Experiment No. 2
Web X.0 Lab

Aim: To study basic Constructs, Inheritance, Access modifiers and Creation of Web Pages in TypeScript.

Theory:

How do you compile TypeScript files?
TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. It offers classes, modules, and interfaces to help you build robust components.

Install the TypeScript compiler
The easiest way to install TypeScript is through npm, the Node.js Package Manager. If you have npm installed, you can install TypeScript globally (-g) on your computer by:

```
npm install -g typescript
```

You can test your install by checking the version.

```
tsc --version
```

Another option is to install the TypeScript compiler locally in your project:

```
npm install --save-dev typescript
```

It has the benefit of avoiding possible interactions with other TypeScript projects you may have.

Transpile TypeScript into JavaScript
Create a simple TS file. Open VS Code on an empty folder and create a helloworld.ts file, place the following code in that file:

```
let message: string = 'Hello World';
console.log(message);
```

To test that you have the TypeScript compiler installed correctly and a working Hello World program, open a terminal and type

```
tsc helloworld.ts
```

You should now see the transpiled **helloworld.js** JavaScript file, which you can run if you have Node.js installed, by typing

```
node helloworld.js
```

What are modules in TypeScript?
JavaScript has a concept of modules from ECMAScript 2015. TypeScript shares this concept of a module.

A **module** is a way to create a group of related variables, functions, classes, and interfaces, etc. It executes in the local scope, not in the global scope. In other words, the variables, functions, classes, and interfaces declared in a module cannot be accessible outside the module directly. We can create a module by using the **export** keyword and can use it in other modules by using the **import** keyword.

Modules import another module by using a module loader. At runtime, the module loader is responsible for locating and executing all dependencies of a module before executing it. The most common module loaders which are used in JavaScript are the:

1. CommonJS module loader for Node.js
2. require.js for Web applications.

We can divide the module into two categories:

1. **Internal Module**
Internal modules were in the earlier version of Typescript. It was used for logical grouping of the classes, interfaces, functions, variables into a single unit and can be exported in another module. The modules are named as a namespace in the latest version of TypeScript.

Earlier Syntax:

```
module Sum {
   export function add(a, b) {
      console.log("Sum: " +(a+b));
   }
}
```

ECMAScript 2015 Syntax:

```
namespace Sum {
   export function add(a, b) {
      console.log("Sum: " +(a+b));
   }
}
```

2. **External Module**
When the applications consist of hundreds of files, then it is almost impossible to handle these files without a modular approach. External Module is used to specify the load dependencies between the multiple external js

files. If the application has only one js file, the external module is not relevant. ECMAScript 2015(ES6) module system treats every file as a module.

Module declaration

We can declare a module by using the **export** keyword. The syntax for the module declaration is given below:

```
//FileName : EmployeeInterface.ts
export interface Employee {
   //code declarations
}
```

We can use the declare module in other files by using an **import** keyword, which looks like below. The file/module name is specified without an extension.

```
import { class/interface name } from 'module_name';
```

Compiling and Executing Modules

Open the terminal and go to the location where you stored your project. Now, type the following command in the terminal window.

```
$ tsc --module commonjs example.ts
$ node ./example.js
```

Importing multiple modules in single file

We can define multiple modules in a single file. The syntax for multiple module declarations is given below:

```
import { export_name1, export_name2 } from 'module_name';
```

Re-exports

In TypeScript, sometimes modules extend other modules, and partially expose some of their features. A re-export does not import it locally or introduce a local variable. In this case, we can re-export some of their features either using their original name or rename it.

Let us understand the re-export concept of a module with the following example.

Module Creation: **module.ts**

```
export class Addition{
    constructor(private x?: number, private y?: number){
    }
    Sum(){
        console.log("SUM: " +(this.x + this.y));
    }
}
```

Create re-exports module: **re-exports.ts**

```
// Re-exporting types Addition as plus from Modules file
export { Addition as plus } from "./module";
```

In the below example, the name of Addition export class is changed to plus using **{Addition as plus}**. The re-exporting is useful in assigning a more meaningful name to an export which increases the readability of our program.

