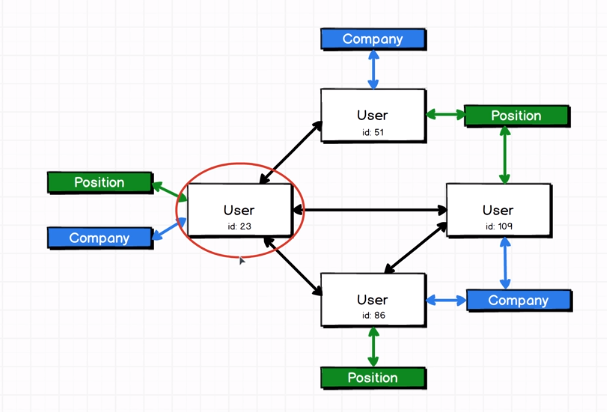
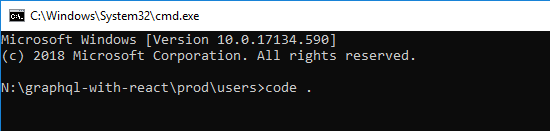
**LECTURE 5 – What is GraphQL**



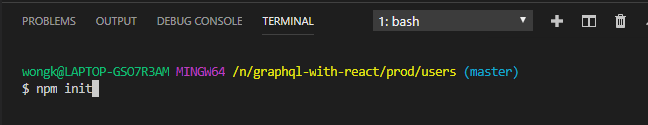
* This is the graph that graphql is referring to.

**LECTURE 6 – Working with GraphQL**

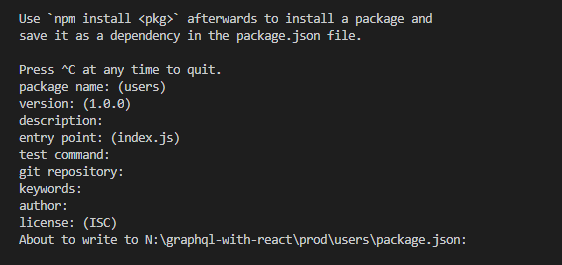
* Create a folder called **prod**.
* Inside that folder create another folder called **users**.
* Inside the users folder open up **vs code**.



* Open up a terminal and create a **package.json** file by using the command **npm init**.



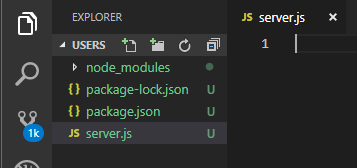
* Answer yes to all the questions by simply pressing **ENTER** until the package.json file has been created.



* Install the following NPM packages by typing **npm install –-save express express-graphql graphql lodash**.



* There are 4 packages being installed: **express**, **express-graphql**, **graphql** and **lodash**.
* At the root of our directory create a new file called **server.js**.



* In **server.js** write the following code to listen on port 4000 and console log the word ‘Listening’.

const express = require('express');

const app = express();

app.listen(4000, () => {

console.log('Listening');

});

**LECTURE 7 – Registering GraphQL with Express**

* In **server.js** import the GraphQL translation layer package.

const express = require('express');

const expressGraphQL = require('express-graphql');

const app = express();

* Add the following GraphQL middleware.

const express = require('express');

const expressGraphQL = require('express-graphql');

const app = express();

app.use('/graphql', expressGraphQL({

graphiql: true

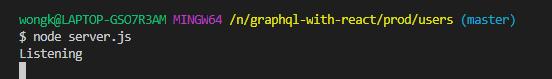
}));

app.listen(4000, () => {

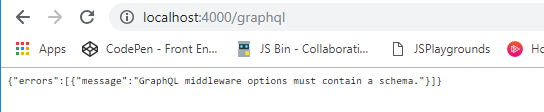
console.log('Listening');

});

* Run the server by typing in the terminal **node server.js**.

