

Diploma Engineering

Laboratory Manual

(Modern Practical Tools)

(4340705)

[Computer Engineering with 4th semester]

Enrolment No	
Name	
Branch	
Academic Term	
Institute	



Directorate Of Technical Education
Gandhinagar - Gujarat

DTE's Vision:

- To provide globally competitive technical education;
- Remove geographical imbalances and inconsistencies;
- Develop student friendly resources with a special focus on girls' education and support to weaker sections;
- Develop programs relevant to industry and create a vibrant pool of technical professionals.

DTE's Mission:

Institute's Vision:

Student should write

Institute's Mission:

Student should write

Department's Vision:

Student should write

Department's Mission:

Student should write

Certificate

Government Polytechnic, Ahmedabad

Certificate

This is to certify that
Mr./Ms.....
Enrollment No. of Semester
of.....of.....Institute
..... (GTU Code:....) has completed the term work
satisfactorily in Subject Modern Practical Tools – 4340705 for the
academic year:Term:as prescribed in the curriculum.

Place:

Date:

Subject Faculty

Head of the Department

Preface

The primary aim of any laboratory/Practical/field work is enhancement of required skills as well as creative ability amongst students to solve real time problems by developing relevant competencies in psychomotor domain. Keeping in view, GTU has designed competency focused outcome-based curriculum -2021 (COGC-2021) for Diploma engineering programmes. In this more time is allotted to practical work than theory. It shows importance of enhancement of skills amongst students and it pays attention to utilize every second of time allotted for practical amongst Students, Instructors and Lecturers to achieve relevant outcomes by performing rather than writing practice in study type. It is essential for effective implementation of competency focused outcome- based Green curriculum-2021. Every practical has been keenly designed to serve as a tool to develop & enhance relevant industry needed competency in each and every student. These psychomotor skills are very difficult to develop through traditional chalk and board content delivery method in the classroom. Accordingly, this lab manual has been designed to focus on the industry defined relevant outcomes, rather than old practice of conducting practical to prove concept and theory.

By using this lab manual, students can read procedure one day in advance to actual performance day of practical experiment which generates interest and also, they can have idea of judgement of magnitude prior to performance. This in turn enhances predetermined outcomes amongst students. Each and every Experiment /Practical in this manual begins by competency, industry relevant skills, course outcomes as well as practical outcomes which serve as a key role for doing the practical. The students will also have a clear idea of safety and necessary precautions to be taken while performing experiment.

This manual also provides guidelines to lecturers to facilitate student-centered lab activities for each practical/experiment by arranging and managing necessary resources in order that the students follow the procedures with required safety and necessary precautions to achieve outcomes. It also gives an idea that how students will be assessed by providing Rubrics.

Today's modern industry uses many frameworks for a front-end web design and Angular is one of them for developing dynamic web applications. It covers all the basics of frontend web application development using the Angular framework in order to provide developers insights into real-world challenges and scenarios they face throughout their day-to-day development process, as well as provides tips and best practices for becoming a web application developer.

Below are major reasons for considering learning Angular over other platforms.

- Optimal code
- Easy to Integrate

- Support for Single Page Applications
- Declarative User Interface
- Modularity
- Cross-platform compatibility

This course will give basic knowledge and skills for front-end design for web application development using Angular framework.

Although we try our level best to design this lab manual, but always there are chances of improvement. We welcome any suggestions for improvement.

Programme Outcomes (POs) :

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.
2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.
3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.
4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.
5. **Engineering practices for society, sustainability and environment:** Apply appropriate technology in context of society, sustainability, environment and ethical practices.
6. **Project Management:** Use engineering management principles individually, as a team member or a leader to manage projects and effectively communicate about well-defined engineering activities.
7. **Life-long learning:** Ability to analyze individual needs and engage in updating in the context of technological changes *in field of engineering*.

Practical Outcome - Course Outcome matrix

Course Outcomes (COs):

a. CO1: Prepare environment for angular project using Node.js, npm and visual code editor.

b. CO2: Apply angular directives, components and pipes in different web page development.

c. CO3: Utilize angular template driven and reactive forms in different problem solutions.

d. CO4: Design pages to make HTTP GET/POST calls to perform CRUD operations using different server-side APIs.

e. CO5: Develop single page dynamic applications using Angular framework and APIs.

S. No.	Practical Outcome/Title of experiment	CO 1	CO 2	CO 3	CO 4	CO 5
1.	Setup environment for Angular framework by Installing Node.js, npm package manager using editor like Visual Code.	√	-	-	-	-
2.	Create first application to print Hello World message using angular framework.	√	-	-	-	-
3.	Design a web page to utilize property binding and event binding concepts using button and textbox controls.	√	-	-	-	-
4.	Create various components of web page using Attribute Directives.		√			
5.	Design a web page to display student grading system in tabular format with alternate color style using ngSwitch, ngStyleDirectives.	-	√	-	-	-
6.	Design component to perform following tasks A. To Add or Remove number of students using textbox and button controls and display it in tabular structure format.	-	√	-	-	-

	B. Give row level remove button option to student table and record should be deleted when click on it.					
7	Create a component to display a products list from array. the product component should display a product Id, name, purchase date, price, and image for the product and search using various pipes.	-	√	-	-	-
8	Design a student registration page using template driven form approach and utilize different form and controls level ng validation classes.	-	-	√	-	-
9	Design component to enter faculty details like Code, Name, Email, Type, Faculty Status (Active, Inactive), Subjects Teaching (with option to add multiple subjects dynamically) using reactive form with various types of validation of form and controls.	-	-	√	-	-
10.	Design a page to implement Add to Cart functionality using decorators, custom properties, custom events of component communication.	-	-	√	-	-
11.	Develop page to demonstrate different methods of angular component lifecycle.	-	-	√	-	-
12.	Design an e-commerce product page and product details page that displays product details when clicking on any particular product.	-	-	√	-	-
13.	Design a page to display student information using dependency Injection.	-	-	-	√	-
14.	Develop a page for product listing and search-as-you-type using observables and web APIs from database.	-	-	-	√	-
15.	Design web page to display student data in table using HTTP GET/POST Calls from web APIs.	-	-	-	√	-

16.	Design web page to insert product data in table using web APIs.	-	-	-	√	-
17.	Design a page to implement Multiview component with login, logout functionalities using different routing options.	-	-	-	-	√
18.	Develop a page to demonstrate page navigation of product list using routing concepts.	-	-	-	-	√
19.	Design a page to load customer and Sales order data using lazy loading technique in angular.	-	-	-	-	√
20.	Design a page to implement CORS concept.		-	-	-	√

Industry Relevant Skills

The following industry relevant skills are expected to be developed in the students by performance of experiments of this course.

- **Use Angular Framework to build appealing dynamic web application for all platforms.**

Guidelines to Course Faculty

1. Course faculty should demonstrate experiment with all necessary implementation strategies described in curriculum.
2. Course faculty should explain industrial relevance before starting of each experiment.
3. Course faculty should involve & give opportunity to all students for hands on experience.
4. Course faculty should ensure mentioned skills are developed in the students by asking.
5. Utilise 2 hrs of lab hours effectively and ensure completion of write up with quiz also.
6. Encourage peer to peer learning by doing same experiment through fast learners.

Instructions for Students

1. Organize the work in the group and make record of all observations.
2. Students shall develop maintenance skill as expected by industries.
3. Student shall attempt to develop related hand-on skills and build confidence.
4. Student shall develop the habits of evolving more ideas, innovations, skills etc.

5. Student shall refer technical magazines and data books.
6. Student should develop habit to submit the practical on date and time.
7. Student should well prepare while submitting write-up of exercise.

Continuous Assessment Sheet

Enrolment No:

Name

Name:

Term:

Date :

SrNo	Practical Outcome/Title of experiment	Page	Date	Marks (25)	Sign
1	Setup environment for Angular framework by Installing Node.js, npm package manager using editor like Visual Code.				
2	Create first application to print Hello World message using angular framework.				
3	Design a web page to utilize property binding and event binding concepts using button and textbox controls.				
4	Create various components of web page using Attribute Directives.				
5	Design a web page to display student grading system in tabular format with alternate color style using ngSwitch, ngStyleDirectives.				
6	Design component to perform following tasks				

	<p>A. To Add or Remove number of students using textbox and button controls and display it in tabular structure format.</p> <p>B. Give row level remove button option to student table and record should be deleted when click on it.</p>				
7	Create a component to display a products list from array. Theproduct component should display a product Id, name, purchase date, price, and image for the product and search using various pipes.				
8	Design a student registration page using template driven form approach and utilize different form and controls level ng validation classes.				
9	Design component to enter faculty details like Code, Name, Email, Type, Faculty Status (Active, Inactive), Subjects Teaching (with option to add multiple subjects dynamically) using reactive form with various types of validation of form and controls.				
10	Design a page to implement Add to Cart functionality using decorators, custom properties, custom events of component communication.				
11	Develop page to demonstrate different methods of angular component lifecycle.				
12	Design an e-commerce product page and product details page that displays product details when clicking on any particular product.				
13	Design a page to display student information using dependency Injection.				
14	Develop a page for product listing and search-as-you-type using observables and web APIs from database.				

15	Design web page to display student data in table using HTTP GET/POST Calls from web APIs.				
16	Design web page to insert product data in table using web APIs.				
17	Design a page to implement Multiview component with login, logout functionalities using different routing options.				
18	Develop a page to demonstrate page navigation of product list using routing concepts.				
19	Design a page to load customer and Sales order data using lazy loading technique in angular.				
20	Design a page to implement CORS concept.				

Practical No.1: Setup environment for Angular framework by Installing Node.js, npm package manager using editor like Visual Code.

A. Objective: Setting up environment for first time execution for angular practicals is important task before we start our angular practical's programming. It consist of many installations like visual code editor, Node.js and npm package manager. Once it is installed we are ready to use our setup environment to make project in angular.

B. Expected Program Outcomes (POs)

- Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the digital electronics engineering problems.
- Design/ development of solutions:** Design solutions for digital electronics engineering well-defined technical problems and assist with the design of systems components or processes to meet specified needs.

3. **Engineering Tools, Experimentation and Testing:** Apply modern digital electronics engineering tools and appropriate technique to conduct standard tests and measurements.

C. Expected Skills to be developed based on competency:

1. Install visual code open source software.
2. Setup environment for angular practical execution using node js and npm package manager.

D. Expected Course Outcomes(Cos)

Prepare environment for angular project using Node.js, npm and visual code editor.

E. Practical Outcome(PRo)

Setup environment for angular practical execution with NODE JS and NPM Package manager.

F. Expected Affective domain Outcome(ADos)

1. Follow Coding standards and practices.
2. Maintain tools and equipment.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory:

Angular is basically is an open-source, JavaScript-based client-side framework that helps us to develop a web-based application. Actually, Angular is one of the best frameworks for developing any Single Page Application or SPA Applications.

Angular is a UI framework for building mobile and desktop web applications. It is built using javascript framework for front-end development. It uses Typescript language (Superset of javascript) to make angular application. You can build amazing client-side applications using HTML, CSS, and Typescript using Angular.

It is maintained by Google and a community of experts acting as a solution for rapid front-end development.

Learner can have following roles after learning Angular in a company.

- Web developer
- Web app developer
- UI developer
- UX developer

- Front-end developer
- JavaScript developer

Now, let's start process of Environment setup to install angular framework. Following tools/packages are required to run angular project.

- Nodejs
- Npm
- Angular CLI
- IDE for writing your code

Here are the steps needs to be followed to Environment setup in angular.

Step1: Install Node.js and npm:

Node.js is tool to run the development server for an Angular application. This allows developers to make changes to the code and see the updates in real-time without having to manually reload. It can be used as a build tool to automate tasks such as compiling TypeScript to JavaScript, bundling the application code, and optimizing the code for production.

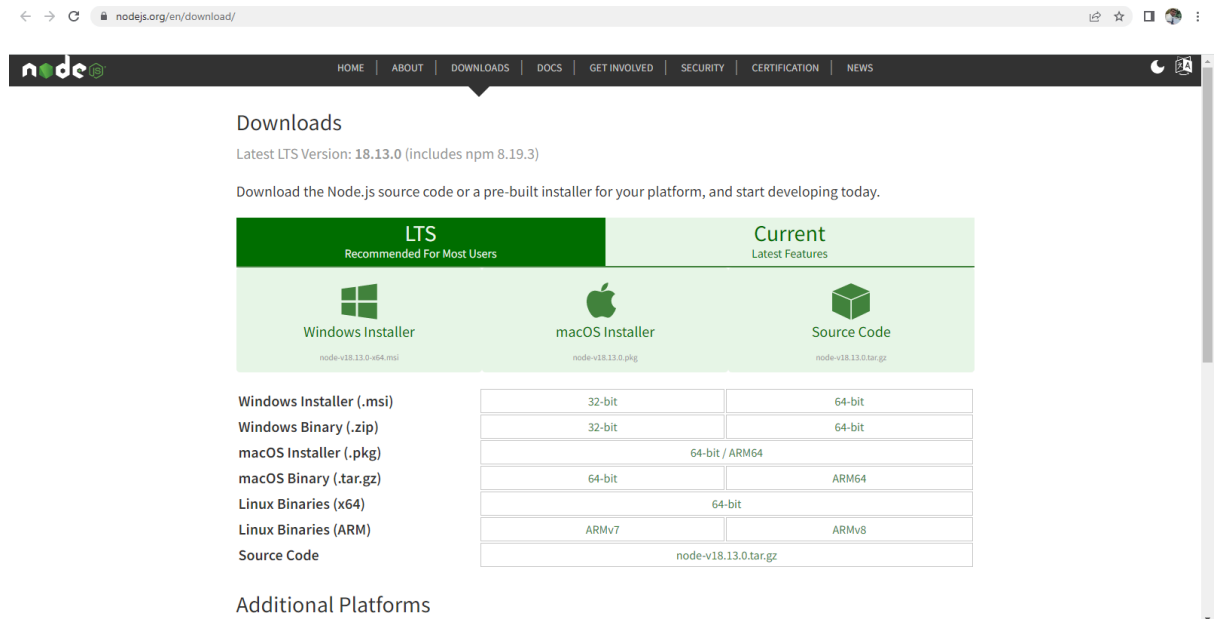
Node.js can also be used to create a backend API that an Angular application can consume. Overall, Node.js can greatly enhance the development and deployment process for Angular applications.

Npm stands for Node Package Manager, which is a package manager for the Node.js runtime environment. npm is used to manage packages and dependencies for Node.js applications, including Angular applications. In an Angular application, npm is used to install and manage packages that the application depends on, such as Angular itself, third-party libraries, and development tools. These packages are typically stored in the "node_modules" directory of the application.

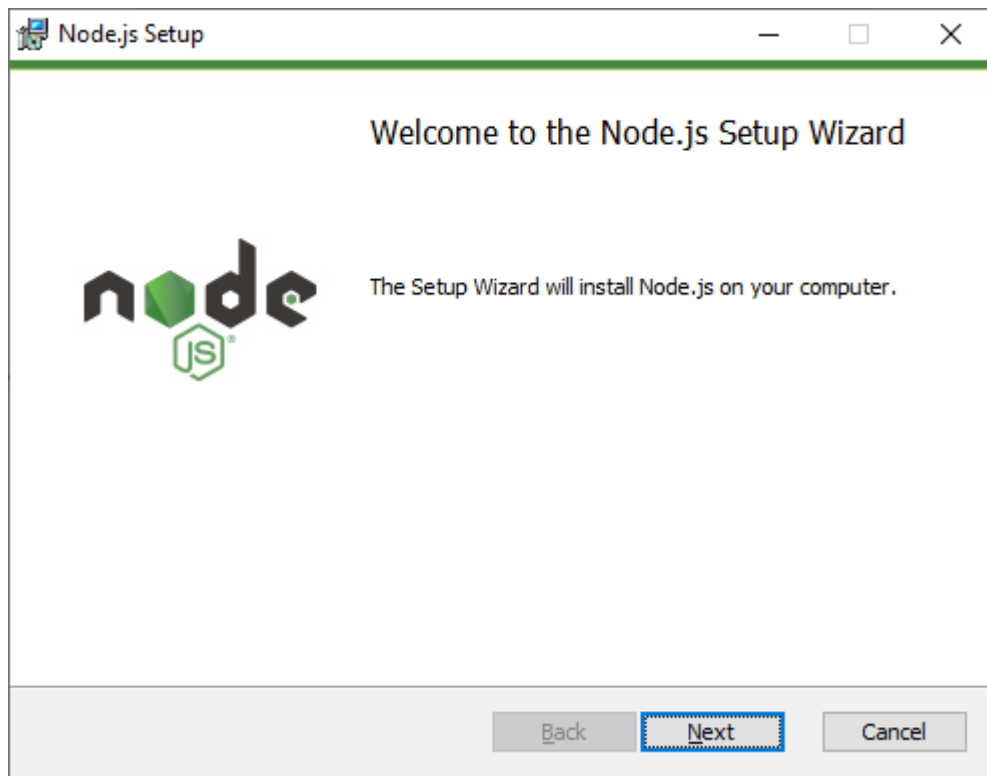
npm provides a command-line interface that allows developers to install, update, and remove packages, as well as manage package versions and dependencies.