Accessing the module in another file by using the import keyword: **app.ts**

```
// Importing the exporting types from re-exports file
import {plus as Addition} from './re-exports';  // Importing plus as Addition

let addObject = new Addition(10, 20);

addObject.Sum();
```

Open the terminal and go to the location where you stored your project. Now, type the following command. You will get the following output.

```
PS D:\IT-Semester 6\Web X.0 Lab> tsc --module commonjs app.ts
PS D:\IT-Semester 6\Web X.0 Lab> node ./app.js
SUM: 30
```

Problem Statement:

1. Implement a simple Calculator

Code:

```
// Synchronous Readline for interactively running to have a conversation with the user
via a console(TTY).
import { question } from 'readline-sync';

// Function to perform calculation of user inputs and its validation
function main(): void { // No return type
    const firstString: string = question('Enter first number:\n');
    const operator: string = question('Enter the operator:\n');
    const secondString: string = question('Enter second number:\n');

    // Check whether the user inputs are valid or not
    const validInput: boolean = isNumber(firstString) && isOperator(operator) &&
isNumber(secondString);
```

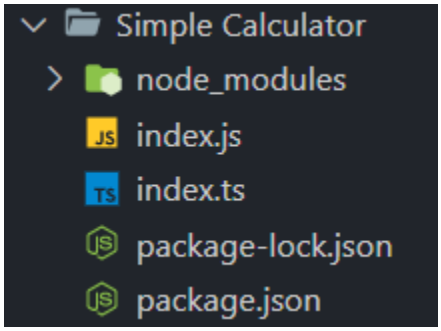
```
    if(validInput) {
        const firstNum: number = parseInt(firstString); // Convert the string to a
number
        const secondNum: number = parseInt(secondString); // Convert the string to a
number
        const result = calculate(firstNum, operator, secondNum); // Perform the
calculation
        console.log("Your answer:");
        console.log(result) // Print the answer
    } else {
        console.log("\nInvalid Input\n"); // Shows the error message
        main(); // Returns the main function for invalid input
    }
}

// Function to check whether the entered number is a number or not
function isNumber(str: string): boolean {
    const strOrNum = parseInt(str); // Will return either a number or NaN
    const isNum: boolean = !isNaN(strOrNum); // Will return true if the number is a
number
    return isNum;
}

// Function to check whether the entered operator is an operator or not
function isOperator(operator: string): boolean {
    switch(operator){ // Switch case to return true if the operator is +, -, *, /
        case '+':
            return true;
        case '-':
            return true;
        case '*':
            return true;
        case '/':
            return true;
        default:
            return false;
    }
}

function calculate(firstNum: number, operator: string, secondNum: number) {
    switch(operator){ // Switch case to return the answer
        case '+':
```

```
            return firstNum + secondNum;
        case '-':
            return firstNum - secondNum;
        case '*':
            return firstNum * secondNum;
        case '/':
            return firstNum / secondNum;
    }
}

// Execute the main function
main();
```

Output:

```
PS D:\IT-Semester 6\Web X.0 Lab\Simple Calculator> tsc index.ts
PS D:\IT-Semester 6\Web X.0 Lab\Simple Calculator> node index.js
Enter first number:
25
Enter the operator:
*
Enter second number:
60
Your answer:
1500
```

File Structure:



2. Create a class with 3 properties with different levels of access modifier and demonstrate its access outside class - Sub class and Non-Subclass

Code:

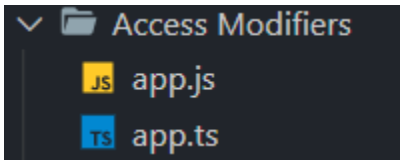
```
class Student {
    public studRollNo: number;  // Public Access Modifier
    protected studName: string;  // Protected Access Modifier
    private studDepartment: string; // Private Access Modifier

    constructor(rollNo: number, name: string, department: string) {
        this.studRollNo = rollNo;
        this.studName = name;
        this.studDepartment = department;
    }

    // Functions shows that Public, Protected and Private access modifier can be
accessed inside the same class
    public showRollNoNameAndDept() {
        return (`Student Roll Number: ${this.studRollNo}\nStudent Name:
${this.studName}\nStudent Department: ${this.studDepartment}\n`);
    }
}

class Person extends Student {
    constructor(rollNo: number, name: string, department: string) {
        super(rollNo, name, department);
    }

    // Functions shows that only Public and Protected access modifier can be accessed
inside the sub/derived class
    public showRollNoAndName() {
        return (`Student Roll Number: ${this.studRollNo}\nStudent Name:
${this.studName}`);
    }
}

// This shows that Public access modifier can be accessed outside the class using an
object of the class
let stud = new Student(58, "Ninad", "IT");
console.log(stud.studRollNo,"\n");

let student: Person = new Person(58, "Ninad", "IT");
console.log(student.showRollNoNameAndDept());
```

```
console.log(student.showRollNoAndName());
```