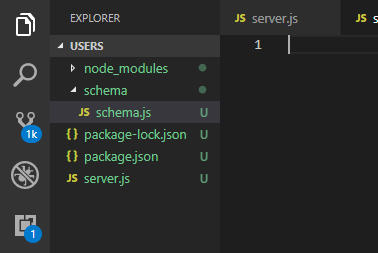


* In a web browser navigate to the url of our application, which is **localhost:4000/graphql**.



**LECTURE 8 – GraphQL Schemas**

* **app.use** is how we wire up middleware to an express application.
* Middlewares are tiny functions used to intercept or modify requests as they come through an express server.
* When we registered **expressGraphQL** it was registered as a middleware.
* A **Schema** tells GraphQL how all the data in our application is arranged and how it can be accessed. The Schema file is the absolute lynch pin of every GraphQL application.
* The Schema file tells GraphQL what type of data we are working with and how those different relations play out between the pieces of data.
* A Schema is what tells GraphQL what our data looks like.
* We will now create a Schema file.
* In the Root directory create a folder called **schema** and inside that folder create a file called **schema.js**.



**LECTURE 9 – Writing a GraphQL Schema**

* In the **schema.js** file, import the **graphql** library.

const graphql = require('graphql');

* We will now do some destructuring from graphql. Write the following code.

const graphql = require('graphql');

const {

GraphQLObjectType

} = graphql;

* Create the following object with the following properties.

const graphql = require('graphql');

const {

GraphQLObjectType

} = graphql;

const UserType = new GraphQLObjectType({

name: 'User',

fields: {

id: { type: GraphQLString },

firstName: { type: GraphQLString },

age: { type: GraphQLInt }

}

});

* Since we used the GraphQLString and GraphQLInt properties from graphql we need to import through destructuring. Make the following modifications.

const graphql = require('graphql');

const {

GraphQLObjectType,

GraphQLString,

GraphQLInt

} = graphql;

const UserType = new GraphQLObjectType({

name: 'User',

fields: {

id: { type: GraphQLString },

firstName: { type: GraphQLString },

age: { type: GraphQLInt }

}

});

**LECTURE 10 – Root Queries**

* GraphQL has difficulty finding a specific item within a set of data. To assist GraphQL we need to use a **Root Query**.
* A **Root Query** is something that allows us to jump into our graph of data.
* The Root Query is the entry point into our data.
* Write the following Root Query.

const graphql = require('graphql');

const {

GraphQLObjectType,

GraphQLString,

GraphQLInt

} = graphql;

const UserType = new GraphQLObjectType({

name: 'User',

fields: {

id: { type: GraphQLString },

firstName: { type: GraphQLString },

age: { type: GraphQLInt }

}

});

const RootQuery = new GraphQLObjectType({

name: 'RootQueryType',

fields: {

user: {

type: UserType,

args: { id: { type: GraphQLString } }

}

}

});

* The args parameter specifies the arguments that are necessary for the Root Query of this specific user.
* This Root Query is saying that if you’re looking for a User, and you give me an id, I’ll give you back a User.
* Within the Root Query we also need to add the resolve function.
* The **resolve** function is where we actually go into our database and we find the actual data that we are looking for. It executes that instructions that we have written before for the Root Query.
* Make the following amendments to the Root Query.

const UserType = new GraphQLObjectType({

name: 'User',

fields: {

id: { type: GraphQLString },

firstName: { type: GraphQLString },

age: { type: GraphQLInt }

}

});

const RootQuery = new GraphQLObjectType({

name: 'RootQueryType',

fields: {

user: {

type: UserType,

args: { id: { type: GraphQLString } },

resolve(parentValue, args) {

}

}

}

});

**LECTURE 11 – Resolving with Data**

* Rather than use a database for our data we are just going to create a hard-coded list of data.
* In **schema.js** create the following array of data.

const graphql = require('graphql');

const {

GraphQLObjectType,

GraphQLString,

GraphQLInt

} = graphql;

const users = [

{ id: '23', firstName: 'Bill', age: 20 },

{ id: '47', firstName: 'Samantha', age: 21 }

]

const UserType = new GraphQLObjectType({

name: 'User',

fields: {

id: { type: GraphQLString },

firstName: { type: GraphQLString },

age: { type: GraphQLInt }

}

});

