*:Response): Promise<User | undefined> => {

  const updateRes = await db.users.findOneAndUpdate(

    { \_id: req.signedCookies.viewer },

    {

      $set: { token }

    },

    { returnOriginal: false }

  );

  let viewer = updateRes.value;

  if(!viewer) {

    res.clearCookie("viewer", cookieOptions);

  }

  return viewer;

}

Daha sorna login fonksiyonunu asagidaki sekilde yapiyoruz

    logIn: async (

*\_root*: undefined,

      { *input* }: LogInArgs,

      { *db*, *req*, *res* } : { db: Database, req: Request, res: Response }

    ): Promise<Viewer> => {

      try {

        const code = input ? input.code : null;

        const token = crypto.randomBytes(16).toString("hex");

        const viewer: User | undefined = code

          ? await logInViaGoogle(code, token, db, res)

          : await logInViaCookie(token, db, req, res);

        if (!viewer) {

          return { didRequest: true };

        }

        return {

          \_id: viewer.\_id,

          token: viewer.token,

          avatar: viewer.avatar,

          walletId: viewer.walletId,

          didRequest: true

        };

      } catch (error) {

        throw new Error(`Failed to log in: ${error}`);

      }

    },