Output:

```
PS D:\IT-Semester 6\Web X.0 Lab\Access Modifiers> tsc app.ts
PS D:\IT-Semester 6\Web X.0 Lab\Access Modifiers> node app.js
58

Student Roll Number: 58
Student Name: Ninad
Student Department: IT

Student Roll Number: 58
Student Name: Ninad
```

File Structure:



3. Create a Login page. Validate the login with default values.

Code:

```
if (typeof document !== 'undefined') {
    // Will run in client's browser only
    var emailInput: HTMLInputElement =
<HTMLInputElement>document.getElementById("floatingInput"); // User entered Email
    var passwordInput: HTMLInputElement =
<HTMLInputElement>document.getElementById("floatingPassword"); // User entered Password

    // Print the validation message
    var emailValidate: HTMLInputElement =
<HTMLInputElement>document.getElementById("emailShow");
    var passwordValidate: HTMLInputElement =
<HTMLInputElement>document.getElementById("passwordShow");

    // Submit function to submit the user credentials
    function submit() {
        const email: string = emailInput.value; // Value of the Email entered by the
user
        const password: string = passwordInput.value; // Value of the Password entered
```

```
by the user

        // Check whether the user inputs are valid or not
        const validInputEmail: boolean = validEmail(email);
        const validInputPassword: boolean = validPassword(password);

        if (validInputEmail == true && validInputPassword == true) {
            // Show the validation
            console.log("Email is valid\n");
            console.log("Password is valid");

            var emailShow = "You have entered the correct email";
            var passwordShow = "You have entered the correct password";

            // Print the validation
            emailValidate.innerHTML = '<p style="color: green">' + emailShow + '</p>';
            passwordValidate.innerHTML = '<p style="color: green">' + passwordShow +
'</p>';
        } else if (validInputEmail == false && validInputPassword == true) {
            // Show the validation
            console.log("Email is not valid\n");
            console.log("Password is valid");

            var emailShow = "You have entered the incorrect email";
            var passwordShow = "You have entered the correct password";

            // Print the validation
            emailValidate.innerHTML = '<p style="color: red">' + emailShow + '</p>';
            passwordValidate.innerHTML = '<p style="color: green">' + passwordShow +
'</p>';
        } else if (validInputEmail == true && validInputPassword == false) {
            // Show the validation
            console.log("Email is valid\n");
            console.log("Password is not valid");

            var emailShow = "You have entered the correct email";
            var passwordShow = "You have entered the incorrect password";

            // Print the validation
            emailValidate.innerHTML = '<p style="color: green">' + emailShow + '</p>';
            passwordValidate.innerHTML = '<p style="color: red">' + passwordShow +
'</p>';
        } else {
```

```
            // Show the validation
            console.log("Email is not valid\n");
            console.log("Password is not valid");

            var emailShow = "You have entered the incorrect email";
            var passwordShow = "You have entered the incorrect password";

            // Print the validation
            emailValidate.innerHTML = '<p style="color: red">' + emailShow + '</p>';
            passwordValidate.innerHTML = '<p style="color: red">' + passwordShow +
'</p>';
        }
    }

    // Function to validate the email entered by the user with default value
    function validEmail(email: string): boolean {
        const myEmail = "ninadsrao@gmail.com";
        const valid: boolean = myEmail === email;
        return valid;
    }

    // Function to validate the password entered by the user with default value
    function validPassword(password: string): boolean {
        const myPassword = "ninadrao1234";
        const valid: boolean = myPassword === password;
        return valid;
    }
}
```