* At the top import **lodash**.

const graphql = require('graphql');

const \_ = require('lodash');

const {

GraphQLObjectType,

GraphQLString,

GraphQLInt

} = graphql;

const users = [

{ id: '23', firstName: 'Bill', age: 20 },

{ id: '47', firstName: 'Samantha', age: 21 }

]

* Make the following changes to the Root Query.

const UserType = new GraphQLObjectType({

name: 'User',

fields: {

id: { type: GraphQLString },

firstName: { type: GraphQLString },

age: { type: GraphQLInt }

}

});

const RootQuery = new GraphQLObjectType({

name: 'RootQueryType',

fields: {

user: {

type: UserType,

args: { id: { type: GraphQLString } },

resolve(parentValue, args) {

return \_.find(users, { id: args.id });

}

}

}

});

* Import the following **helper** from the graphql library.

const graphql = require('graphql');

const \_ = require('lodash');

const {

GraphQLObjectType,

GraphQLString,

GraphQLInt,

GraphQLSchema

} = graphql;

const users = [

{ id: '23', firstName: 'Bill', age: 20 },

{ id: '47', firstName: 'Samantha', age: 21 }

]

* GraphQLSchema takes in a Root Query and returns a GraphQL instance.
* Create the following GraphQLSchema object.

const RootQuery = new GraphQLObjectType({

name: 'RootQueryType',

fields: {

user: {

type: UserType,

args: { id: { type: GraphQLString } },

resolve(parentValue, args) {

return \_.find(users, { id: args.id });

}

}

}

});

module.exports = new GraphQLSchema({

query: RootQuery

});

* Now we will need to import all this into our server.js file.
* Go back to **server.js** and make the following changes.

const express = require('express');

const expressGraphQL = require('express-graphql');

const schema = require('./schema/schema');

const app = express();

app.use('/graphql', expressGraphQL({

schema,

graphiql: true

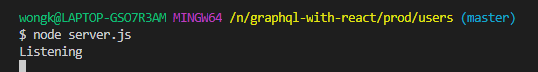
}));

app.listen(4000, () => {

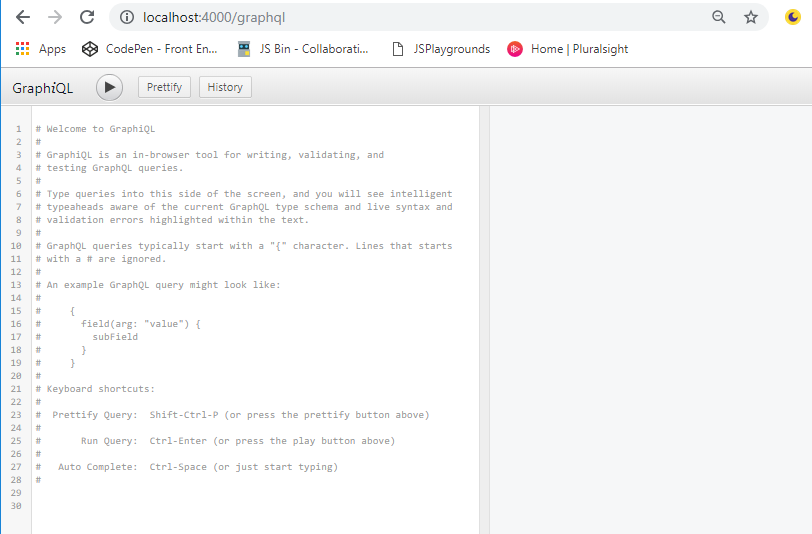
console.log('Listening');

});

* Now we will use this in the browser.
* First, restart the server by cancelling any existing server with CTRL + C and then typing **node server.js**.



* Now go to a web browser and navigate to **localhost:4000/graphql**
* You should get the following pop-up in the browser.