Overall, npm plays a critical role in the development and management of Angular applications, making it easier for developers to build and maintain their applications and collaborate with others in the community.

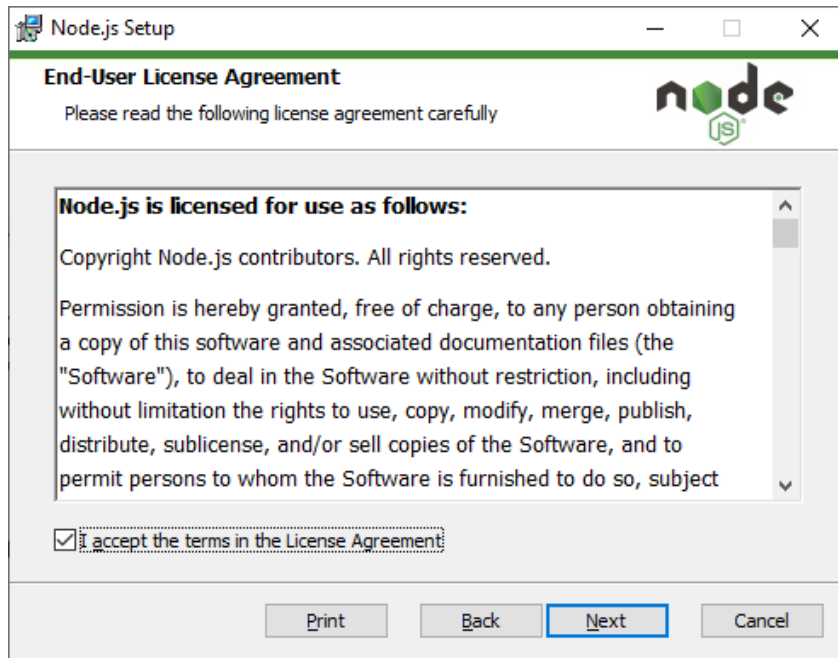
To install Node.js, Go to <https://nodejs.org/en/> and download the latest version of Node.js that corresponds to your operating system.



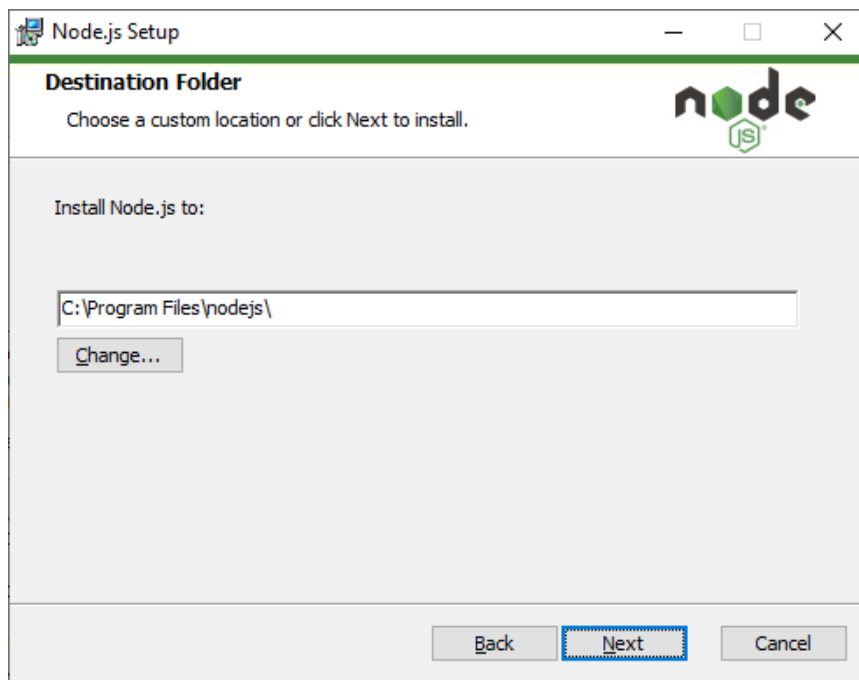
After download nodejs, Run the downloaded installer file and Follow the installation wizard to install Node.js on your computer.



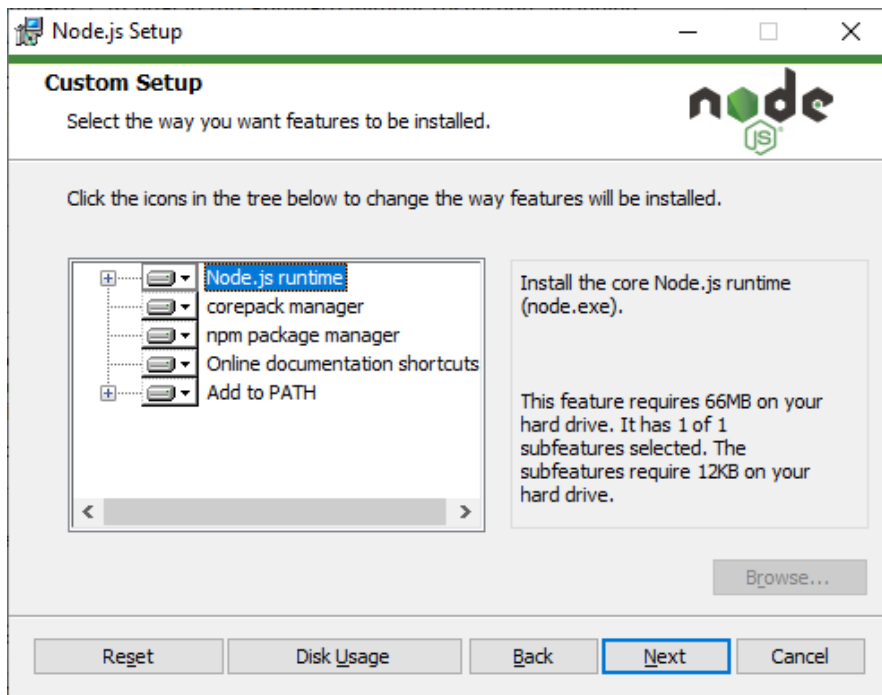
When click on next button, it will ask to accept End-User Licence Agreement as shown below.



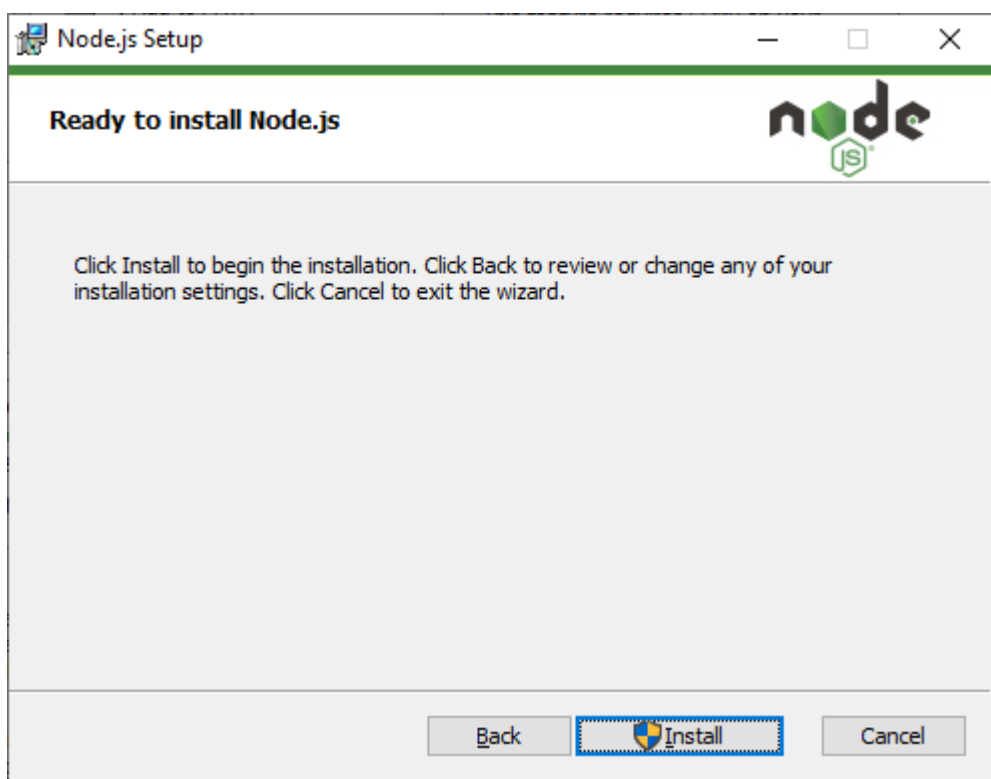
Once you check in checkbox and click on Next Button, you will get prompt box for node.js installation directory. Here you can see default location path as shown in below however you can customize your location by click on change button option.



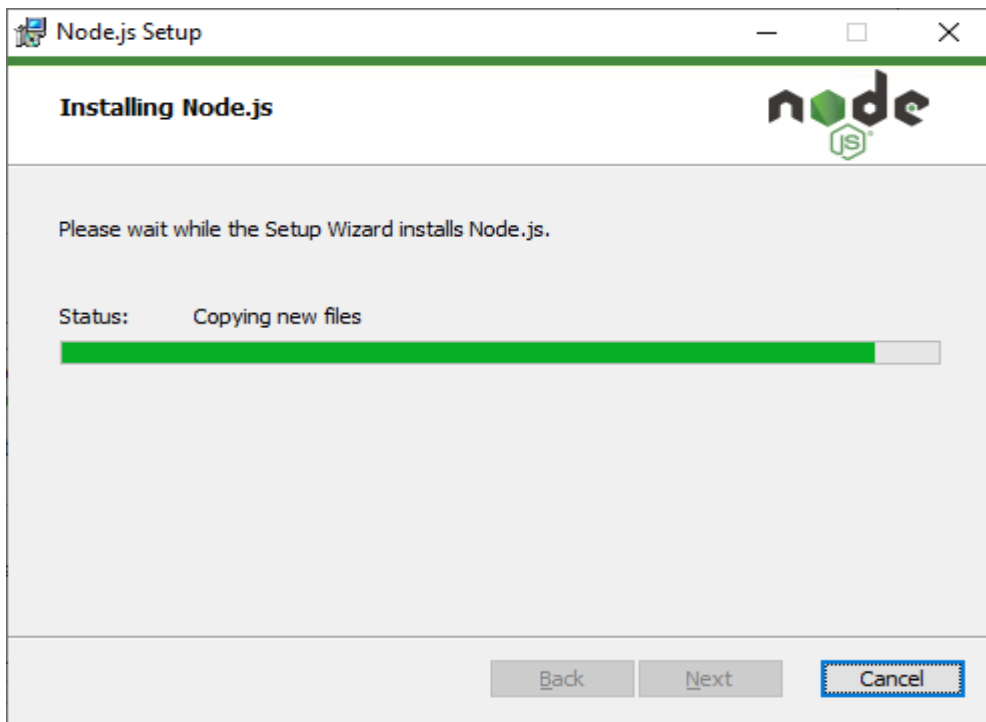
After choose directory, you can select features that you want to customize as shown in below.



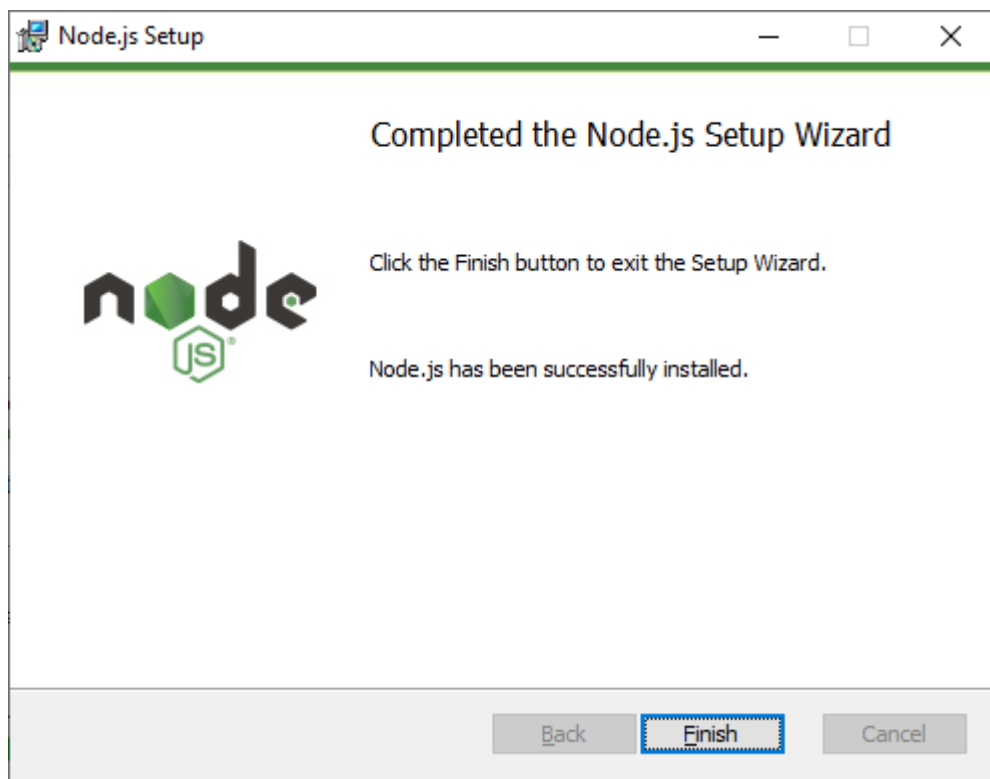
Here npm package manager is also installed automatically as you can see from above figure. After select custom features and click on next button, Node.js Setup wizard ready to install node.js and npm as shown below.



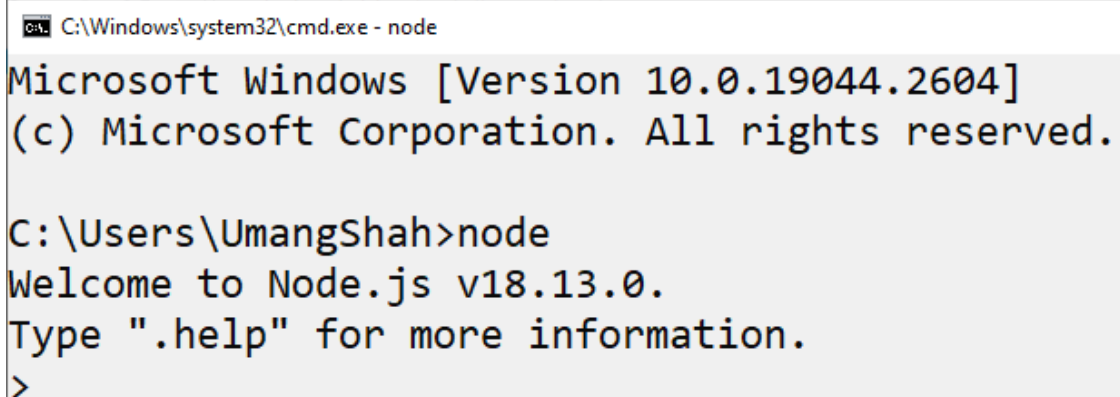
When you click on Install button, wizard will start for installation as shown in below. Wait for the installation to complete. This may take a few minutes.