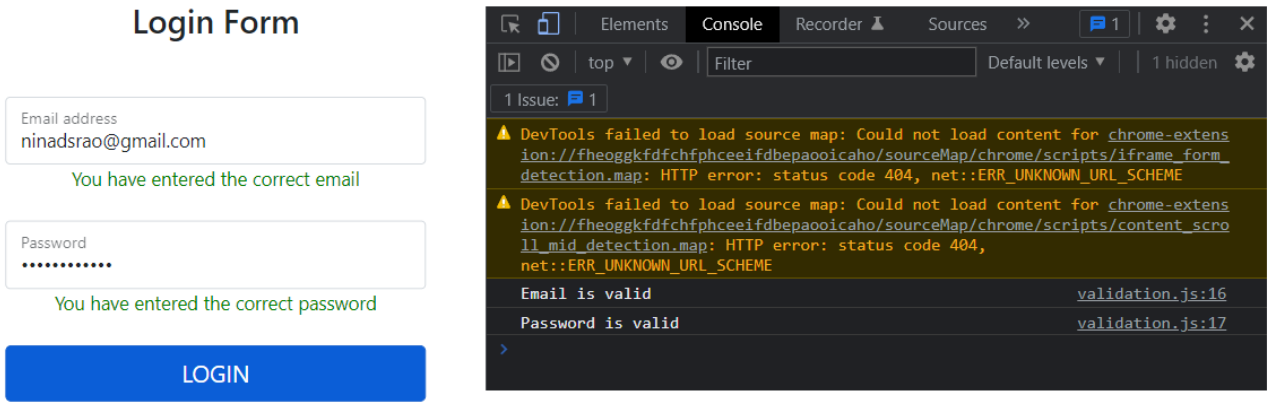
HTML Code

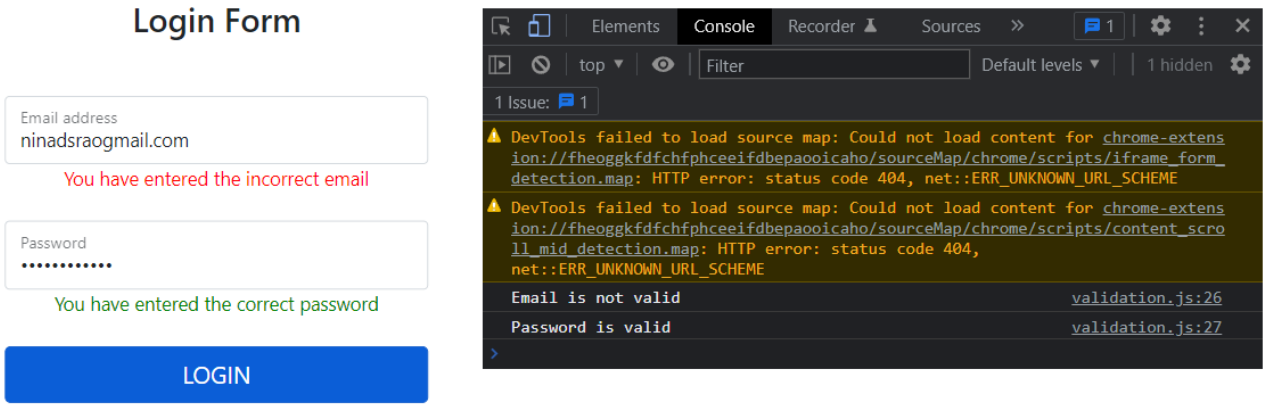
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />

    <!-- Bootstrap CSS -->
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
      rel="stylesheet"
```

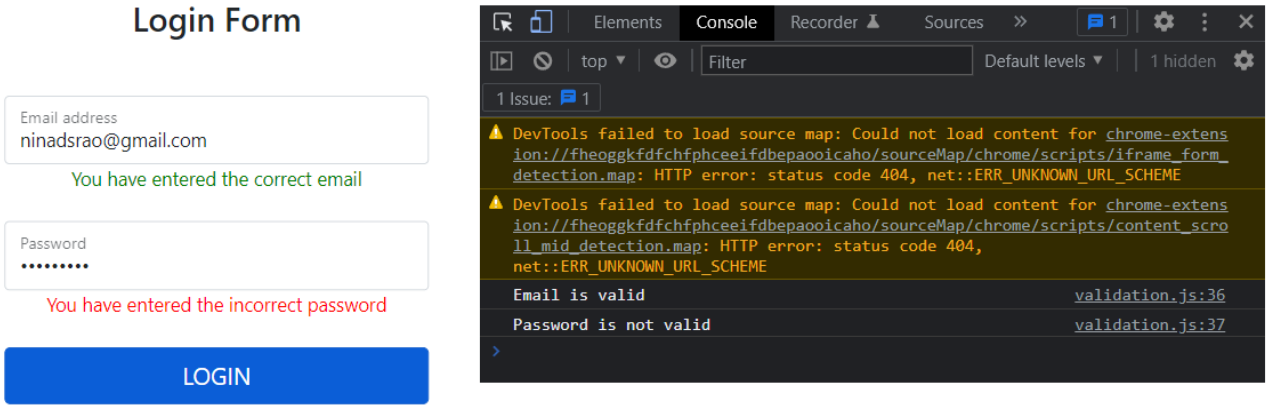
```
integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
      crossorigin="anonymous"
    />

    <title>Login Page</title>
  </head>
  <body>
    <section class="vh-100" style="background-color: #508bfc">
      <div class="container py-5 h-100">
        <div class="row d-flex justify-content-center align-items-center">
          <div class="col-12 col-md-8 col-lg-6 col-xl-5">
            <div class="card shadow-2-strong" style="border-radius: 1rem">
              <div class="card-body p-5 text-center">
                <h3 class="mb-5">Login Form</h3>
                <div class="form-floating mb-4">
                  <input
                    type="email"
                    class="form-control"
                    id="floatingInput"
                    placeholder="name@example.com"
                    style="box-shadow: none"
                    required
                  />
                  <label for="floatingInput">Email address</label>
                  <div id="emailShow"></div>
                </div>
                <div class="form-floating mb-4">
                  <input
                    type="password"
                    class="form-control"
                    id="floatingPassword"
                    placeholder="Password"
                    style="box-shadow: none"
                    required
                  />
                  <label for="floatingPassword">Password</label>
                  <div id="passwordShow"></div>
                </div>
                <div class="d-grid gap-2">
                  <input
                    class="btn btn-primary btn-lg btn-block"
                    type="button"
                    id="submitButton"
```

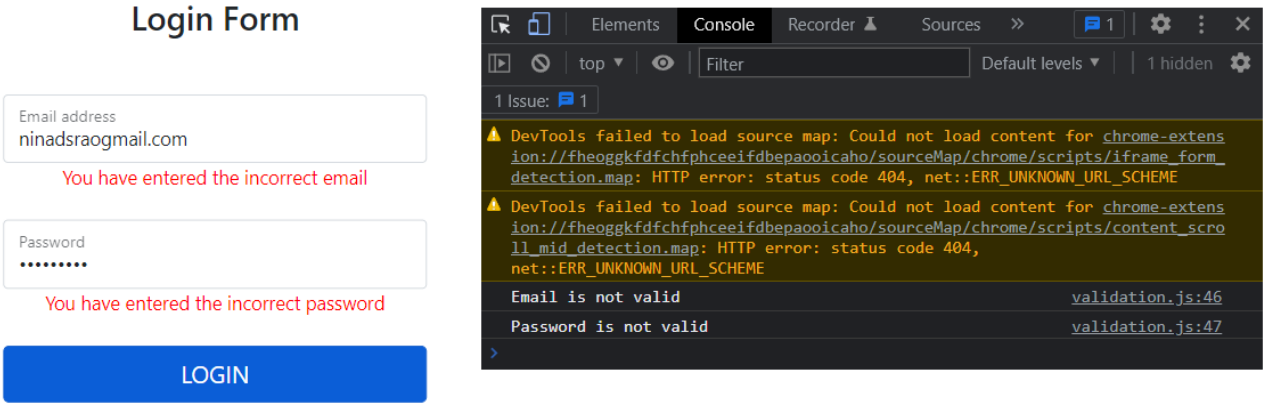
```
                    onclick="submit()"
                    value="LOGIN"
                    style="box-shadow: none"
                  >
                  </input>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </section>
    <script src="./validation.js"></script>
  </body>
</html>
```

Output:









Conclusion: Hence, we understood to study basic Constructs, Inheritance, Access modifiers and Creation of Web Pages in TypeScript.