https://www.youtube.com/watch?v=fov5e6XJgwc&list=RDCMUC8S4rDRZn6Z\_StJ-hh7ph8g&start\_radio=1

https://github.com/facebook/create-react-app

0) Open vs code in an empty directory and create "client" and "server" folder

**Creating the server**

1) cd to the server directory and enter "yarn init"

2) This will create a package.json (you can leave the fields empty)

3) Enter "tsc --init". If tsc is not installed, run "npm install -g typescript" first

4) This will create a tsconfig.json

5) Install the basic packages that your project might require, e.g. "yarn add typescript ts-node express cors nodemon"

6) Install @types for the packages, e.g. "yarn add @types/cors @types/express @types/node"

|  |
| --- |
|  |

7) Create a folder in the server directory called “dist” and another folder called “src”

8) (Important) In the tsconfig file, change “target” to “es6” and uncomment “moduleResolution”, "strictPropertyInitialization", "experimentalDecorators", “emitDecoratorMetadata”

9) If you have a typescript application, your code will not be converted into typescript because no browser will understand typescript – they only can understand JavaScript. Therefore, we must redirect the typescript files to another directory for conversion into JavaScript and a file that contains the converted JavaScript code

10) In the tsconfig file, uncomment and change “rootDir” to “./src” and “outDir” to “./dist”

11) package.json should look like this (version might be different):

|  |
| --- |
|  |

12) If you are working with databases, you might need an ORM for your project

|  |  |  |
| --- | --- | --- |
| [database - What is an ORM, how does it work, and how should I use one? - Stack Overflow](https://stackoverflow.com/questions/1279613/what-is-an-orm-how-does-it-work-and-how-should-i-use-one)  [Object-Relational Mapping](https://en.wikipedia.org/wiki/Object-relational_mapping) (ORM) is a technique that lets you query and manipulate data from a database using an object-oriented paradigm. When talking about ORM, most people are referring to a *library* that implements the Object-Relational Mapping technique, hence the phrase "an ORM".  An ORM library is a completely ordinary library written in your language of choice that encapsulates the code needed to manipulate the data, so you don't use SQL anymore; you interact directly with an object in the same language you're using.  For example, here is a completely imaginary case with a pseudo language:  You have a book class, you want to retrieve all the books of which the author is "Linus". Manually, you would do something like that:   |  | | --- | | book\_list = new List();  sql = "SELECT book FROM library WHERE author = 'Linus'";  data = query(sql); // I over simplify ...  while (row = data.next())  {  book = new Book();  book.setAuthor(row.get('author');  book\_list.add(book);  } |   With an ORM library, it would look like this:   |  | | --- | | book\_list = BookTable.query(author="Linus"); |   The mechanical part is taken care of automatically via the ORM library.  **Using ORM saves a lot of time because:**   * [DRY](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself): You write your data model in only one place, and it's easier to update, maintain, and reuse the code. * A lot of stuff is done automatically, from database handling to [I18N](https://en.wikipedia.org/wiki/Internationalization_and_localization). * It forces you to write [MVC](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) code, which, in the end, makes your code a little cleaner. * You don't have to write poorly-formed SQL (most Web programmers really suck at it, because SQL is treated like a "sub" language, when in reality it's a very powerful and complex one). * Sanitizing; using prepared statements or transactions are as easy as calling a method.   **Using an ORM library is more flexible because:**   * It fits in your natural way of coding (it's your language!). * It abstracts the DB system, so you can change it whenever you want. * The model is weakly bound to the rest of the application, so you can change it or use it anywhere else. * It lets you use OOP goodness like data inheritance without a headache.   **But ORM can be a pain:**   * You have to learn it, and ORM libraries are not lightweight tools; * You have to set it up. Same problem. * Performance is OK for usual queries, but a SQL master will always do better with his own SQL for big projects. * It abstracts the DB. While it's OK if you know what's happening behind the scene, it's a trap for new programmers that can write very greedy statements, like a heavy hit in a for loop. |

13) Install additional packages for querying and editing the database “yarn add typeorm mysql2 graphql express-graphql”