Once the installation is complete, click "Finish" to close the installer as shown below.



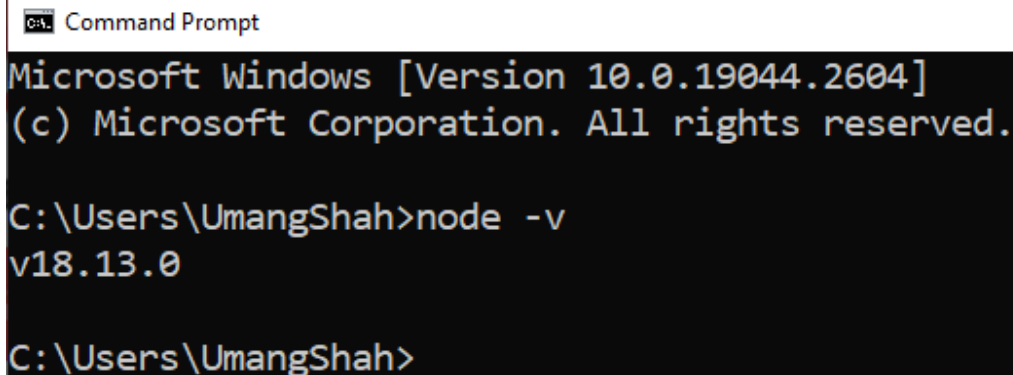
On successful installation, the command prompt will be displayed like this:



```
C:\Windows\system32\cmd.exe - node
Microsoft Windows [Version 10.0.19044.2604]
(c) Microsoft Corporation. All rights reserved.

C:\Users\UmangShah>node
Welcome to Node.js v18.13.0.
Type ".help" for more information.
>
```

You can Verify that Node.js has been installed correctly by opening a command prompt and typing "node -v" This should display the version of Node.js that you have installed as shown below.

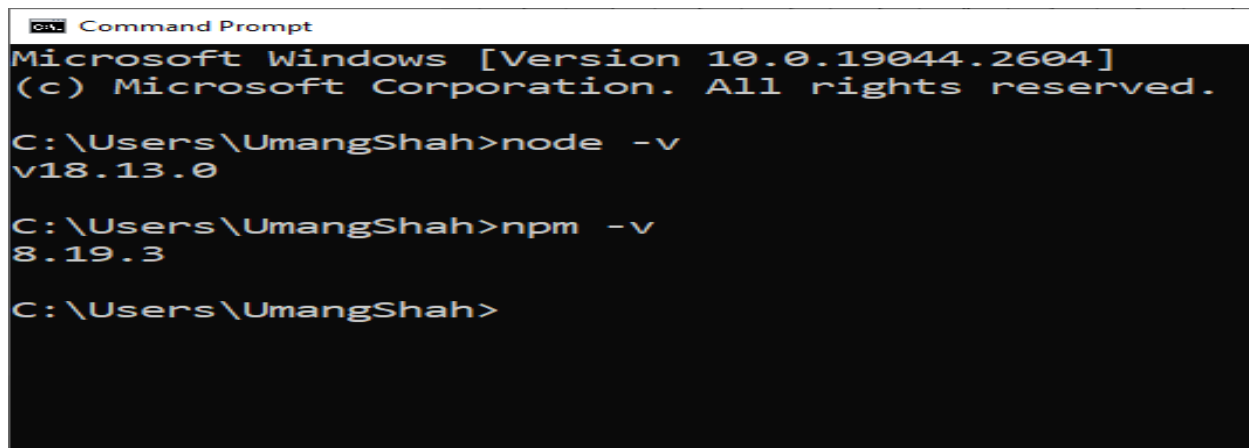


```
Command Prompt
Microsoft Windows [Version 10.0.19044.2604]
(c) Microsoft Corporation. All rights reserved.

C:\Users\UmangShah>node -v
v18.13.0

C:\Users\UmangShah>
```

You can verify that npm (Node Package Manager) has been installed correctly by opening a command prompt and typing "npm -v". This should display the version of npm that you have installed as shown below.



```
Command Prompt
Microsoft Windows [Version 10.0.19044.2604]
(c) Microsoft Corporation. All rights reserved.

C:\Users\UmangShah>node -v
v18.13.0

C:\Users\UmangShah>npm -v
8.19.3

C:\Users\UmangShah>
```

Step2: Install Angular CLI:

Angular CLI is used to create, build, and manage your Angular projects.

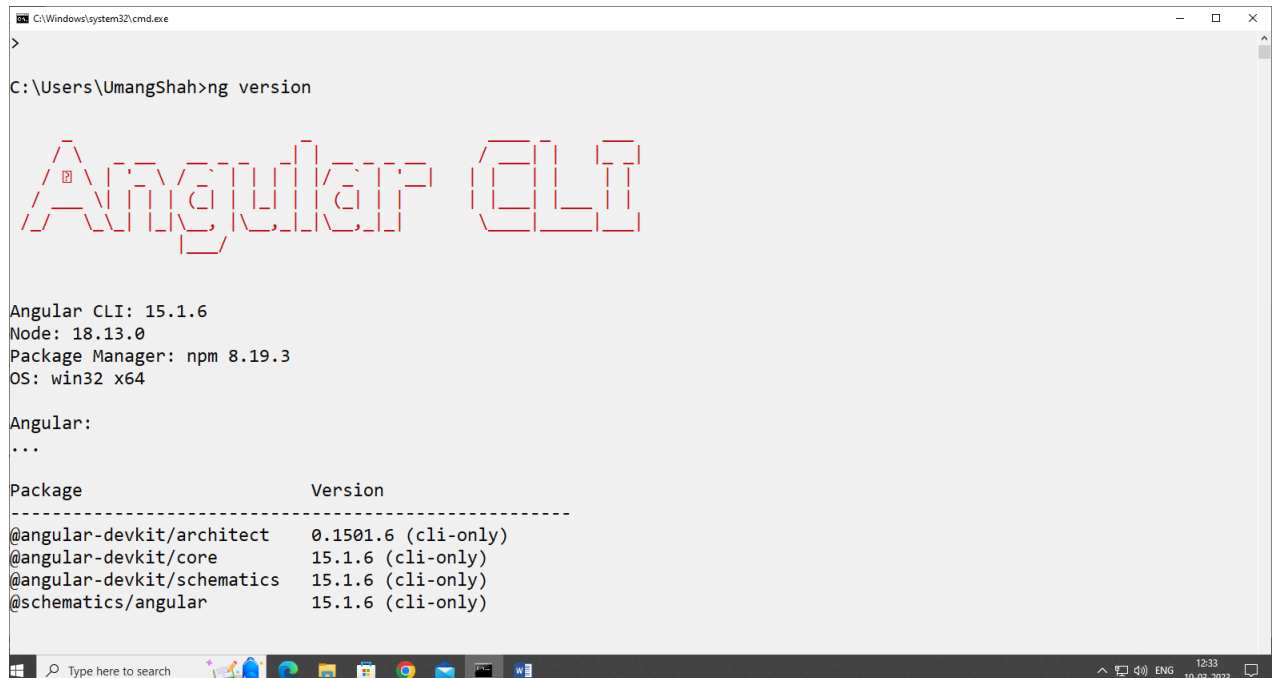
Once Node.js is installed, open a command prompt or terminal window and run the following command to install the Angular CLI

```
npm install -g @angular/cli
```

Wait for the installation to complete. This may take a few minutes depending on your internet speed. Once the installation is complete, run the following command to verify that the Angular CLI has been installed correctly.

```
ng version
```

You should see the version number of the Angular CLI displayed in the console output as given below.



```
C:\Windows\system32\cmd.exe
>
C:\Users\UmangShah>ng version

Angular CLI
Angular CLI: 15.1.6
Node: 18.13.0
Package Manager: npm 8.19.3
OS: win32 x64

Angular:
...

Package                                Version
-----
@angular-devkit/architect              0.1501.6 (cli-only)
@angular-devkit/core                   15.1.6 (cli-only)
@angular-devkit/schematics             15.1.6 (cli-only)
@schematics/angular                   15.1.6 (cli-only)
```

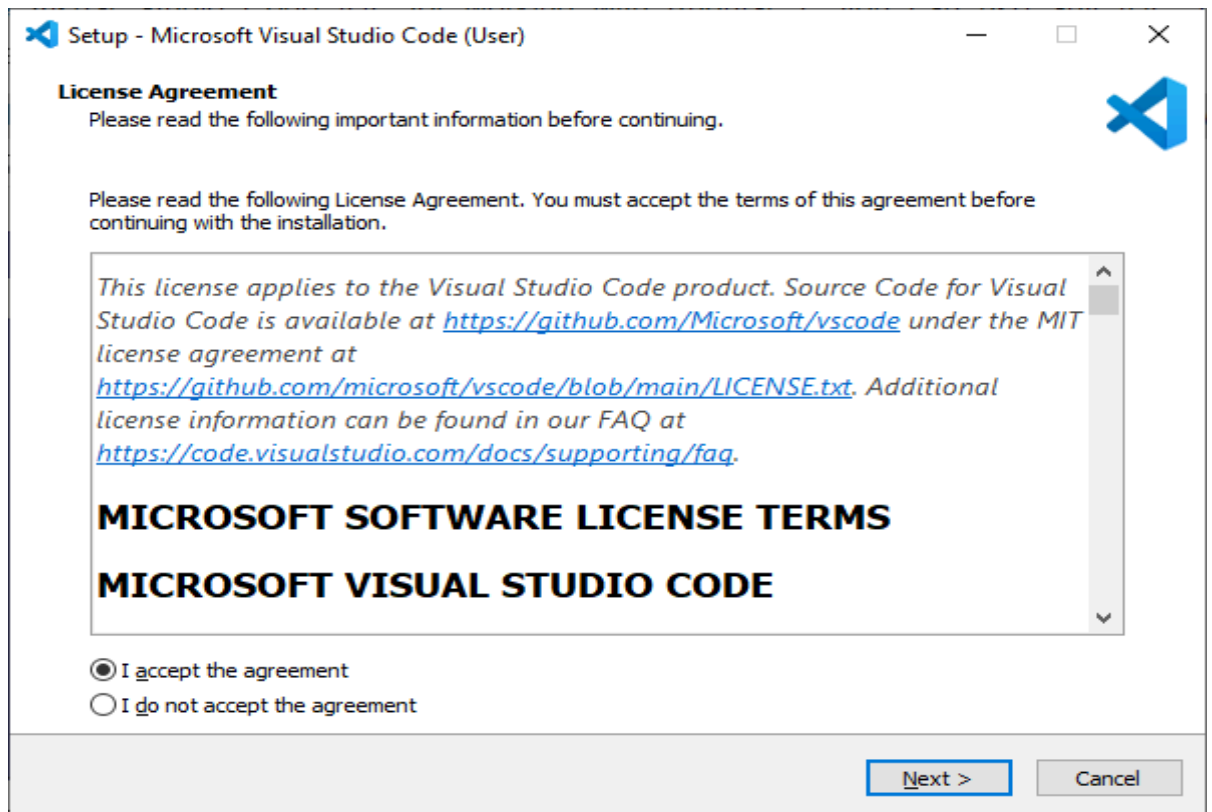
You have successfully installed the Angular CLI on your system. It gives the version for Angular CLI, typescript version and other packages available for Angular.

We are done with the installation of Angular.

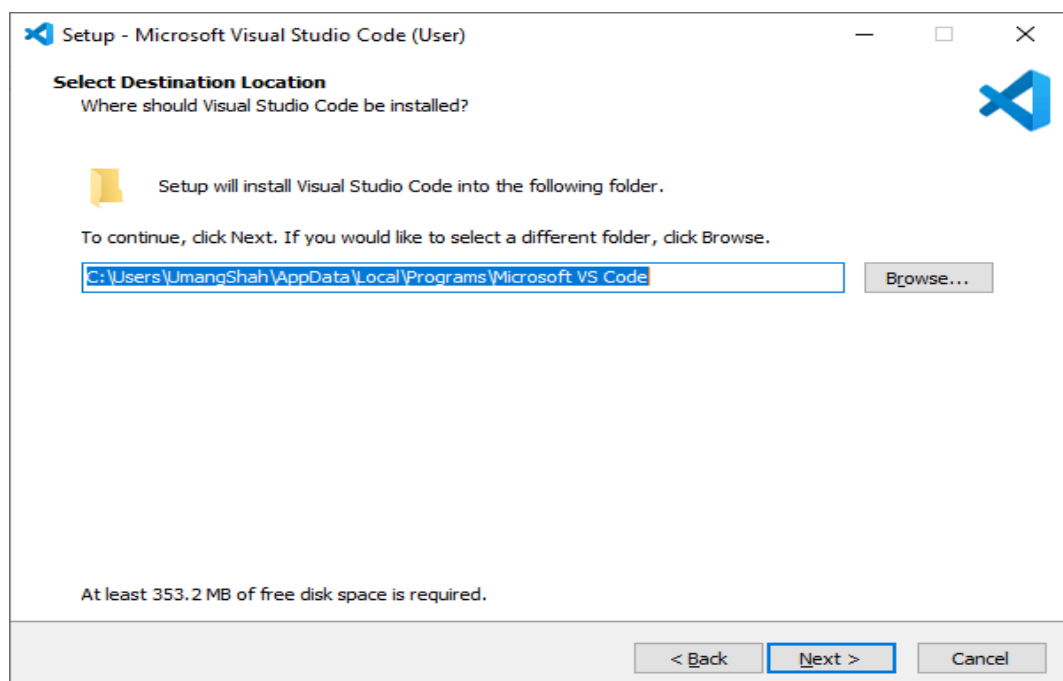
Step3: Install Visual Studio Code Editor:

Following steps are needed to perform for installing Visual Studio Code.

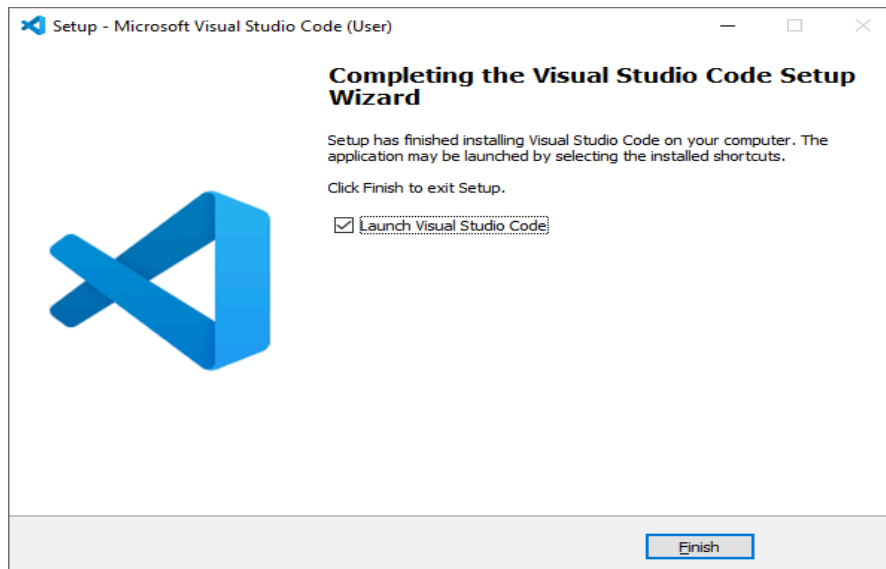
1. Go to the official Visual Studio Code website at <https://code.visualstudio.com/>
2. Click on the "Download for Windows" button to download the Windows version of Visual Studio Code.