**LECTURE 12 – The GraphiQL tool**

* In the left-pane of the GraphiQL tool, type the following query.

{

user(id: "23") {

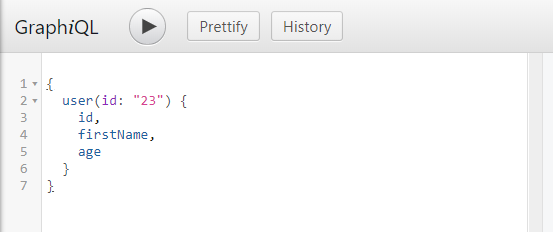
id,

firstName,

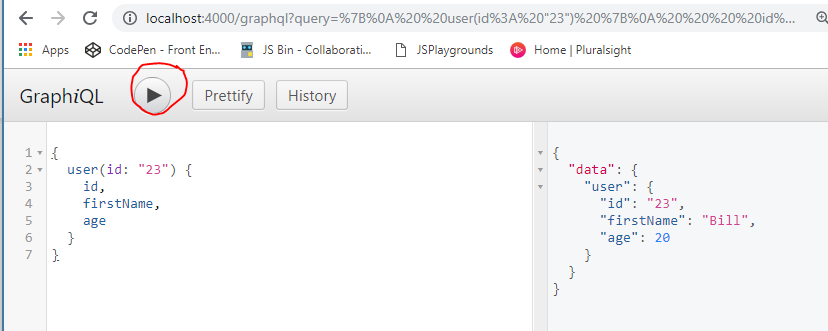
age

}

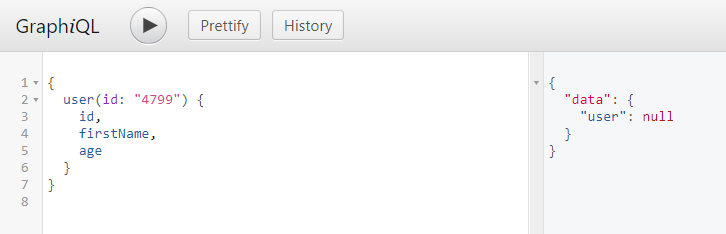
}



* Hit on the **Play** icon to execute the query, and the results should show up on the right-hand panel.



* If you provide an id that doesn’t exist then you will get a **null** response.



* If you don’t provide an id at all then you will get an error response.

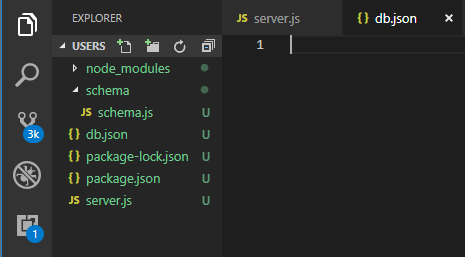


**LECTURE 13 – A Realistic Data Source**

* We will be using an npm package called **JSON Server** to act as our dynamic datastore.
* In the terminal, stop your server.
* In the terminal type **npm install –-save json-server**.



* At the root of the directory, create a new file called **db.json**.



* In the **db.json** file create the following data objects.

{

"users": [

{ "id": "23", "firstName": "Bill", "age": 20 },

{ "id": "40", "firstName": "Alex", "age": 40 }

]

}

* We are going to write a little helper code to start the **db.json**.
* Go to **package.json** and write the following code.

{

"name": "users",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"json:server": "json-server --watch db.json"

},

"author": "",

"license": "ISC",

"dependencies": {

"express": "^4.16.4",

"express-graphql": "^0.7.1",

"graphql": "^14.1.1",

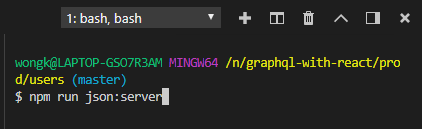
"json-server": "^0.14.2",

"lodash": "^4.17.11"

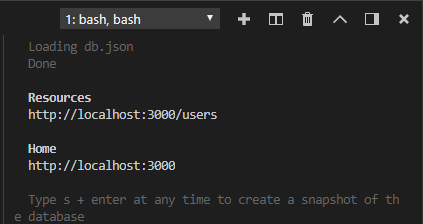
}

}

* We now need to run both servers so you can split the terminal or open up a new one.
* In the new terminal window, run the db.json server by typing **npm run json:server**.