14) Add the “scripts” to package.json

|  |
| --- |
|  |

15) Create a “index.ts” file in the server/src folder – this will be the entry point of our application

16) Populate the “index.ts” file and verify that there are no errors with running the main function via “**yarn dev**” – you should see the “Server running on port 3001” on the terminal (take note the graphql part is commented for now – we will uncomment it in step 21 after setting the graphQL schema)

|  |
| --- |
|  |

17) Connect to the database via typeorm in the main function

|  |
| --- |
|  |

18) Create a “Schema” folder in the server, with the following subfolders: “Mutations”, “Queries”, and “TypeDefs”. Also create a file called “index.ts” in the same “Schema” folder.

19) In the “TypeDefs” folder, create a “User.ts” file and populate it

|  |
| --- |
|  |

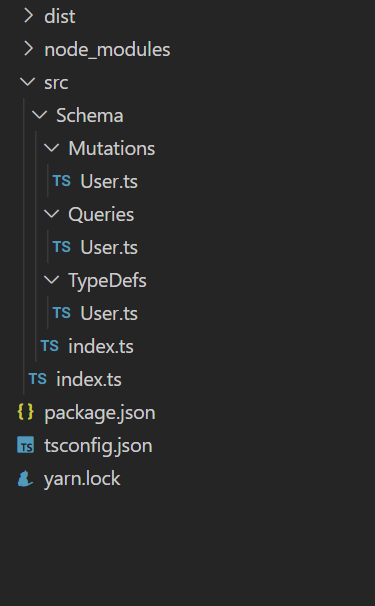
20) In the index.ts file of the “Schema” folder, compile the all Query and Mutation and export it as a schema to the main index.ts file

|  |
| --- |
|  |

21) Un-comment the graphql part in the main index.ts and key in “yarn dev”. Connect to “localhost:3001/graphql” to verify that it is working

|  |
| --- |
|  |

22) Create a “User.ts” file for Mutations and Queries



23) Populate the “User.ts” file in the “Queries” folder

|  |
| --- |
|  |

24) Populate the “User.ts” file in the “Mutations” folder

|  |
| --- |
|  |

25) Import the Query and Mutation into the schema index.ts folder and deploy them in their respective fields

|  |
| --- |
|  |

26) Now we need to integrate typeorm into our project – create a new “Entities” folder under the “src” branch and create a new “Users.ts” file

|  |
| --- |
|  |

27) Import the newly created class and edit the “createConnection” function in the main index.ts file (ensure that the database, username and password are all valid). If everything is ok up until this step – the SQL query will run in the terminal when it is saved or started

|  |
| --- |
|  |

28) Once the table has been created in SQL (you can verify this in SQL Workbench), change in the “createConnection” function, change “synchronize” to false and edit the “index.ts” file in the “Queries” folder to accommodate the change

|  |
| --- |
|  |

29) The query can be tested on the GraphQL playground

|  |
| --- |
|  |

30) Edit the “index.ts” file in the “Mutations” folder to accommodate creating a user

|  |
| --- |
|  |

31) Test the mutation function in the GraphQL playground

|  |
| --- |
|  |

32) Refresh the database and check if the entry has been added, or you can run the query function to check if it can correctly retrieve the data

|  |
| --- |
|  |

33) Add in the function to delete a user in the “User.ts” file in the “Mutations” folder

|  |
| --- |
|  |

34) Add the newly created function in the “Index.ts” file of the “Schema” folder

|  |
| --- |
|  |

35) Delete the 2entry of the database on the GraphQL playground and validate the result

|  |
| --- |
|  |

36) Repeat steps 33-34 for the “UPDATE PASSWORD” function

|  |
| --- |
|  |

35) Verify that the function is working in the GraphQL Playground

|  |
| --- |
|  |

36) Create a new “Messages.ts” file in the “TypeDefs” folder and create a new type that handles all messages for error handling – this will return if an entry is successful or not, along with a message

|  |
| --- |
|  |

37) Change the return type of the functions in the “Mutations” folder

|  |
| --- |
|  |

|  |
| --- |
|  |

38) Verify that the messages are correct in the GraphQL playground

|  |
| --- |
|  |

|  |
| --- |
|  |

**Creating the client**

1) In the client folder, turn off the strict-ssl on yarn/npm config

yarn config set “strict-ssl” false -g

2) Create all the files and modules that you need for the front-end

yarn create react-app . --template typescript

3) Delete the following files, and afterwards remove all instances where they may be imported in the remaining files:

app.test.tsx

index.css

logo.svg

setupTests.js

4) Delete all the code in App.css (but do not delete the file)

