**GRAPHQL CODE for CRUD**

**Inside index.js**

import {GraphQLServer, PubSub} from 'graphql-yoga'

import db from './db'

import Mutation from './resolvers/Mutation'

import Query from "./resolvers/Query"

import User from "./resolvers/User"

import Prod from './resolvers/Prod'

import Comment from './resolvers/Comment'

import Review from './resolvers/Review'

import Subscription from './resolvers/Subscription'

const pubsub = new PubSub()

const server = new GraphQLServer({

    typeDefs:'./src/schema.graphql',

    resolvers: {

        Query,

        Mutation,

        User,

        Prod,

        Review,

        Comment,

        Subscription

    },

    context : {

        db,

        pubsub

    }

})

server.start(()=>console.log('The server has started !'))

**Inside db.js**

const prodArray = [

    {

        prodID: '1',

        name: 'Lassi',

        price: 24,

        isAvailable: false,

        stockUnits:0,

    },

………

]

let reviewsArray = [

    {

        title:'Nice Lassi',

        body:'Perfect consistency and balance of sweetness and creaminess',

        id:'1',

        upvotes:5,

        parentProdOfRev:'1',

        isPublished:false,

        parentUserOfRev:1

    },

………..

]

let commentsArray = [

    {

        title:'The samosa is indeed that tasty',

        body:"I too had the pleasure of trying it last week, and man........it was the best samosa I've ever had",

        commId :"1",

        parentRevOfComm:'7',

        parentProdOfComm:'4',

        parentUserOfComm:5

    },

…….

]

let usersArray = [

    {

        name:'Ankit',

        id:"1",

        email:'Ankit@graphql.com',

        age:19,

        favouriteProd:1

},

………

]

const db = {

    usersArray,

    prodArray,

    reviewsArray,

    commentsArray

}

export {db as default}

**Inside schema.graphql**

type Query {

    greeting(name: String, position: String): String!

    prodID: ID!

    name: String!

    price: Float!

    isAvailable: Boolean!

    stockUnits: Int!

    availableDishes(specific: String): [Prod!]!

    review: Review!

    reviews(specific: String): [Review]!

    comment: Comment

    comments(specific: String): [Comment]

    bill(prices: [Float]): Float!

    users(name: String, email: String, age: Int): [User]!

}

type User {

    id: ID!

    name: String!

    age: Int

    email: String!

    posts: [Review]

    comments: [Comment]

    favouriteProd: Prod

}

type Review {

    title: String!

    body: String!

    id: ID!

    upvotes: Int

    parentProdOfRev: Prod!

    commentsChildOfRev: [Comment]

    parentUserOfRev: User!

    isPublished: Boolean!

}

type Prod {

    prodID: ID!

    name: String!

    price: Float!

    isAvailable: Boolean!

    stockUnits: Int!

    prodReviews: [Review]

    prodComments: [Comment]

    frequentUsers: [User]

}

type Comment {

    title: String!

    body: String!

    commId: ID!

    parentRevOfComm: Review!

    parentProdOfComm: Prod!

    parentUserOfComm: User!

}

type Mutation {

    createUser(data: CreateUserInput): User

    createReview(data: CreateReviewInput): Review

    createComment(data: CreateCommentInput): Comment

    deleteUser(id: ID!): User!

    deleteReview(id: ID!): Review!

    deleteComment(id: ID!): Comment!

    deleteProduct(id: ID!): Prod!

    updateUser(id: ID!, data: UpdateUserInput): User!

    updateReview(id: ID!, data: UpdateReviewInput): Review!

    updateComment(id: ID!, data: UpdateCommentInput): Comment!

    updateProd(id: ID!, data: UpdateProdInput): Prod!

}

input UpdateProdInput {

    name: String!

    isAvailable: Boolean!

    price: Float

    stockUnits: Int

}

input UpdateReviewInput {

    title: String

    body: String

    upvotes: Int

    isPublished: Boolean!

    parentProdOfRev: ID

    parentUserOfRev: ID

}

input UpdateCommentInput {

    title: String

    body: String

    parentRevOfComm: ID

    parentProdOfComm: ID

    parentUserOfComm: ID

}

input UpdateUserInput {

    name: String

    age: Int

    email: String

}

input CreateUserInput {

    name: String!

    email: String!

    age: Int

}

input CreateReviewInput {

    title: String!

    body: String!

    upvotes: Int

    isPublished: Boolean!

    parentProdOfRev: ID!

    parentUserOfRev: ID!