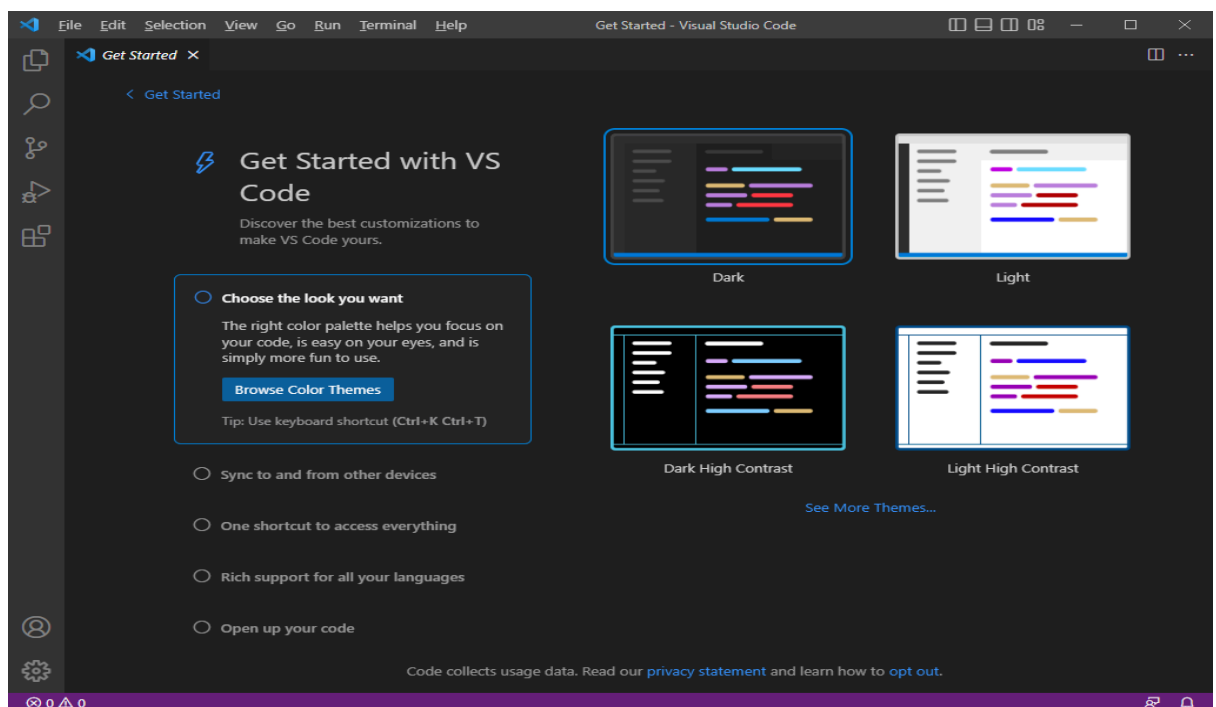
3. Run the downloaded installer.
4. Click on the "Next" button to begin the installation process.
5. Choose the installation location and click on the "Next" button.



6. Choose the start menu folder and click on the "Next" button.
7. Choose the additional tasks you want the installer to perform and click on the "Next" button.
8. Click on the "Install" button to start the installation process.
9. Wait for the installation process to complete.



10. Once the installation is complete, click on the "Finish" button to exit the installer.
11. Launch Visual Studio Code by double-clicking on the desktop icon or by searching for "Visual Studio Code" in the Windows Start menu.



You have successfully installed Visual Studio Code on your Windows computer for manage Angular application.

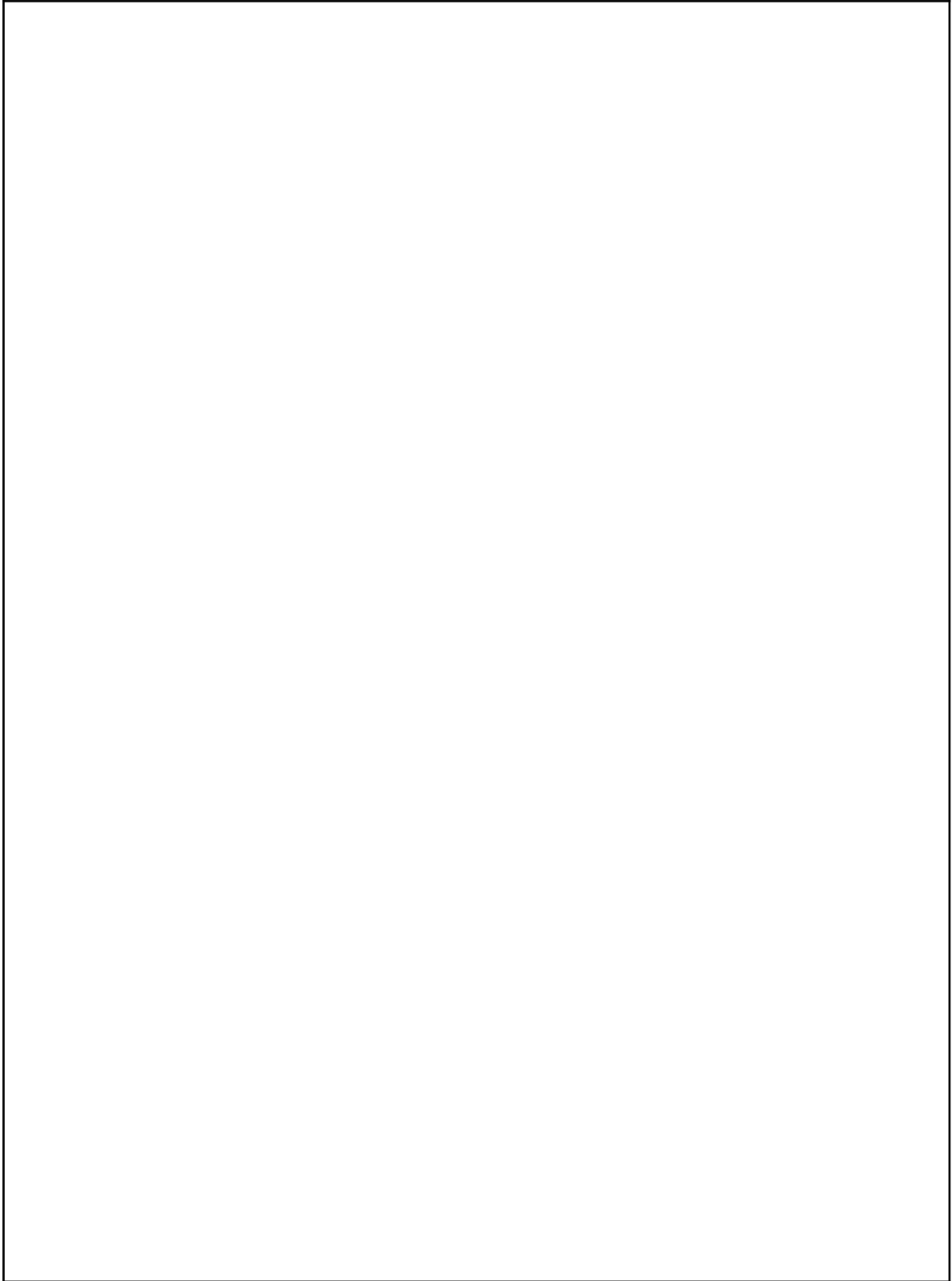
H. Resources/Equipment Required

S r. N o.	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

I. Output Source code:

1. Snapshot of Node.Js successfully installed.

OUTPUT



2. Snapshot of Angular/CLI installed successfully.

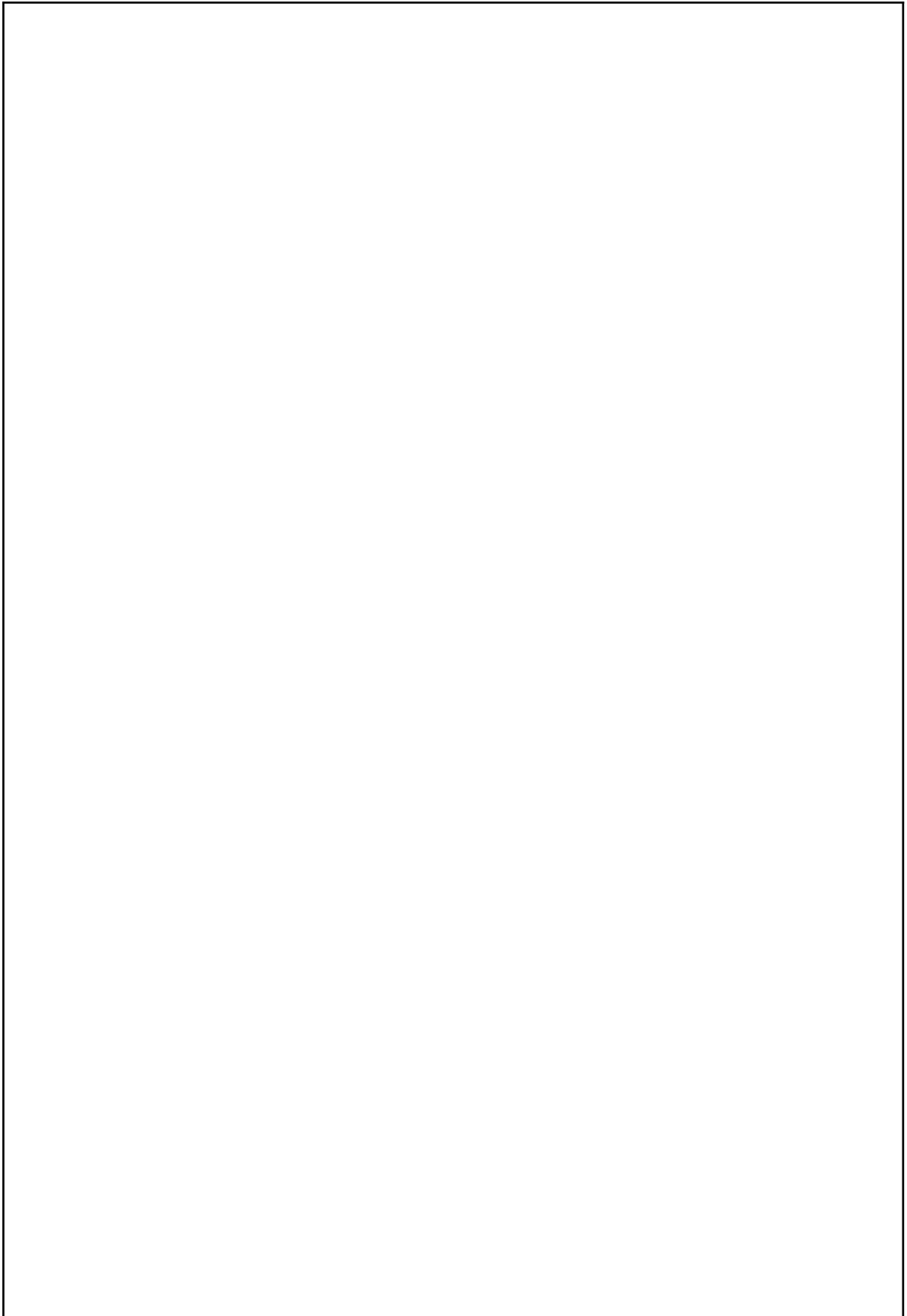
OUTPUT

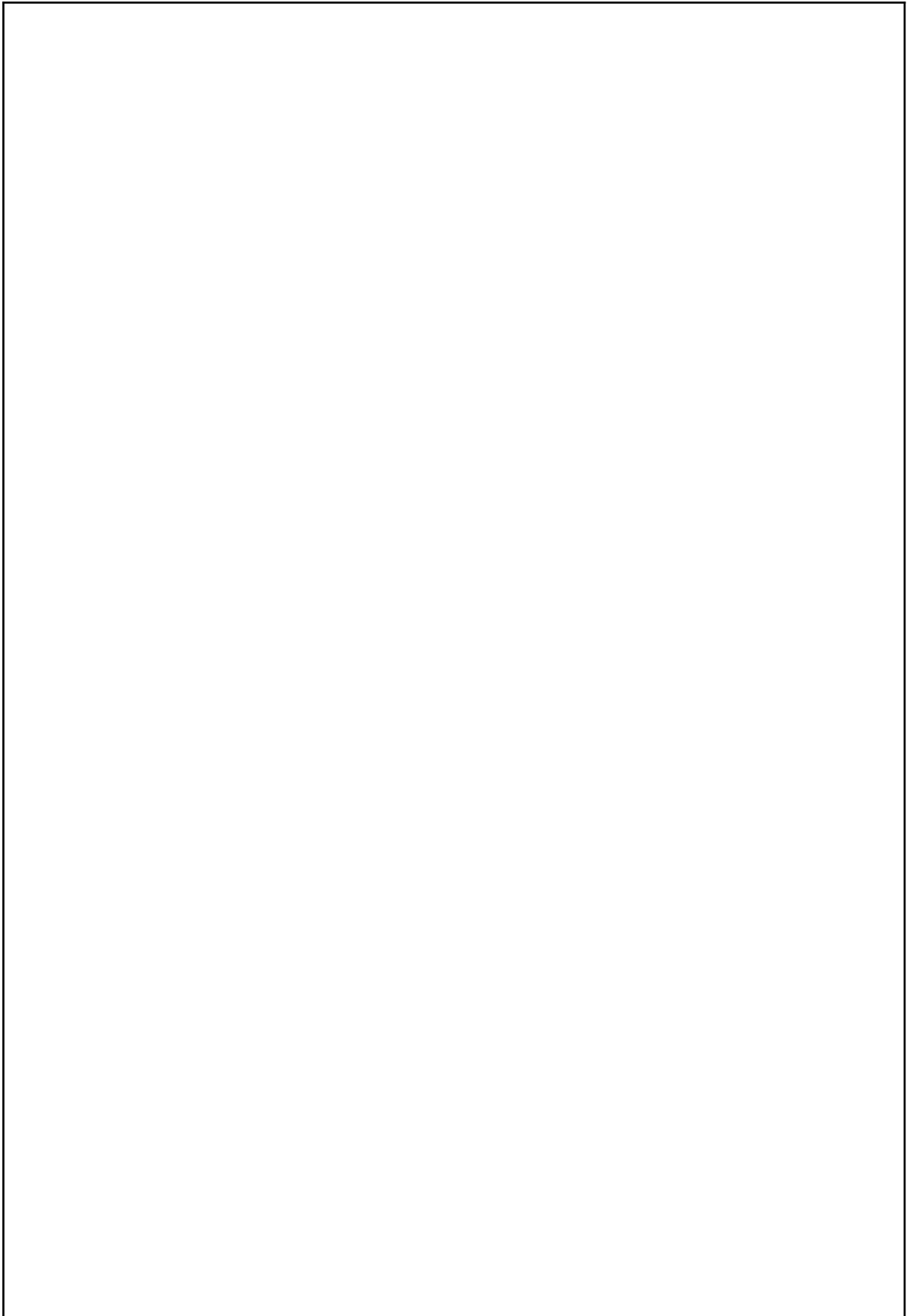
3. Snapshot of Visual Studio Code successfully installed.

OUTPUT

J. Practical related Questions.

1. Why Angular is more popular than other framework?
2. Describe NPM in detail.
3. How do you verify that Node.js is installed correctly on your system?
4. How do you verify that the Angular CLI is installed correctly on your system?





K. References Links

1. <https://angular.io/guide/setup-local>
2. <https://www.geeksforgeeks.org/angular-cli-angular-project-setup/>
3. <https://www.c-sharpcorner.com/article/how-to-install-angular/>
4. <https://www.w3schools.blog/install-angular->
5. https://www.tutorialspoint.com/angular7/angular7_environment_setup.htm

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of Input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No.2: Create first application to print Hello World message using angular framework.

A. Objective:

The objective of this program is a simple that is starting point for learning a new programming language or framework that is to display a simple message or output on the screen to demonstrate the basic functionality of the framework.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the digital electronics engineering problems.
2. **Design/ development of solutions:** Design solutions for digital electronics engineering well-defined technical problems and assist with the design of systems components or processes to meet specified needs.
3. **Engineering Tools, Experimentation and Testing:** Apply modern digital electronics engineering tools and appropriate technique to conduct standard tests and measurements.

