Sign up form:

# Client Side

→ to handle signup button

```
        const signupButton = document. getElementById("signupBtn")
        signupButton.addEventListener('click', handelSignup)
```

→ add event listener to signup button means that when the client makes an event which is '**click'** event in this ex. Then, execute a bock of code which is **handelSignup** method

→To read the values:
Ex:

```
const name = document.getElementById('nameId').value
const email =document.getElementById('emailId').value,
```

To read **all** the forma values you can do this way
→ note: this code inside **handelSignup** method

```
const formValue = {

  name:document.getElementById('nameId').value,
 email: document.getElementById('emailId').value,
 pass1: document.getElementById('passId').value,
 pass2: document.getElementById('passIdRepeat').value,
 course: document.getElementById('courseId').value,
}
```

Validate the form values on the client side:

```
const validateSignup = (formValue) => {
        if ((! formValue.email || formValue.email === "")) {
                showError('Please provide an email')
                return false;
                }
        }
```

Check if the values exist or not → if it is not provided, so we can show a message to warn the client to enter the item value ex: email

→ **insertAdjacentHTML**: to insert html code into a specified position
→ syntax: `element.insertAdjacentHTML(position, html)`
→ `position: (the position relative to the element)` **`afterbegin, afterend, beforebegin, beforeend`**

```
Ex: body.insertAdjacentHTML('beforeend', <div>……</div>)
```

https://developer.mozilla.org/en-US/docs/Web/API/Element/insertAdjacentHTML

→ after form validation, make a req to the server to save the data

→ using **fetch** method taking request info.
First parameter: **url** or the endpoint
Second parameter: list of req info like request **method**, request **headers**, request **body**
Returning: Promise so **await** it inside **async** method
In this ex: storing the result into **response**.

```
const response = await fetch('/signup', {
       method: 'POST',
       headers: {'Content-Type': 'application/json'},
       body: JSON.stringify(formValue)


})
```

JSON.stringify → convert JavaScript value to JSON string

`if(response.status !== 200)` → if the status is 200 means the request was successful

# Server Side

1- Install mongoose package → **npm i mongoose** → mongoose is a library for connecting and dealing with MongoDB
2- Require mongoose → `const mongoose = require('mongoose')`
3- Create a connection to DB

```
         →let connection = undefined;
            const getConnection = async () => {
            if(connection) {
                   console.log('returning existing connection')
                   return connection
                   }
            else{
                   connection = await
                   mongoose.connect('mongodb+srv://comit:qNuhtBKAwzOchJGt@clus
                   ter0.fk2n5r2.mongodb.net/?retryWrites=true&w=majority')
                   return connection;
                   }
```

- Define the connection at undefined at first and then check if the connection exists or not → if it exists, so don't have to create it again. If not, create it.
- To create the connection, we need **mongoose.connect**(connectionString)
  Note: connection string -> url to the database using the credential (username and password)

  `NOTE: module.exports = { getConnection,  mongoose, Schema: mongoose.Schema}` → here we export getConnection, mongoose, the schema to use them in another file and then we can create the model → instead of requiring mongoose again in another file. We

```
can require all of them just from one file which is in this case
'../db/mongoose.js'
```

4- Create the schema → Schema represents the structure of the document

See→ users_modules/model.js

Ex:

```
const userSchema = new Schema ({  → Schema is mongoose.schema
  name:    String,                →name property of String type
  email: {                        →email property has some properties (String
  type: String,                   type, it is required, and unique which
  required: true,                 cannot be repeated)
  unique: true,
  },

  createdAt: {                    →createdAt has Date type and the default
  type: Date,                     value is Date.Now to know when this document
  default: Date.Now               has been created
  }
})
```

5- Define or create the model

```
const userModel = mongoose.model('User', userSchema)
User → 1st param, the name of the model
userSchema→ 2nd param, the schema we created before
```

6- Create new user

See → user_modules/ service.js

```
const {userModel} = require('./model.js')
const storeUser = async (userData) => {
   const user = new userModel(userData). →create object from userModel
   try {
          await user.save(). → save user
       }
   catch(err) {
          throw 'failed to create user, please check your input'
    }

   }

   → using try and catch block to catch the error if it found
```

There is another way to create a user → **`userModel.create(userData)`**

7- **`In app.js`**

```
const {getConnection} = require('./db/mongoose')
const userService = require('./users_module/service') →which stored
→ to require the connection, we created before           the user data

'/signup' → this is the endpoint

app.post('/signup', async (req, res) => { → in case of post request
try {
        await userService.storeUser(req.body) →req.body → data in the
    } catch(err) {                                              request body
      res.status(400).json({
                error: err
    })
    return
    }
    res.status(200).json({
        message: "user created sucessfully"
    })
})
```