}

input CreateCommentInput {

    title: String!

    body: String!

    parentProdOfComm: ID!

    parentRevOfComm: ID!

    parentUserOfComm: ID!

}

type Subscription {

    count: Int!

    comment(parentRevOfComm: ID!): CommentSubscriptionPayload!

    review: ReviewSubscriptionPayload!

}

enum MutationType {

    CREATED

    UPDATED

    DELETED

}

type ReviewSubscriptionPayload {

    mutation: MutationType!

    data: Review

}

type CommentSubscriptionPayload {

    mutation: MutationType!

    data: Comment

}

**Inside resolvers/Query.js**

const Query = {

    greeting(parent, args, { db }, info){

        if(args.name && args.position){

            return `Hello ${args.name} ! You are my favourite ${args.position} `

        } else if(args.name){

            return `Hello ${args.name} !`

        }else {

            return 'Hello !'

        }

    },

    bill(parent,args,{ db },info){

        if(args.prices.length == 0){

            return 0

        }

        return args.prices.reduce((accumulator,currentValue) => accumulator + currentValue)

    },

    prodID(){

        // return 12345

        return '4'

    },

    name(){

        return 'Aloo samosa'

    },

    price(){

        return 12.34

    },

    isAvailable(){

        return true

    },

    stockUnits(){

        return 123

    },

    review(){

        return({

            title:'Sabse best samosa',

            body:'Kya masala tha bc',

            id:'7',

            upvotes:2

        })

    },

    reviews(parent, args, { db }, info){

        if(args.specific){

            return db.reviewsArray.filter(rev => rev.title.toLowerCase().includes(args.specific.toLowerCase()))

        }

        return db.reviewsArray

    },

    availableDishes(parent, args, { db }, info){

        if(args.specific){

            return db.prodArray.filter(prod => prod.name.toLowerCase().includes(args.specific.toLowerCase()))

        }

        else{

            return db.prodArray

        }

    },

    comments(parent, args, { db },info){

        if(!args.specific){

            return db.commentsArray

        }

        else{

            return db.commentsArray.filter(comment => comment.title.toLowerCase().includes(args.specific.toLowerCase()) || comment.body.toLowerCase().includes(args.specific.toLowerCase))

        }

    },

    users(parent,args,{ db },info){

        if(args.name){

            return db.usersArray.filter(user => user.name == args.name)

        }else if(args.email){

            return db.usersArray.filter(user => user.email == args.email)

        }else{

            return db.usersArray

        }

    }

}

export {Query as default}

**Inside resolvers/User.js**

const User = {

    posts(parent,args,{ db },info){

        return db.reviewsArray.filter(rev => rev.parentUserOfRev == parent.id )

    },

    comments(parent,args,{ db },info){

        return db.commentsArray.filter(comm => comm.parentUserOfComm == parent.id)

    },

    favouriteProd(parent,args,{ db },info){

        return db.prodArray.find(prod => prod.prodID == parent.favouriteProd)

    }

}

export {User as default}

**Inside resolvers/Review.js**

const Review = {

    parentProdOfRev(parent,args, { db }, info){

        return db.prodArray.find(prod => prod.prodID == parent.parentProdOfRev)

    },

    commentsChildOfRev(parent,args,{ db },info){

        return db.commentsArray.filter(comm => comm.parentRevOfComm == parent.id)

    },

    parentUserOfRev(parent,args,{ db },info){

        return db.usersArray.find(user => user.id == parent.parentUserOfRev)

    }

}

export {Review as default}

**Inside resolvers/Comment.js**

const Comment = {

    parentRevOfComm(parent,args,{ db },info){

        return db.reviewsArray.find(review => review.id == parent.parentRevOfComm)

    },

    parentProdOfComm(parent,args,{ db },info){

        return db.prodArray.find(prod => prod.prodID == parent.parentProdOfComm)

    },

    parentUserOfComm(parent,args,{ db },info){

        return db.usersArray.find(user => user.id == parent.parentUserOfComm)

    }

}

export {Comment as default}

**Inside resolvers/Prod.js**

const Prod = {

    prodReviews(parent, args, { db },info){

        return db.reviewsArray.filter(rev=> rev.parentProdOfRev == parent.prodID)