C. Expected Skills to be developed based on competency:

1. Project Environment setup for run application in Angular framework.
2. Display Data on web application using Angular framework.

D. Expected Course Outcomes(Cos)

Setup environment for angular practical execution with NODE JS and NPM Package manager.

E. Practical Outcome(PRo)

Create first application to print Hello World message using angular framework.

F. Expected Affective domain Outcome(ADos)

1. Follow Coding standards and practices.
2. Maintain tools and equipment.
3. Follow safety practices.
4. Follow ethical practices

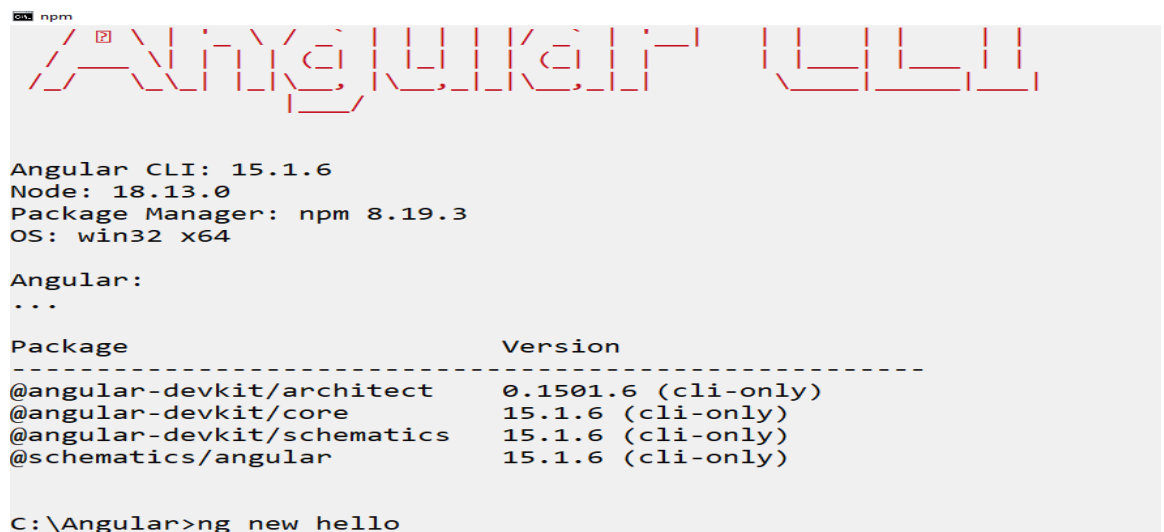
G. Prerequisite Theory:

To create a project in Angular After setup environment, go to directory where you already installed Angular/Cli and type following commands using either terminal of visual studio or command line.

ng new projectname

Here, You can use the projectname of your choice.

Let us now run the above command in the command line.



```

Angular CLI: 15.1.6
Node: 18.13.0
Package Manager: npm 8.19.3
OS: win32 x64

Angular:
...

Package                                Version
-----
@angular-devkit/architect              0.1501.6 (cli-only)
@angular-devkit/core                   15.1.6 (cli-only)
@angular-devkit/schematics             15.1.6 (cli-only)
@schematics/angular                    15.1.6 (cli-only)

C:\Angular>ng new hello

```

Once you run the command it will ask you about routing as shown below –



```

Angular CLI: 15.1.6
Node: 18.13.0
Package Manager: npm 8.19.3
OS: win32 x64

Angular:
...

Package                                Version
-----
@angular-devkit/architect              0.1501.6 (cli-only)
@angular-devkit/core                   15.1.6 (cli-only)
@angular-devkit/schematics             15.1.6 (cli-only)
@schematics/angular                    15.1.6 (cli-only)

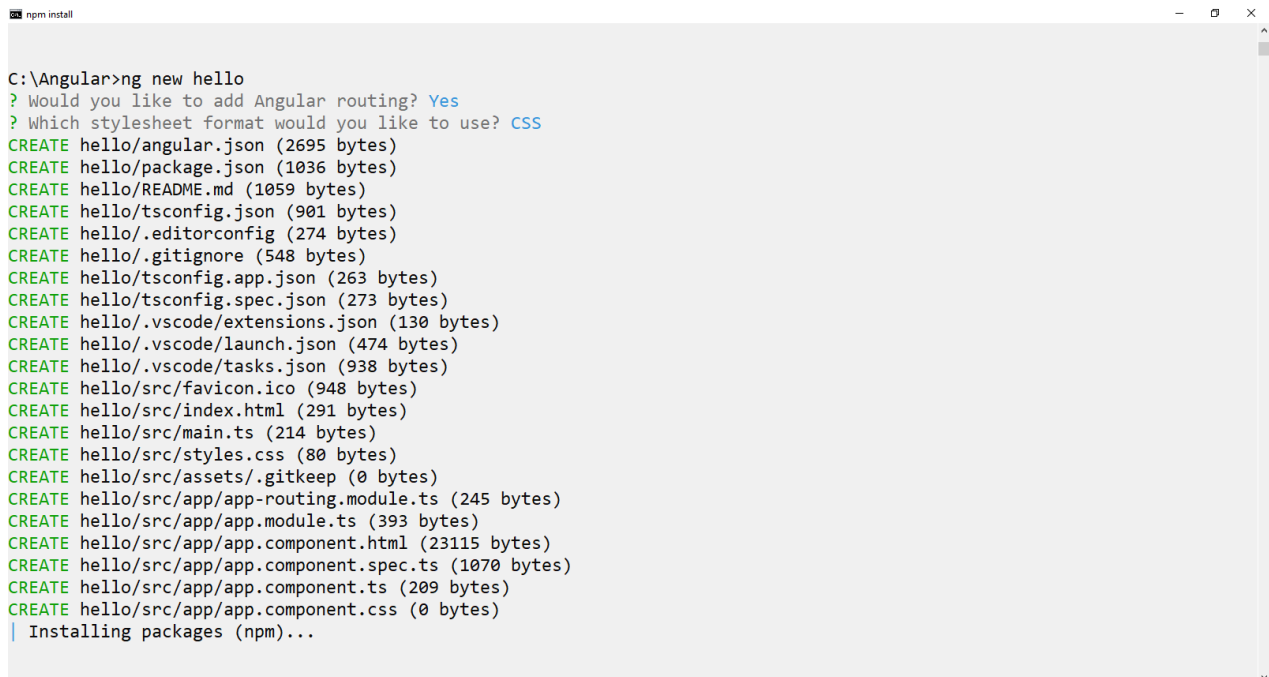
C:\Angular>ng new hello
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use?
> CSS
SCSS [ https://sass-lang.com/documentation/syntax#scss ]
Sass [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
Less [ http://lesscss.org ]

```

Type y to add routing to your project setup and again ask question for the stylesheet as shown below.

The options available are CSS, Sass, Less and Stylus. In the above screenshot, the arrow is on CSS. To change, you can use arrow keys to select the one required for your project setup. At present, we select CSS option for our project-setup.

It installs all the required packages necessary for our project to run in Angular as shown below.



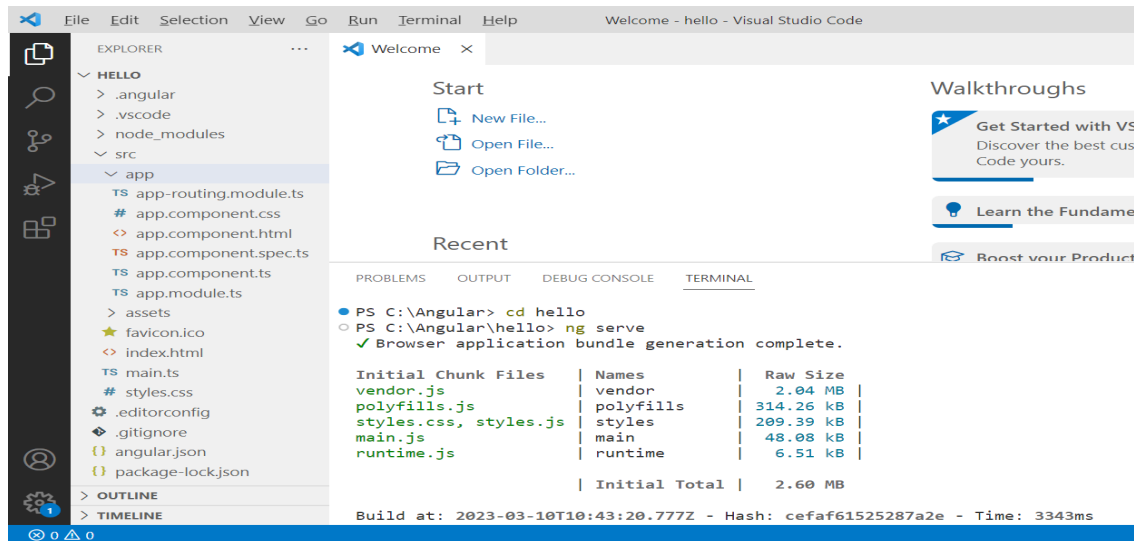
```
npm install

C:\Angular>ng new hello
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE hello/angular.json (2695 bytes)
CREATE hello/package.json (1036 bytes)
CREATE hello/README.md (1059 bytes)
CREATE hello/tsconfig.json (901 bytes)
CREATE hello/.editorconfig (274 bytes)
CREATE hello/.gitignore (548 bytes)
CREATE hello/tsconfig.app.json (263 bytes)
CREATE hello/tsconfig.spec.json (273 bytes)
CREATE hello/.vscode/extensions.json (130 bytes)
CREATE hello/.vscode/launch.json (474 bytes)
CREATE hello/.vscode/tasks.json (938 bytes)
CREATE hello/src/favicon.ico (948 bytes)
CREATE hello/src/index.html (291 bytes)
CREATE hello/src/main.ts (214 bytes)
CREATE hello/src/styles.css (80 bytes)
CREATE hello/src/assets/.gitkeep (0 bytes)
CREATE hello/src/app/app-routing.module.ts (245 bytes)
CREATE hello/src/app/app.module.ts (393 bytes)
CREATE hello/src/app/app.component.html (23115 bytes)
CREATE hello/src/app/app.component.spec.ts (1070 bytes)
CREATE hello/src/app/app.component.ts (209 bytes)
CREATE hello/src/app/app.component.css (0 bytes)
| Installing packages (npm)...
```

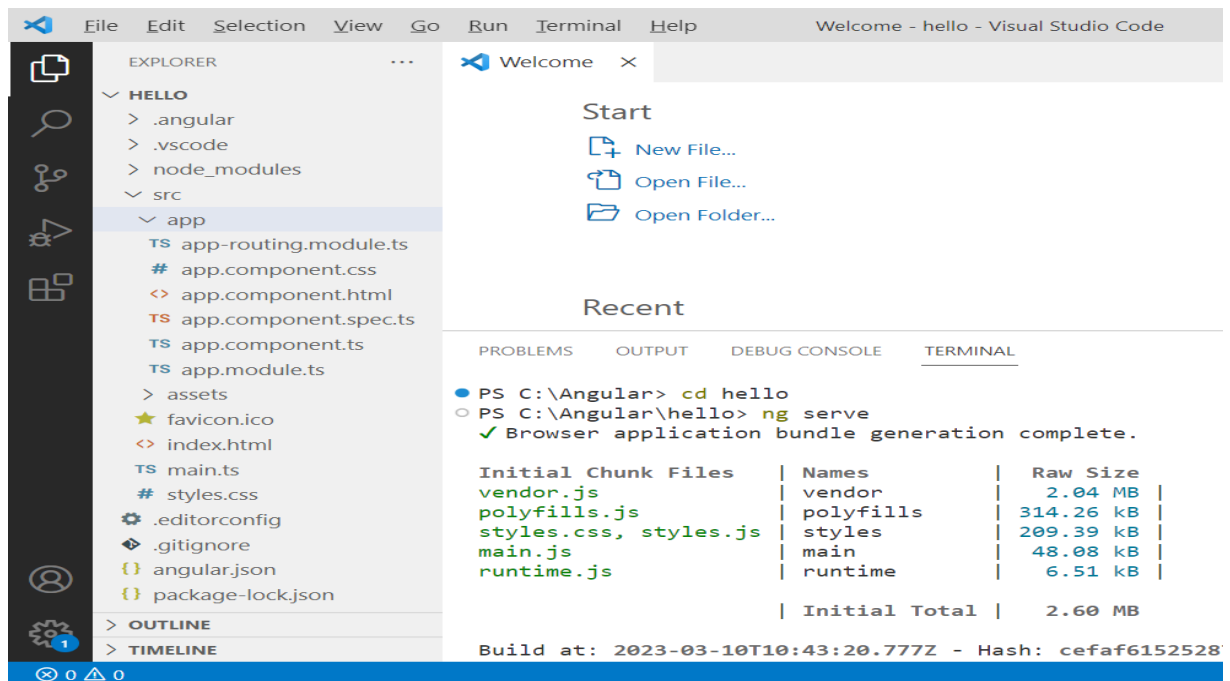
Once all packages installed, Our project is installed successfully. We need to go our project directory using following command

cd projectname

Now I am going to show you using Visual code editor. You can see above command in our example as given below.



If you want to see the file structure of our project where you can manage your application, open Visual studio code editor which is already installed and open our project directory, you will get the folder structure that looks like the one given below.



To compile our project, the following command is used in angular.

ng serve

The `ng serve` command builds the application and starts the web server.

You will see the below output when the command starts executing.

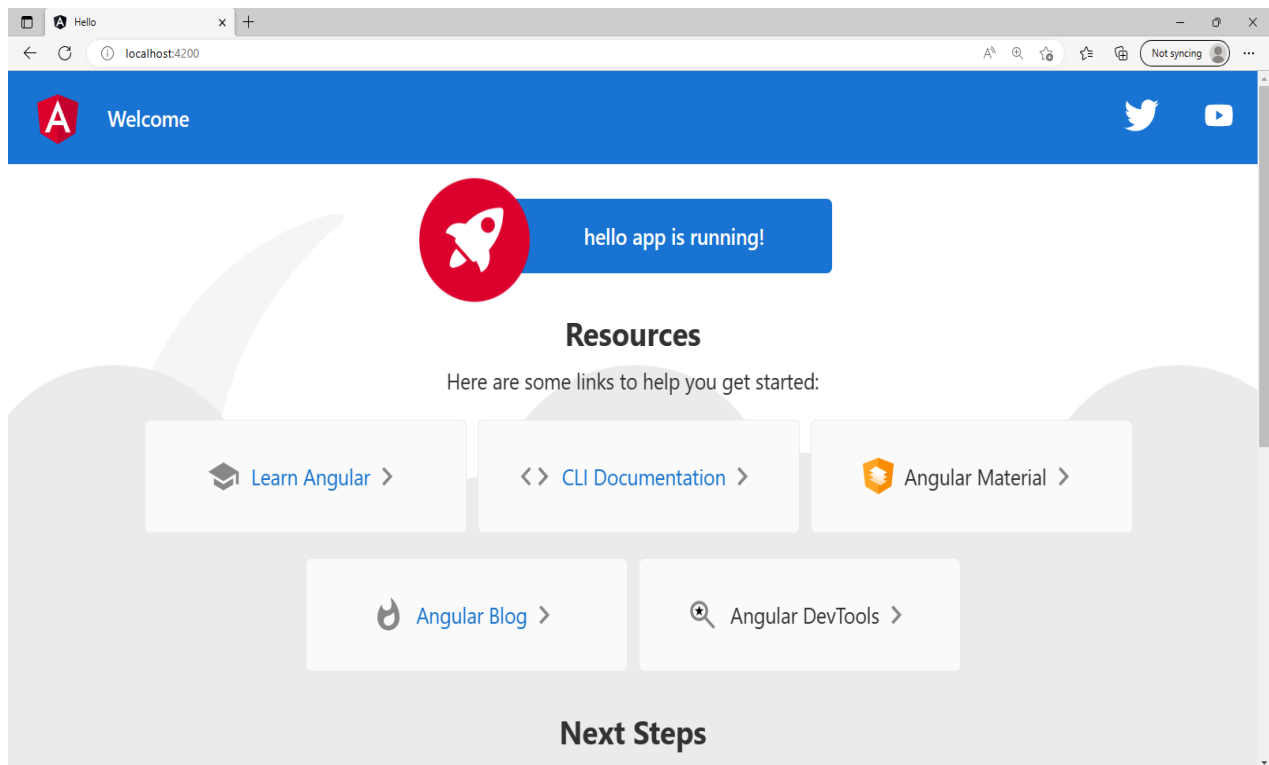
```
PS C:\Angular\hello> cd..
PS C:\Angular> cd hello
PS C:\Angular\hello> ng serve
✓ Browser application bundle generation complete.

Initial Chunk Files | Names | Raw Size
vendor.js           | vendor | 2.04 MB
polyfills.js        | polyfills | 314.26 kB
styles.css, styles.js | styles | 209.39 kB
main.js             | main | 48.08 kB
runtime.js          | runtime | 6.51 kB
                    | Initial Total | 2.60 MB

Build at: 2023-03-10T10:43:20.777Z - Hash: cefaf61525287a2e - Time: 3343ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
```

Once the project is compiled and type url, `http://localhost:4200/` in the browser, you will see your project home page as shown below.