* The server is telling us that the data is being served up at [**http://localhost:3000/users**](http://localhost:3000/users).

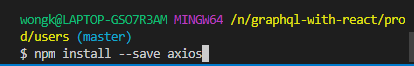


* Navigate to that address in the web browser.



**LECTURE 14 – Async Resolve Functions**

* We will need to install **axios** from npm.
* In the terminal type **npm install –-save axios**.



* Go to **shema.js** and import **axios**. Also, delete **lodash** as we no longer need it.

const graphql = require('graphql');

const axios = require('axios');

const \_ = require('lodash');

const {

GraphQLObjectType,

GraphQLString,

GraphQLInt,

GraphQLSchema

} = graphql;

const users = [

{ id: '23', firstName: 'Bill', age: 20 },

{ id: '47', firstName: 'Samantha', age: 21 }

]

* Delete the static list of users.

const graphql = require('graphql');

const axios = require('axios');

const {

GraphQLObjectType,

GraphQLString,

GraphQLInt,

GraphQLSchema

} = graphql;

const users = [

{ id: '23', firstName: 'Bill', age: 20 },

{ id: '47', firstName: 'Samantha', age: 21 }

]

const UserType = new GraphQLObjectType({

name: 'User',

fields: {

id: { type: GraphQLString },

firstName: { type: GraphQLString },

age: { type: GraphQLInt }

}

});

* Make the following changes to the Root Query.

const UserType = new GraphQLObjectType({

name: 'User',

fields: {

id: { type: GraphQLString },

firstName: { type: GraphQLString },

age: { type: GraphQLInt }

}

});

const RootQuery = new GraphQLObjectType({

name: 'RootQueryType',

fields: {

user: {

type: UserType,

args: { id: { type: GraphQLString } },

resolve(parentValue, args) {

return axios.get(`http://localhost:3000/users/${args.id}`)

.then(resp => resp.data);

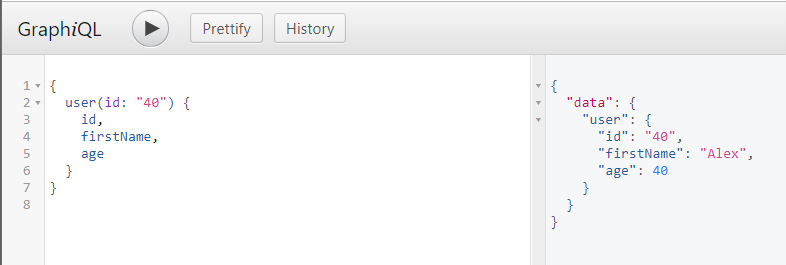
}

}

}

});

* Now we will test this.
* Go back to the terminal and restart the server by typing **node server.js**.
* Go back to GraphiQL and send a query.



**LECTURE 15 – Nodemon Hookup**

* Install **nodemon** by going to the terminal and typing **npm install –-save nodemon**.



* Now we will make changes to the **package.json** file to add another script for the dev environment.
* Go to **package.json** and make the following changes.

{

"name": "users",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"json:server": "json-server --watch db.json",

"dev": "nodemon server.js"

},

"author": "",

"license": "ISC",

"dependencies": {

"axios": "^0.18.0",

"express": "^4.16.4",

"express-graphql": "^0.7.1",

"graphql": "^14.1.1",

"json-server": "^0.14.2",

"lodash": "^4.17.11",

"nodemon": "^1.18.10"

}

}

* Go back to the terminal and restart the server by typing **npm run dev**.

**LECTURE 16 – Company Definitions**

* We are now going to hookup relating a company to a user.
* Go to **db.json** and add the following data.