    },

    prodComments(parent,args,{ db },info){

        return db.commentsArray.filter(comm => comm.parentProdOfComm == parent.prodID)

    },

    frequentUsers(parent,args,{ db },info){

        return db.usersArray.filter(user => user.favouriteProd == parent.prodID)

    }

}

export {Prod as default}

**Inside resolvers/Mutation.js**

import { v4 as uuidv4 } from 'uuid';

const Mutation = {

    createUser(parent,args,{ db },info){

        const isEmailTaken = db.usersArray.find(user => user.email == args.data.email)

        if(isEmailTaken){

            throw new Error('Email le liya koi pehle se')

        }

        const user = {

            id:uuidv4(),

            ...args.data

        }

        db.usersArray.push(user)

        return user

    },

    createReview(parent,args,{ db,pubsub },info){

        const doesUserExist = db.usersArray.find(user => user.id == args.data.parentUserOfRev)

        const doesProdExist = db.prodArray.find(prod => prod.prodID == args.data.parentProdOfRev)

        if(!doesProdExist){

            throw new Error('Gaand se nikalu kya yeh product...heh ?')

        }

        if(!doesUserExist){

            throw new Error('Yeh user kaun hai bhai ?')

        }

        const review = {

            id:uuidv4(),

            commentsChildOfRev:[],

            ...args.data

        }

        db.reviewsArray.push(review)

        pubsub.publish("review",{

            review:{

                mutation:'CREATED',

                data:review

            }

        })

        return review

    },

    createComment(parent,args,{ db,pubsub },info){

        const doesUserExist = db.usersArray.find(user => user.id == args.data.parentUserOfComm)

        const doesProdExist = db.prodArray.find(prod => prod.prodID == args.data.parentProdOfComm)

        const doesReviewExist = db.reviewsArray.find(review => review.id == args.data.parentRevOfComm)

        if(!doesProdExist){

            throw new Error('Abbe yaar......ye konse product ki baat ho reli ?')

        }

        if(!doesUserExist){

            throw new Error('Yeh user kaun hai bhai ?')

        }

        if(!doesReviewExist){

            throw new Error('Aise naam ka koi review nahi meri jaan')

        }

        const comment = {

            commId:uuidv4(),

            ...args.data

        }

        db.commentsArray.push(comment)

        pubsub.publish("comment", {

            comment:{

                mutation:'CREATED',

                data:comment

        }})

        return comment

    },

    deleteUser(parent,args,{ db },info){

        const userIndex = db.usersArray.findIndex(user => user.id == args.id)

        if(userIndex === -1){

            throw new Error('Yeh waala user toh mereko pata hi nahi yaar')

        }

        const deletedUsers = db.usersArray.splice(userIndex,1)

        db.reviewsArray = db.reviewsArray.filter(review => {

            const match = review.parentUserOfRev == args.id

            if(match){

                db.commentsArray = db.commentsArray.filter(comm => comm.parentRevOfComm !== review.id)

            }

            // console.log(match);

            return !match

        })

        db.commentsArray = db.commentsArray.filter(comm => comm.parentUserOfComm !== args.id )

        return deletedUsers[0]

    },

    deleteReview(parent,args, { db,pubsub }, info){

        const revIndex = db.reviewsArray.findIndex(review => review.id == args.id)

        if(revIndex === -1){

            throw new Error('Kuch bhi hava hi naa lag rahi ke kis review ki baat ho rahi hai')

        }

        // As splice method will return an array even if we just splice 1 single elem

        const [deletedReview] = db.reviewsArray.splice(revIndex, 1)

        db.commentsArray = db.commentsArray.filter(comment => comment.parentRevOfComm !== args.id)

        pubsub.publish("review",{

            review:{

                mutation:'DELETED',

                data:deletedReview

            }

        })

        return deletedReview

    },

    deleteComment(parent,args,{ db,pubsub },info){

        const commentIndex = db.commentsArray.findIndex(comment => comment.commId == args.id)

        if(commentIndex === -1){

            throw new Error('Aisa sab commenting humare waha nahi karte')

        }

        const [deletedComment] = db.commentsArray.splice(commentIndex,1)

        pubsub.publish("comment",{

            comment:{

                mutation:'DELETED',

                data:deletedComment

            }

        })

        return deletedComment

    },

    deleteProduct(parent,args,{ db },info){

        const productIndex = db.prodArray.findIndex(prod => prod.prodID == args.id)

        if(!productIndex){

            throw new Error('Product does not exist')