The necessary files for this app are given below:

1. **app.module.ts**: This is the root module of an Angular application. It is responsible for importing and configuring all the components, services, directives, and other modules used in the application. The module is typically defined in a file named **app.module.ts**. Here is an example:

```
import { NgModule } from '@angular/core';
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

```
export class AppModule { }
```

2. **app.component.ts**: This is the root component of an Angular application. It defines the structure and behavior of the main view of the application. The component is typically defined in a file named **app.component.ts**.

Here is an example:

```
import { Component } from '@angular/core';
```

```
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
}
```

3. **app.component.html**: This is the HTML template for the root component. It defines the layout and content of the main view of the application. The template is typically defined in a file named **app.component.html**. Here is an example:

```
<h1>Hello World!</h1>
```

4. **main.ts**: This is the main entry point of an Angular application. It bootstraps the root module of the application and starts the Angular runtime. The file is typically named **main.ts**.

Here is an example:

```
import { enableProdMode } from '@angular/core';  
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';  
import { AppModule } from './app.module';  
enableProdMode();  
platformBrowserDynamic().bootstrapModule(AppModule)  
  .catch(err => console.error(err));
```

5. **angular.json**: This is the configuration file for an Angular application. It defines the settings and options for building, serving, and testing the application. The file is typically named **angular.json**.

Now we are going to understand concepts of string interpolation to make our desired output.

String interpolation is a feature in Angular that allows you to embed dynamic values from your component's TypeScript code into your HTML template. This can make your HTML templates

more dynamic and interactive. Here's an example of how to use string interpolation in both the HTML and TypeScript files of an Angular component:

HTML Template:

```
<h1>Welcome {{name}}!</h1>
```

TypeScript File:

```
import { Component } from '@angular/core';
```

```
@Component({  
  selector: 'app-welcome',  
  templateUrl: './welcome.component.html',  
  styleUrls: ['./welcome.component.css']  
})  
export class WelcomeComponent {  
  name = 'Admin';  
}
```

In the HTML template, we've used double curly braces `{{ }}` to surround the name variable, which is defined in the TypeScript file. This tells Angular to replace the `{{name}}` with the value of the name variable when the component is rendered.

In the TypeScript file, we've imported the Component decorator from `@angular/core` and used it to define our component. We've also defined a name variable and set it to 'John'. When the component is rendered, Angular will replace `{{name}}` with the value of name, which is 'Admin'.

You can also use string interpolation to perform simple operations within the template, such as concatenating strings or performing basic arithmetic:

```
<p>{{firstName + ' ' + lastName}} is {{age + 1}} years old</p>
```

Where firstName,lastName and age property variables's value must be declared in TypeScript file.

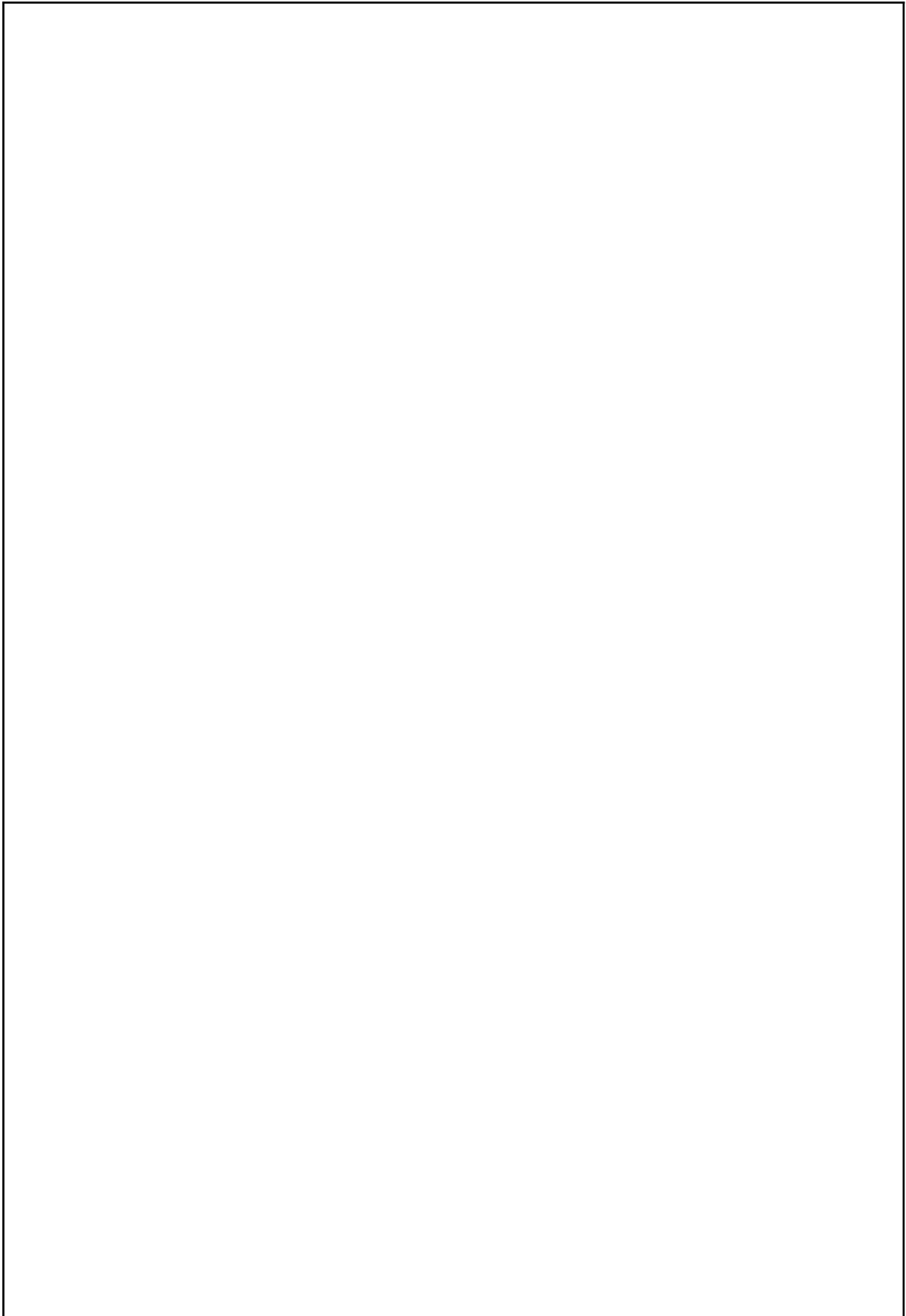
H. Resources/Equipment Required

S r.	Equipment/ Software Resources	Specification
---------	----------------------------------	---------------

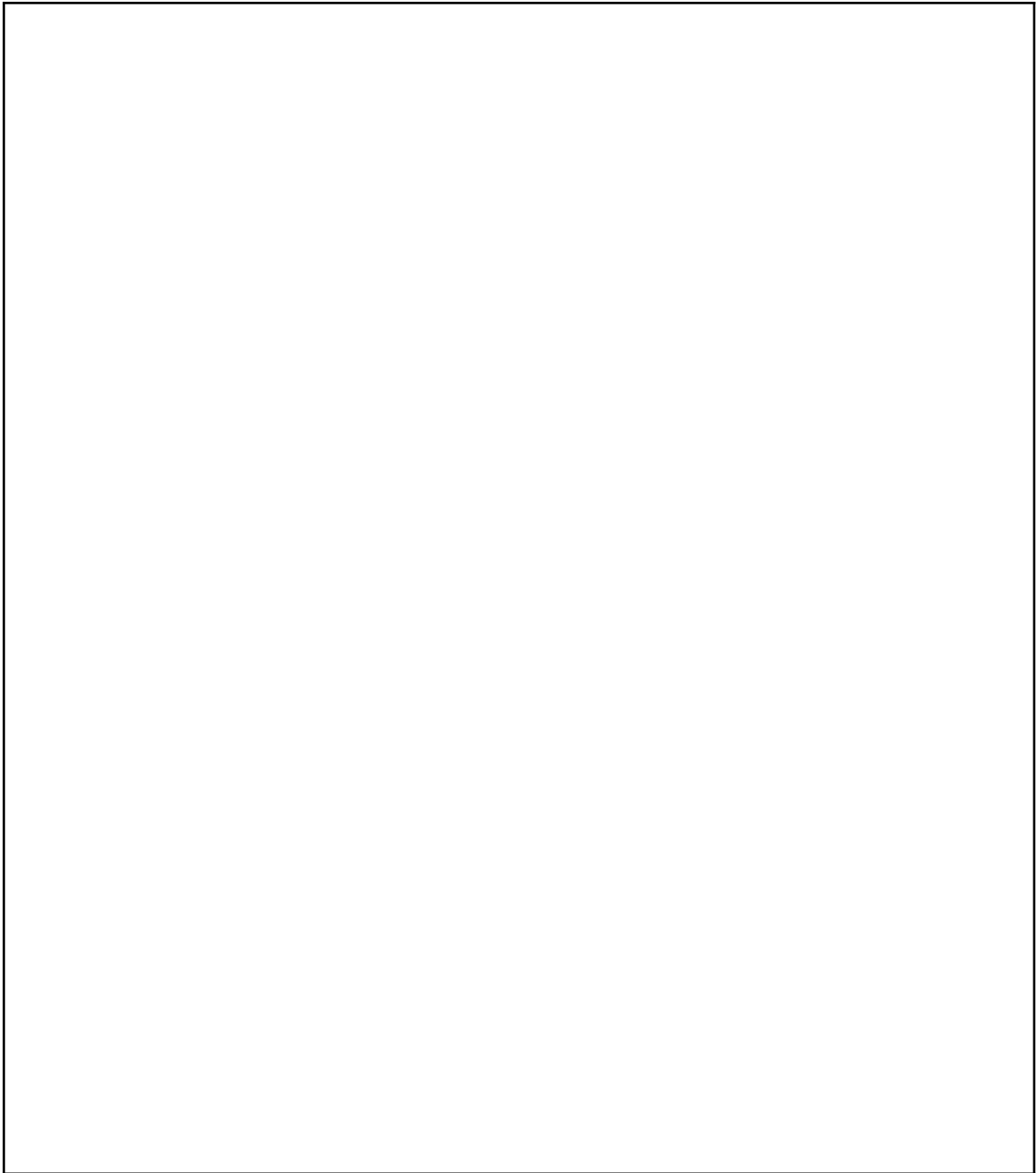
N o.		
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

I. Program Source code Output

SOURCE CODE

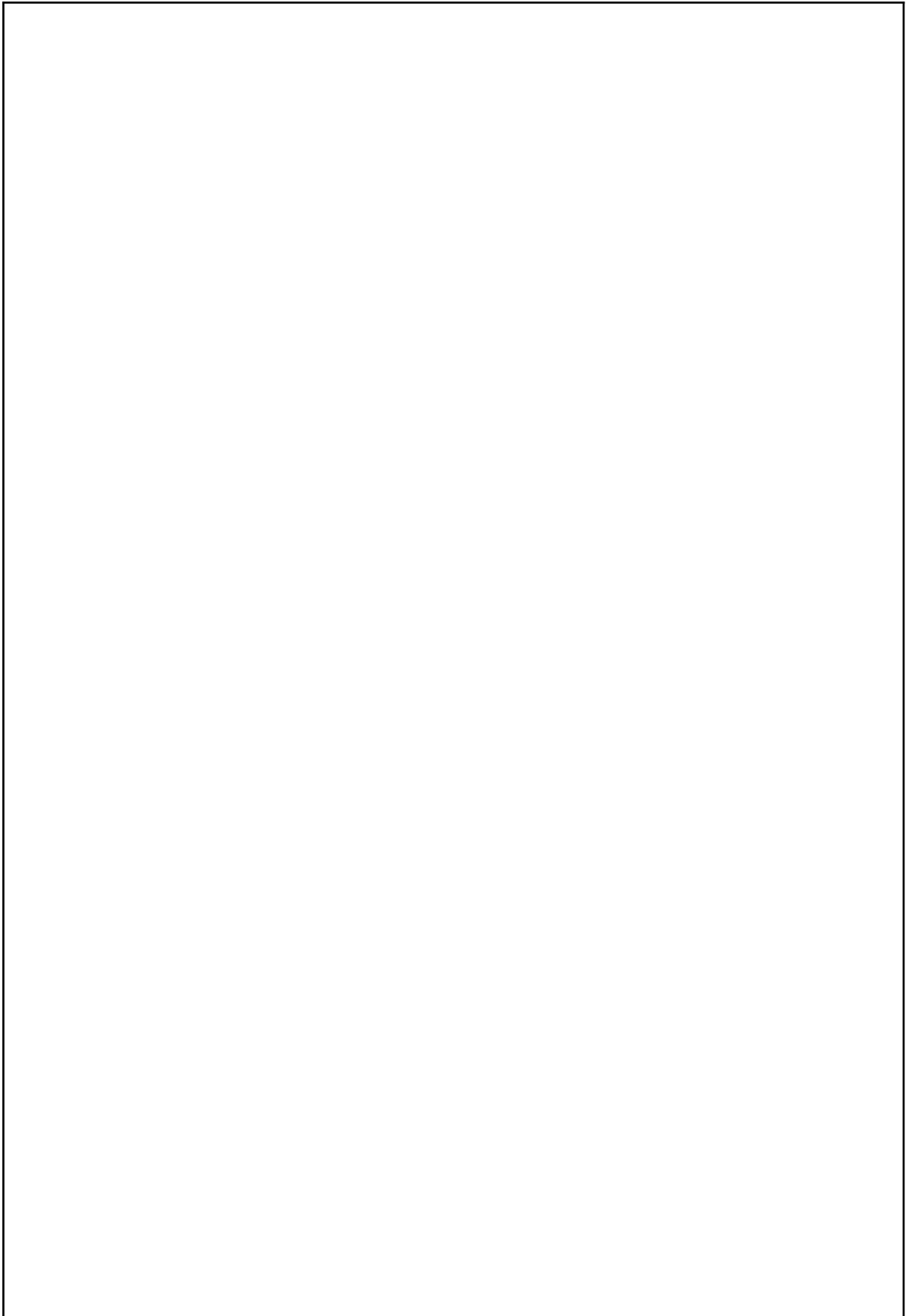


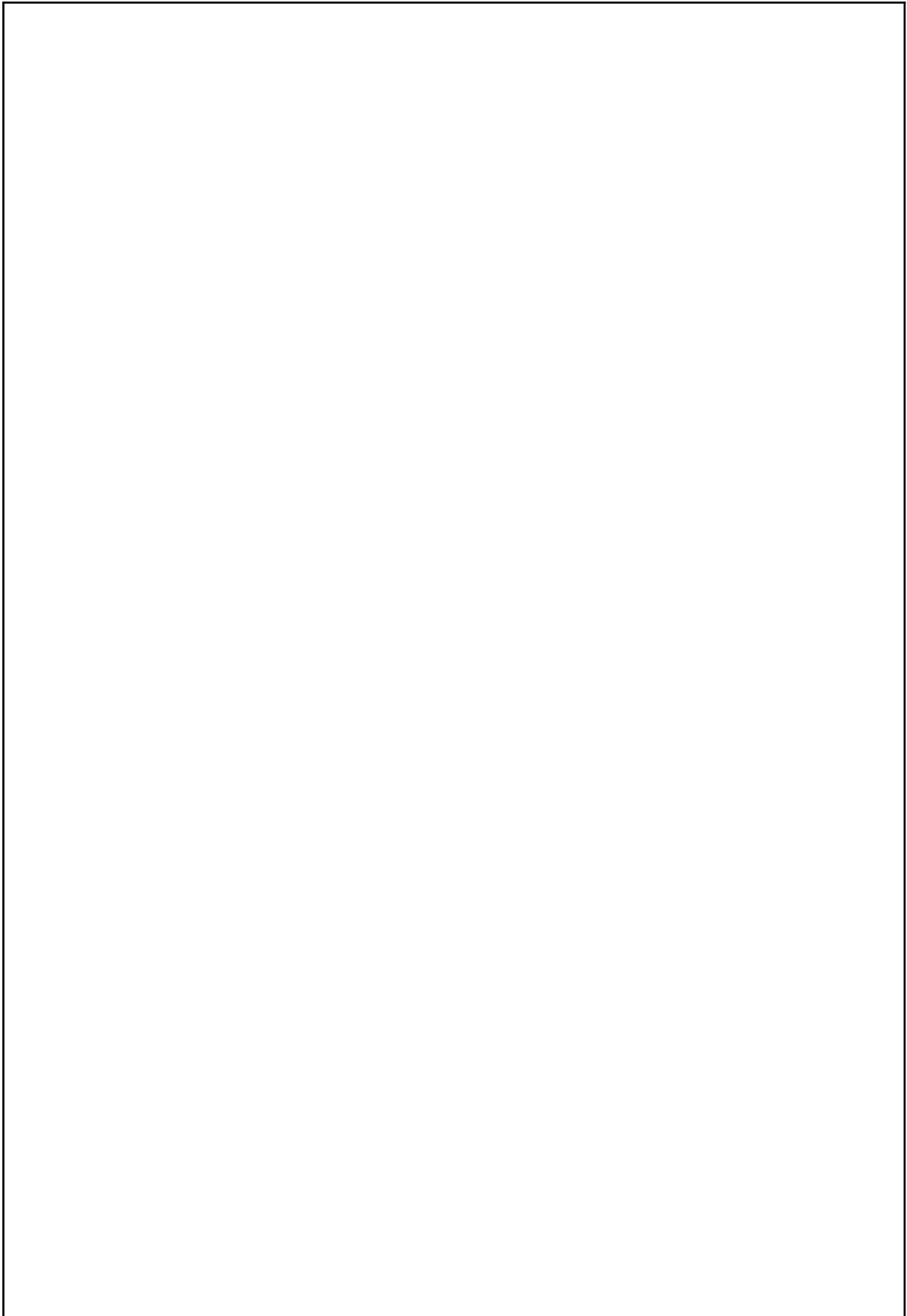
OUTPUT

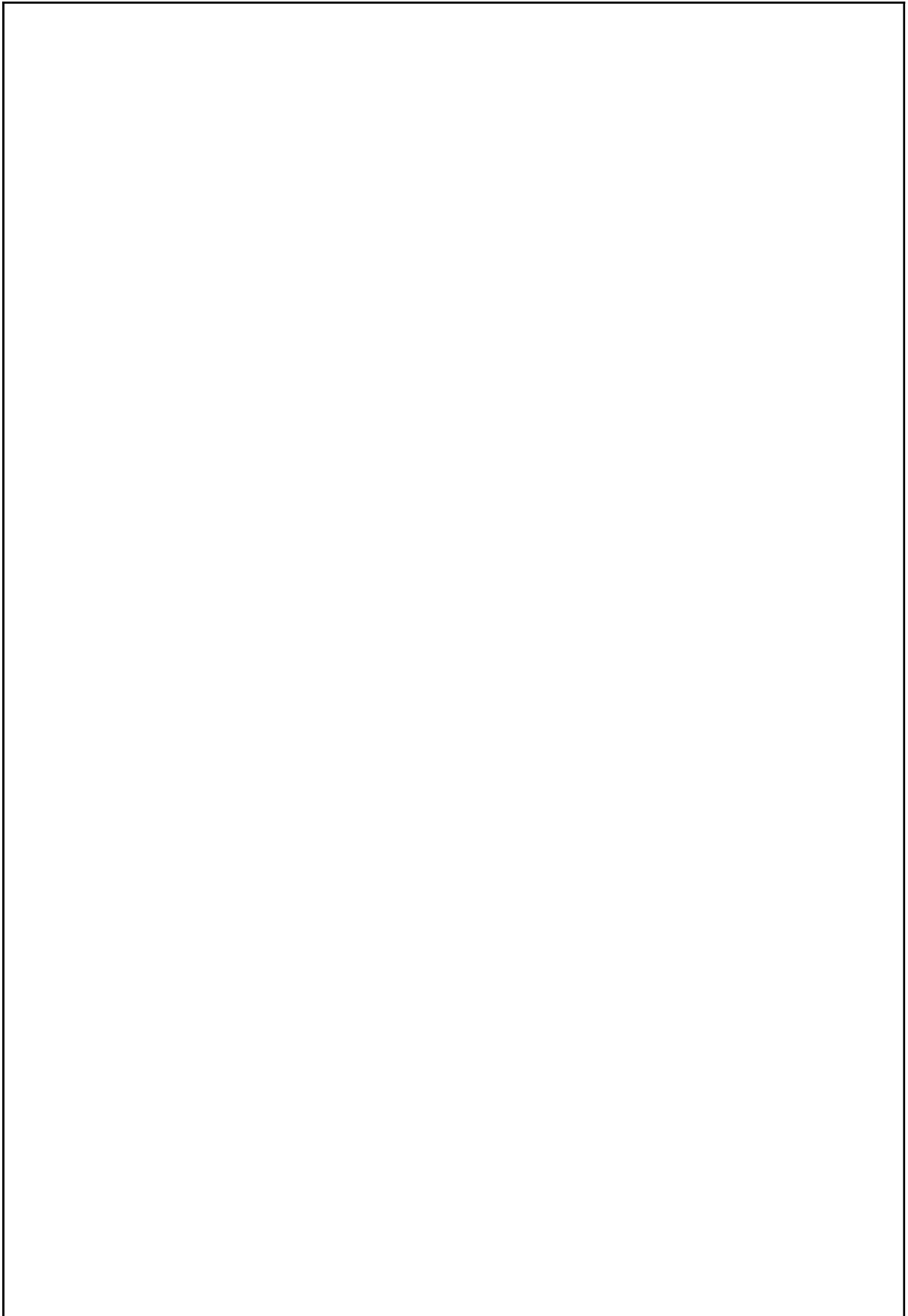


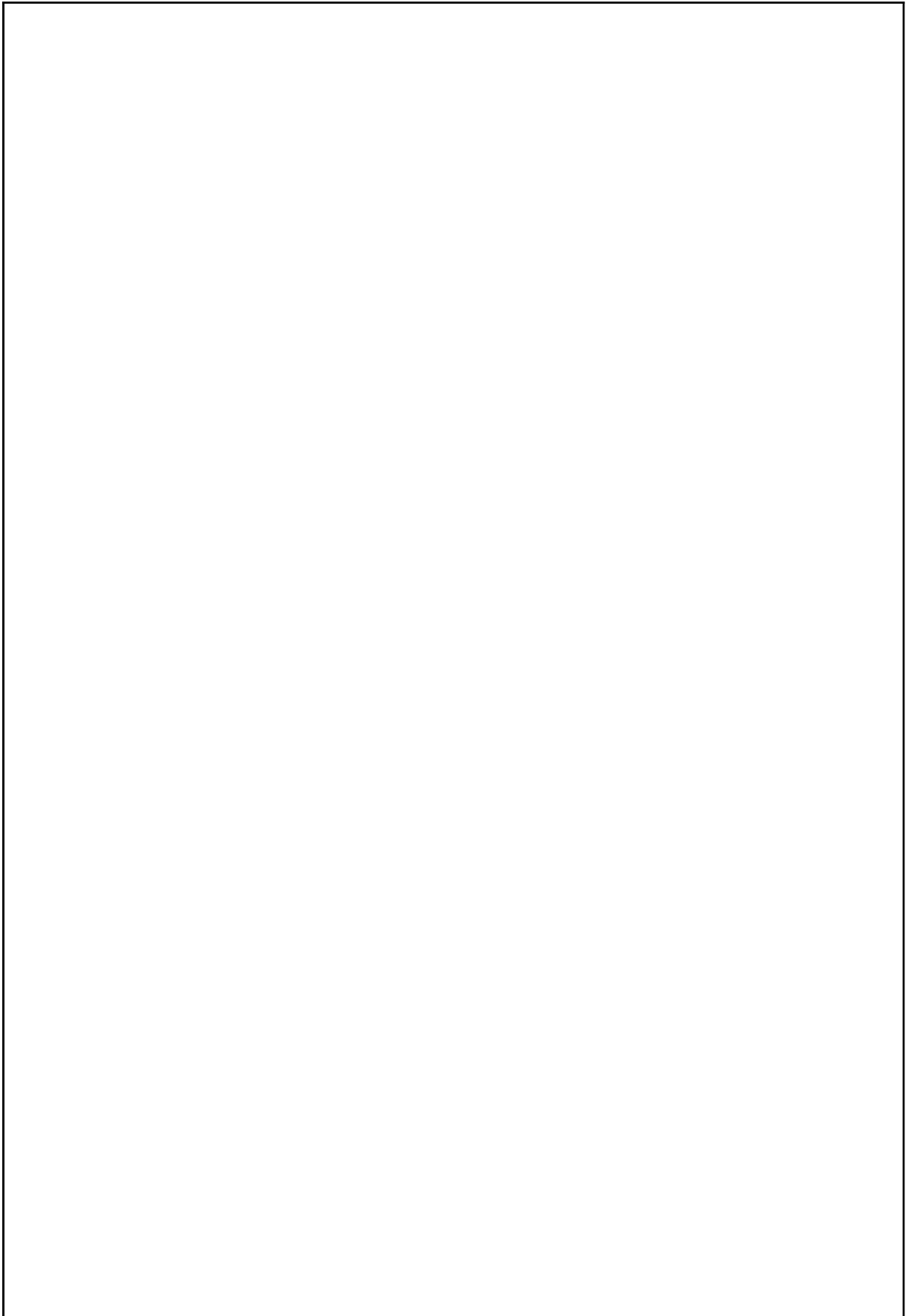
J. Practical related Exercises.

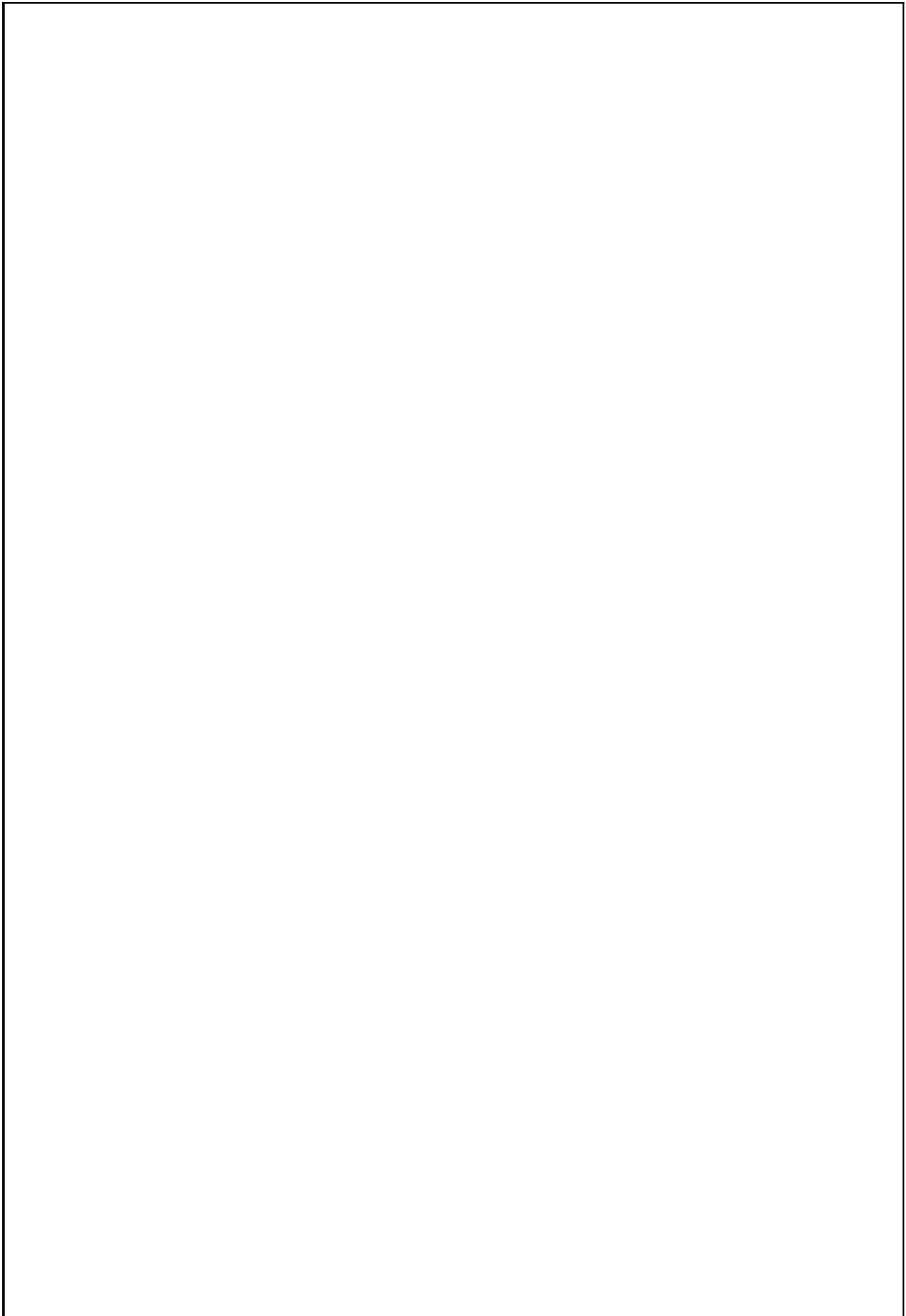
1. How do you create and run a new Angular project using the Angular CLI?
2. Displaying a variable value such as your Name in the template.
3. Using expressions to calculate values on web page.
4. Describe NgModul Decorator in angular with example.

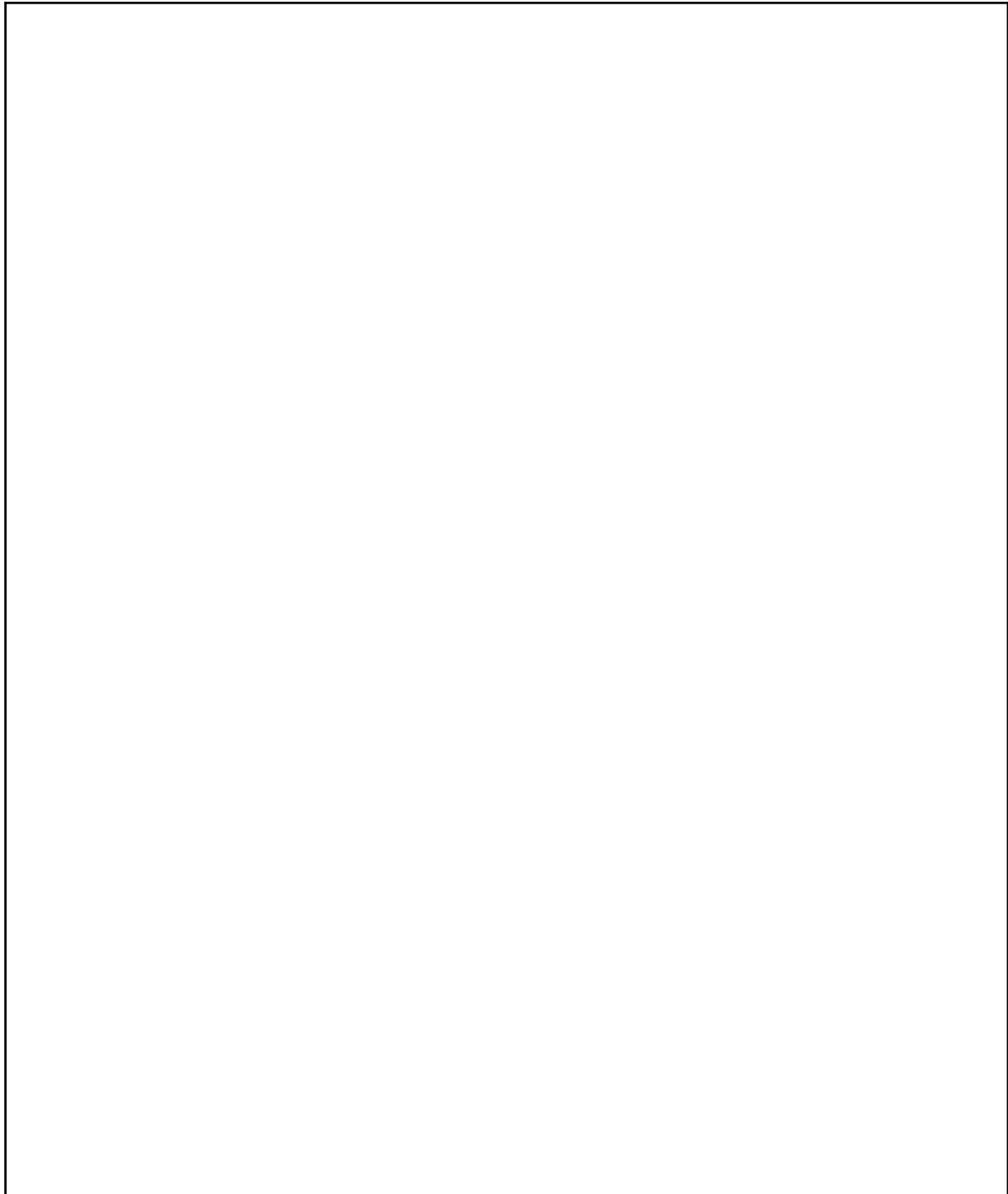












K. References Links

1. <https://www.youtube.com/watch?v=TC1oVXIVE3M>
2. <https://www.geeksforgeeks.org/hello-world-program-first-program-while-learning-programming/>
3. <https://www.simplilearn.com/tutorials/angular-tutorial/angular-hello-world>
4. <https://www.w3schools.blog/hello-world-angular-7>

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of Input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No.3: Design a web page to utilize property binding and event binding concepts using button and textbox controls.

A. Objective:

1. To know how to exchange data between the component and the template
2. Create dynamic, interactive web applications that respond to user input and update dynamically based on changes in the component.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the digital electronics engineering problems.
2. **Design/ development of solutions:** Design solutions for digital electronics engineering well-defined technical problems and assist with the design of systems components or processes to meet specified needs.
3. **Engineering Tools, Experimentation and Testing:** Apply modern digital electronics engineering tools and appropriate technique to conduct standard tests and measurements.

C. Expected Skills to be developed based on competency:

To create dynamic and interactive user interfaces, Manipulating the DOM, Creating responsive UIs, Implementing data-driven applications.

D. Expected Course Outcomes(Cos)

Setup environment for angular practical execution with NODE JS and NPM Package manager.

E. Practical Outcome(PRo)

Design a web page to utilize property binding and event binding concepts using button and textbox controls.

F. Expected Affective domain Outcome(ADos)

1. Follow Coding standards and practices.
2. Maintain tools and equipment.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory:

PropertyBinding :

Property binding is a feature in Angular that allows you to set the value of an HTML element property to a value from the component. In other words, you can bind a property of an HTML element to a property in your component, and the value of the HTML property will be dynamically updated as the value of the component property changes.

Here is an example of how to use property binding in an Angular component:

HTML Template (app.component.html)