5) Remove “./index.css” from index.tsx

6) Delete the header part of App.tsx

|  |
| --- |
|  |

7) Upon running “yarn start”, you should see an empty page at “localhost:3000”

8) Add the following dependencies for our project

yarn add @apollo/client graphql

9) Import and initialize the @apollo/client in App.tsx – make sure that the server is running as well!

|  |
| --- |
|  |

10) Populate the front-end with 3 input boxes and a button to capture the name, username and password

|  |
| --- |
|  |

11) Create a folder in the “src/” called “Graphql” to hold the mutations and queries. In this “Graphql” folder, create a “Mutation.tsx” and “Queries.tsx”

12) Populate the “Mutation.tsx” with the “createUser” function from the server

|  |
| --- |
|  |

13) Import “{gql}” for the “Queries.tsx” file as well to silence the error

14) Create a new folder “components” under the “src” branch. Then, create a “CreateUser.tsx” file and cut out the relevant parts from the main “App.tsx”

|  |
| --- |
|  |

15) Clean up the imports (if not already done) and import in the newly created “CreateUser” component

|  |
| --- |
|  |

16) Test and verify that the front-end is working by adding a new user, and then querying the backend

|  |
| --- |
|  |

|  |
| --- |
|  |

17) Populate the “Queries.tsx” with the “getAllUsers” function from the server (under Server/src/Schema/index.ts)

|  |
| --- |
|  |

18) IMPORTANT – it is good practice to test the query on the GraphQL playground first – the above error will throw a POST http://localhost:3001/graphql 400 (Bad Request) error because it does not recognize ‘userName’ (it is spelled as “username” in the server/src/schema/typedefs/users.ts” file)

The correct syntax and message that you should see are as follows:

|  |
| --- |
|  |

19) Complete the UI for the ListOfUsers component

|  |
| --- |
|  |

20) Add in the mutation for deleting a user in the “Mutation.tsx” file

|  |
| --- |
|  |

21) Create a new file in the “Components” folder called “UpdatePassword.tsx” – this will be the UI to update the password for a user

|  |
| --- |
| import React, {useState} from 'react';  import {UPDATE\_USER\_PASSWORD} from '../Graphql/Mutation';  import {useMutation} from "@apollo/client";  function UpdatePassword() {         // Setters         const [username, setUserName] = useState("");         const [oldPassword, setCurrentPassword] = useState("");         const [newPassword, setNewPassword] = useState("");         // updatePassword - the function that you want to call         const [updatePassword, {error}] = useMutation(UPDATE\_USER\_PASSWORD);        return (          <div>              <input type="text" placeholder="Username" onChange={(event)=>{setUserName(event.target.value);}}/>              <input type="password" placeholder="Current Password" onChange={(event)=>{setCurrentPassword(event.target.value);}}/>              <input type="password" placeholder="New Password" onChange={(event)=>{setNewPassword(event.target.value);}}/>              <button onClick={()=>{updatePassword({variables: {username: username, oldPassword: oldPassword, newPassword: newPassword}})}}>UPDATE PASSWORD</button>          </div>      )  }  export default UpdatePassword; |

22) Don’t forget to write the gql query in the “Mutations.tsx” file as well

|  |
| --- |
|  |

23) Test out the function and see that the password changes (if the fields are valid)

[Prisma ORM Tutorial for Beginners | CRUD, CreateMany, Associations... - YouTube](https://www.youtube.com/watch?v=E37-33M6Ypk&list=RDCMUC8S4rDRZn6Z_StJ-hh7ph8g&index=5)