{

"users": [

{ "id": "23", "firstName": "Bill", "age": 20 },

{ "id": "40", "firstName": "Alex", "age": 40 }

],

"companies": [

{ "id": "1", "name": "Apple", "description": "iphone" },

{ "id": "2", "name": "Google", "description": "search" }

]

}

* Add an extra user.

{

"users": [

{ "id": "23", "firstName": "Bill", "age": 20 },

{ "id": "40", "firstName": "Alex", "age": 40 },

{ "id": "41", "firstName": "Nick", "age": 40 }

],

"companies": [

{ "id": "1", "name": "Apple", "description": "iphone" },

{ "id": "2", "name": "Google", "description": "search" }

]

}

* Add in a **companyId** property for each user.

{

"users": [

{ "id": "23", "firstName": "Bill", "age": 20, "companyId": "1" },

{ "id": "40", "firstName": "Alex", "age": 40, "companyId": "2" },

{ "id": "41", "firstName": "Nick", "age": 40, "companyId": "2" }

],

"companies": [

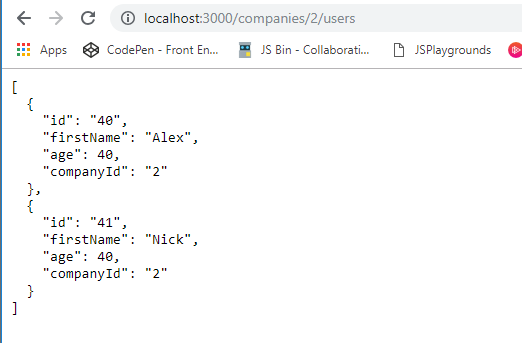
{ "id": "1", "name": "Apple", "description": "iphone" },

{ "id": "2", "name": "Google", "description": "search" }

]

}

* Now if you go to a web browser and type in the url: [**http://localhost:3000/users**](http://localhost:3000/users) you should get the list of users, and if you type [**http://localhost:3000/companies**](http://localhost:3000/companies) you should get the list of companies.
* Now if you want to get the list of users that work for Google (companyId of 2), in the web browser just type in the Url [**http://localhost:3000/companies/2/users**](http://localhost:3000/companies/2/users).



**LECTURE 17 – Nested Queries**

* We are now going to add the idea of a **CompanyType** into our Schema.
* Go to **schema.js** and add the following code, making sure that that CompanyType is **ABOVE** the UserType.

const graphql = require('graphql');

const axios = require('axios');

const {

GraphQLObjectType,

GraphQLString,

GraphQLInt,

GraphQLSchema

} = graphql;

const CompanyType = new GraphQLObjectType({

name: "Company",

fields: {

id: { type: GraphQLString },

name: { type: GraphQLString },

description: { type: GraphQLString }

}

});

const UserType = new GraphQLObjectType({

name: 'User',

fields: {

id: { type: GraphQLString },

firstName: { type: GraphQLString },

age: { type: GraphQLInt }

}

});

* Add the following CompanyType field to the UserType object.

const CompanyType = new GraphQLObjectType({

name: "Company",

fields: {

id: { type: GraphQLString },

name: { type: GraphQLString },

description: { type: GraphQLString }

}

});

const UserType = new GraphQLObjectType({

name: 'User',

fields: {

id: { type: GraphQLString },

firstName: { type: GraphQLString },

age: { type: GraphQLInt },

company: {

type: CompanyType

}

}

});

**LECTURE 18 – More on Nested Queries**

* You will notice in the UserType object, instead of creating a field called **companyId** we called it just **company**.
* Make the following changes to the UserType object.
* Go to **schema.js** and make the following changes.

const CompanyType = new GraphQLObjectType({

name: "Company",

fields: {

id: { type: GraphQLString },

name: { type: GraphQLString },

description: { type: GraphQLString }

}

});

const UserType = new GraphQLObjectType({

name: 'User',

fields: {

id: { type: GraphQLString },

firstName: { type: GraphQLString },

age: { type: GraphQLInt },

company: {

type: CompanyType,

resolve(parentValue, args) {

return axios.get(`http://localhost:3000/companies/${parentValue.companyId}`)