        }

        const [deletedProduct] = db.prodArray.splice(productIndex,1)

        usersArray = usersArray.filter(user => user.favouriteProd !== args.id)

        commentsArray = commentsArray.filter(comm => comm.parentProdOfComm !== args.id)

        reviewsArray = reviewsArray.filter(rev => rev.parentProdOfRev !== args.id)

        return deletedProduct

    },

    updateUser(parent, args, {db}, info){

        const { data, id } = args

        const user = db.usersArray.find(user => user.id == id)

        if(!user){

            throw new Error('User not found')

        }

        if(typeof data.email == "string"){

            const isEmailTaken = db.usersArray.find(user => user.email == data.email)

            if(isEmailTaken){

                throw new Error("Email has already been taken")

            }

            user.email = data.email

        }

        if(typeof data.name == "string"){

            user.name = data.name

        }

        if(typeof data.age !== "undefined"){

            user.age = data.age

        }

        if(typeof data.age !== "undefined"){

            user.age = data.age

        }

        return user

    },

    updateComment(parent, args, {db, pubsub}, info){

        const {data, id} = args

        const comment = db.commentsArray.find(comm => comm.commId == id)

        const originalComment = {...comment}

        if(!comment){

            throw new Error("Comment does not exist")

        }

        if(typeof data.title == "string"){

            comment.title = data.title

        }

        if(typeof data.body == "string"){

            comment.body = data.body

        }

        if(!originalComment){

            pubsub.publish("comment",{

                comment:{

                    mutation:'CREATED',

                    data:comment

                }

            })

        }

        if(originalComment){

            pubsub.publish("comment",{

                comment:{

                    mutation:'UPDATED',

                    data:comment

                }

            })

        }

        return comment

    },

    updateReview(parent, {data, id}, {db, pubsub}, info){

        const review = db.reviewsArray.find(rev => rev.id == id)

        const originalReview = {...review}

        if(!review){

            throw new Error('Review does not exist')

        }

        if(typeof data.title == "string"){

            review.title = data.title

        }

        if(typeof data.body == "string"){

            review.body = data.body

        }

        if(typeof data.upvotes =="number"){

            review.upvotes = data.upvotes

        }

        if(typeof data.isPublished =="boolean"){

            review.isPublished = data.isPublished

        }

        if(!originalReview && review.isPublished){

            pubsub.publish("review",{

                review:{

                    mutation:'CREATED',

                    data:review

                }

            })

        }

        if(originalReview && !review.isPublished){

            pubsub.publish("review",{

                review:{

                    mutation:'DELETED',

                    data:review

                }

            })

        }

        if(originalReview && review.isPublished){

            pubsub.publish("review",{

                review:{

                    mutation:'UPDATED',

                    data:review

                }

            })

        }

        return review

    },

    updateProd(parent,{data,id},{db},info){

        const prod = db.prodArray.find(prod => prod.prodID == id)

        if(!prod){

            throw new Error('Product does not exist')

        }

        if(typeof data.name === "string"){

            prod.name = data.name

        }

        if(typeof data.price === "number"){

            prod.price = data.price

        }

        if(typeof data.stockUnits === "number"){

            prod.stockUnits = data.stockUnits

        }

        if(typeof data.name === "string"){

            prod.name = data.name

        }

        if(typeof data.isAvailable === "boolean"){

            prod.isAvailable = data.isAvailable

        }

        return prod

    }

}

export {Mutation as default}

**Inside resolvers/Subscription.js**

const Subscription = {

    count:{

        subscribe(parent, args, {pubsub}, info){

            let number = 0;

            setInterval(()=>{

                number++;

                pubsub.publish("mycountingchannel", {

                    count : number

                })

            },300 )

            return pubsub.asyncIterator("mycountingchannel")

        }

    },

    comment:{

        subscribe(parent, {parentRevOfComm}, {db, pubsub}, info){

            const review = db.reviewsArray.find(rev => rev.id == parentRevOfComm)

            if(!review){

                throw new Error('Review does not exist')

            }

            return pubsub.asyncIterator("comment")

        }

    },

    review:{

        subscribe(parent,args, {pubsub}, info){

            return pubsub.asyncIterator("review")

        }

    }

}

export {Subscription as default}