.then(res => res.data)

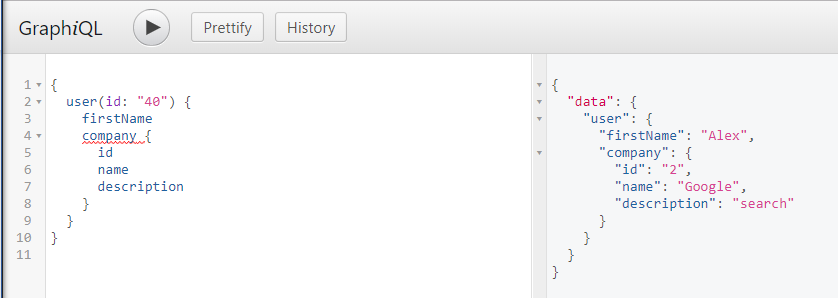
}

}

}

});

* Now go back to GraphiQL and write the following query to get the Company Id, name and description of the company that User of Id 40 works for.



**LECTURE 19 – A Quick Breather**

**LECTURE 20 – Multiple RootQuery Entry Points**

* Sfsdf
* Sd
* Fsd
* Fsd
* Fsd
* Fdd
* Sdfs
* Fss
* Sf
* Sf
* Sff
* Fs
* Asd
* Fa
* Sf
* Asf
* Asf
* Asf
* fasdf
* Sfsdf
* Sd
* Fsd
* Fsd
* Fsd
* Fdd
* Sdfs
* Fss
* Sf
* Sf
* Sff
* Fs
* Asd
* Fa
* Sf
* Asf
* Asf
* Asf
* fasdf
* Sfsdf
* Sd
* Fsd
* Fsd
* Fsd
* Fdd
* Sdfs
* Fss
* Sf
* Sf
* Sff
* Fs
* Asd
* Fa
* Sf
* Asf
* Asf
* Asf
* fasdf
* Sfsdf
* Sd
* Fsd
* Fsd
* Fsd
* Fdd
* Sdfs
* Fss
* Sf
* Sf
* Sff
* Fs
* Asd
* Fa
* Sf
* Asf
* Asf
* Asf
* fasdf
* Sfsdf
* Sd
* Fsd
* Fsd
* Fsd
* Fdd
* Sdfs
* Fss
* Sf
* Sf
* Sff
* Fs
* Asd
* Fa
* Sf
* Asf
* Asf
* Asf
* fasdf
* Sfsdf
* Sd
* Fsd
* Fsd
* Fsd
* Fdd
* Sdfs
* Fss
* Sf
* Sf
* Sff
* Fs
* Asd
* Fa
* Sf
* Asf
* Asf
* Asf
* fasdf
* Sfsdf
* Sd
* Fsd
* Fsd
* Fsd
* Fdd
* Sdfs
* Fss
* Sf
* Sf
* Sff
* Fs
* Asd
* Fa
* Sf
* Asf
* Asf
* Asf
* fasdf
* Sfsdf
* Sd
* Fsd
* Fsd
* Fsd
* Fdd
* Sdfs
* Fss
* Sf
* Sf
* Sff
* Fs
* Asd
* Fa
* Sf
* Asf
* Asf
* Asf
* fasdf
* Sfsdf
* Sd
* Fsd
* Fsd
* Fsd
* Fdd
* Sdfs
* Fss
* Sf
* Sf
* Sff
* Fs
* Asd
* Fa
* Sf
* Asf
* Asf
* Asf
* fasdf
* Sfsdf
* Sd
* Fsd
* Fsd
* Fsd
* Fdd
* Sdfs
* Fss
* Sf
* Sf
* Sff
* Fs
* Asd
* Fa
* Sf
* Asf
* Asf
* Asf
* fasdf