```
<p>Welcome to {{title}}!</p>
```

```
<button [disabled]="isDisabled">Click me</button>
```

In this example, we have two different uses of property binding. The first one binds the content of a paragraph (<p>) to a property **title** in the component. The curly braces {{ }} indicate that this is an example of one-way data binding. When the component is rendered, the value of the **title** property is interpolated into the template.

The second use of property binding binds the **disabled** attribute of a button to the **isDisabled** property of the component. Square brackets [] are used to indicate that

this is an example of property binding. When the value of the **isDisabled** property changes, the **disabled** attribute of the button will be updated accordingly.

TypeScript File (app.component.ts)

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'Property binding Demo';
  isDisabled = false;
}
```

In the TypeScript file for our component, we define two properties: **title** and **isDisabled**. The **title** property is a simple string that is used in the first example of property binding to set the content of a paragraph.

The **isDisabled** property is a boolean value that is used in the second example of property binding to disable or enable a button. By default, the **isDisabled** property is set to **false**, so the button is enabled. If we were to change the value of the **isDisabled** property to **true**, the button would become disabled.

By using property binding, we can make our Angular templates more dynamic and interactive. We can easily bind properties of HTML elements to properties in our component, and update them in real time as our data changes.

Event binding:

Event binding is a feature in Angular that allows you to listen for and respond to events that occur in the HTML template, such as button clicks, key presses, or mouse movements. You can bind an event to a method in your component, and when the event is triggered, the method will be called with any relevant data.

Here is an example of how to use event binding in an Angular component:

HTML Template (app.component.html)

```
<button (click)="onClick()">Click me</button>
```

```
<p>{{message}}</p>
```

In this example, we have a button element that is used to trigger an event, and a paragraph element that is used to display a message based on the event. The (click) syntax is used to bind the click event of the button to a method `onClick()` in the component. When the button is clicked, the `onClick()` method will be called.

The message that is displayed in the paragraph element is determined by the message property in the component, which is updated by the `onClick()` method.

TypeScript File (`app.component.ts`)

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  message: string = "";
  onClick() {
    this.message = 'Button clicked!';
  }
}
```

In the TypeScript file for our component, we define a property **message** that will be used to display a message in the paragraph element. We also define a method **onClick()** that will be called when the button is clicked.

In the **onClick()** method, we update the value of the **message** property to **'Button clicked!'**. This will cause the message to be updated in the HTML template, and the new message will be displayed to the user.

By using event binding, we can create more interactive and responsive Angular components. We can listen for events in the HTML template and respond to them by updating the properties and methods in our component. This allows us to create dynamic and engaging user interfaces that respond to user input in real time.

ngModel Directive:

ngModel is a built-in directive in Angular that allows you to perform two-way data binding between a form control element and a property in your component. It's typically used with input and select elements to capture user input and update the component's properties with the entered value.

Here's an example of how to use **ngModel** in an Angular component:

HTML Template (app.component.html)

```
<input [(ngModel)]="name" placeholder="Enter your name">
<p>Your name is: {{name}}</p>
```

In this example, we have an input element that is used to capture the user's name. We use the **[(ngModel)]** syntax to bind the input's value to the **name** property in the component.

This is a two-way data binding, which means that any changes to the input element's value will be reflected in the **name** property, and any changes to the **name** property will be reflected in the input element's value.

We also have a paragraph element that displays the value of the **name** property.

TypeScript File (app.component.ts)

```
import { Component } from '@angular/core';
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  name: string = "";
}
```

In the TypeScript file for our component, we define a property **name** that will be used to store the user's name. By default, the **name** property is initialized to an empty string.

When the user types a value into the input element, the **name** property will be updated with that value. When the **name** property is updated (for example, if it's updated in code), the input element's value will be updated as well.

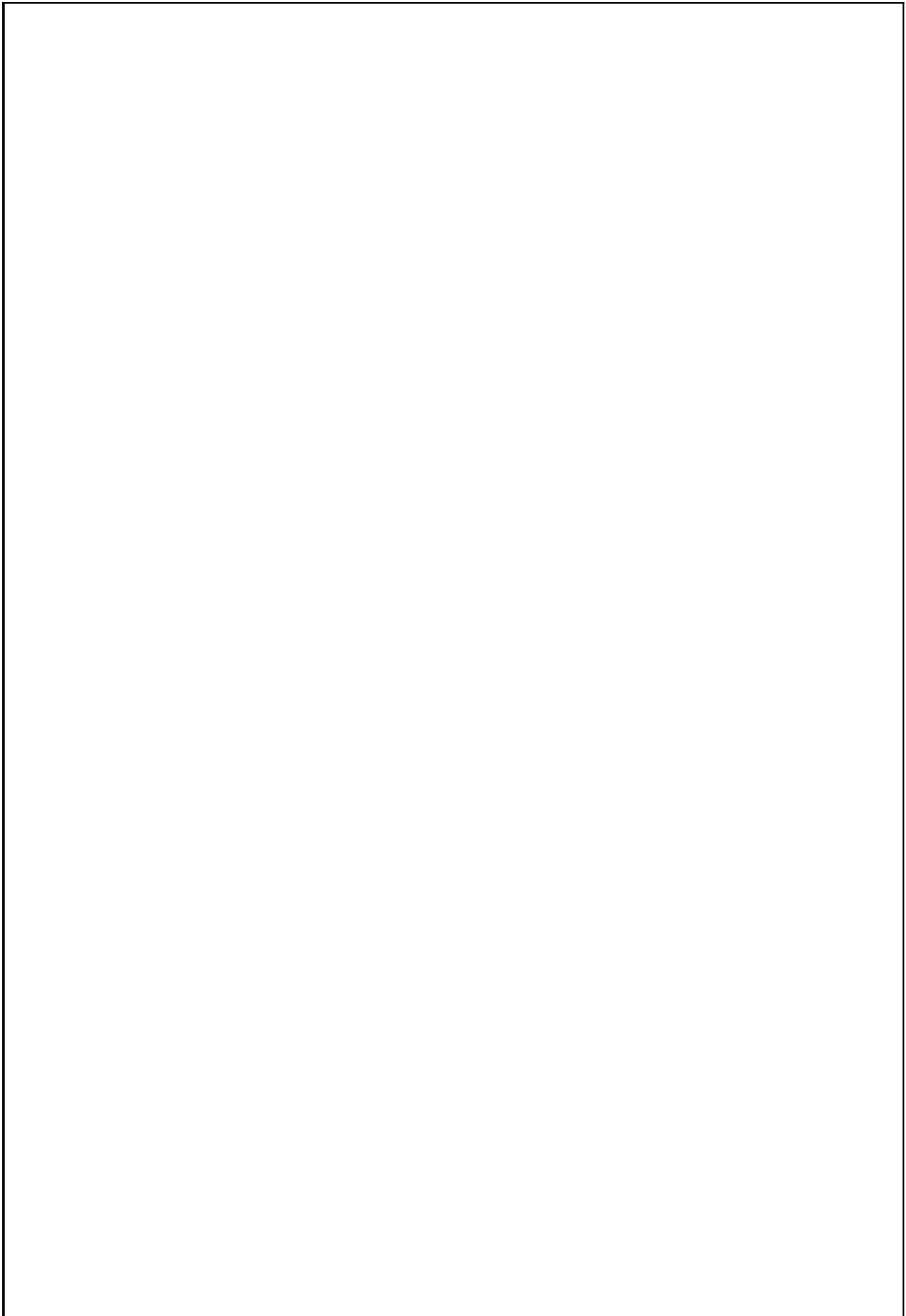
By using **ngModel**, we can easily capture user input and update our component's properties in real time. We don't have to manually update the properties or listen for events - **ngModel** takes care of all of that for us. This makes it a very useful tool for creating forms and other interactive user interfaces in Angular.

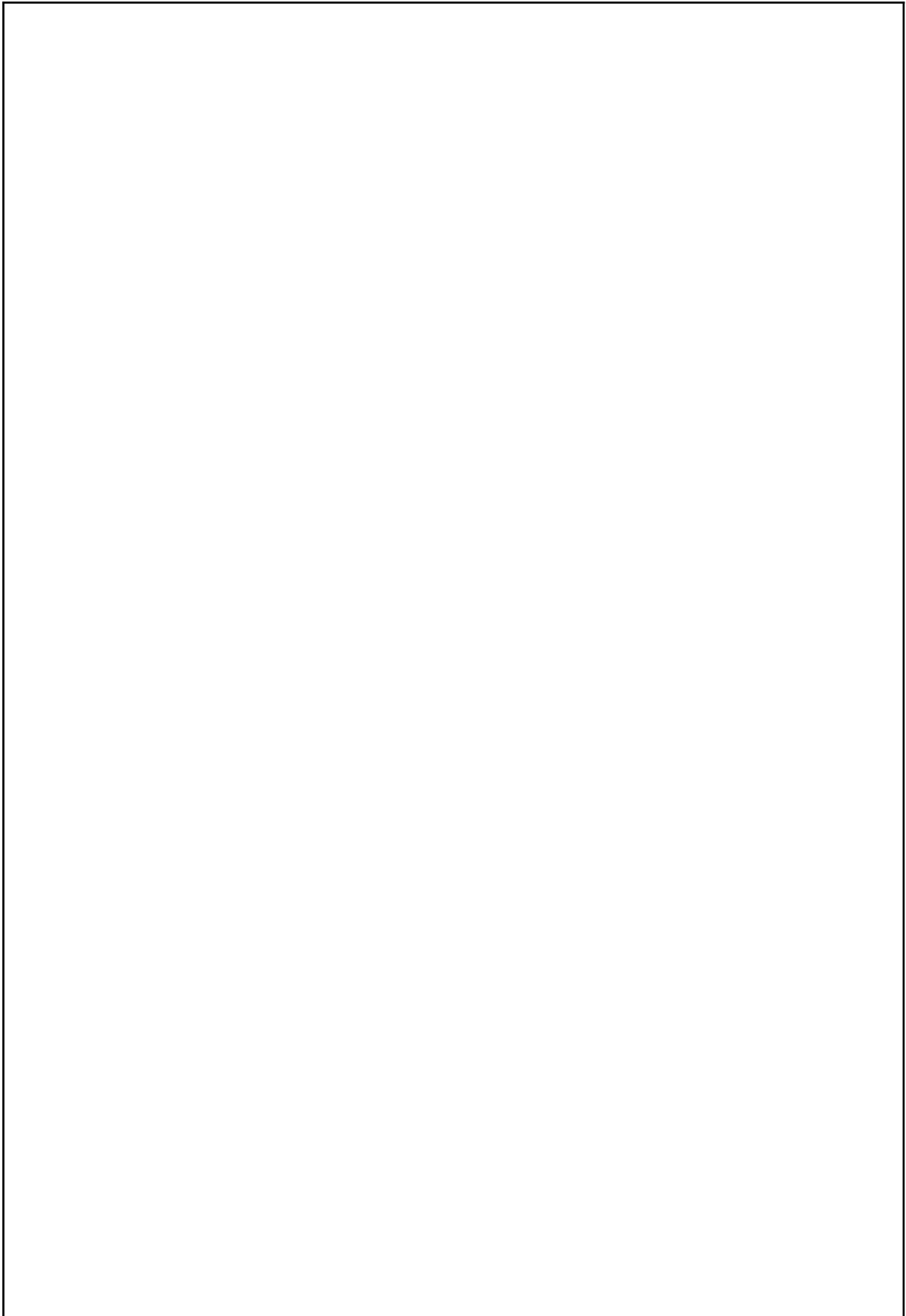
H. Resources/Equipment Required

S r. N o.	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

I. Program Source code Output

SOURCE CODE

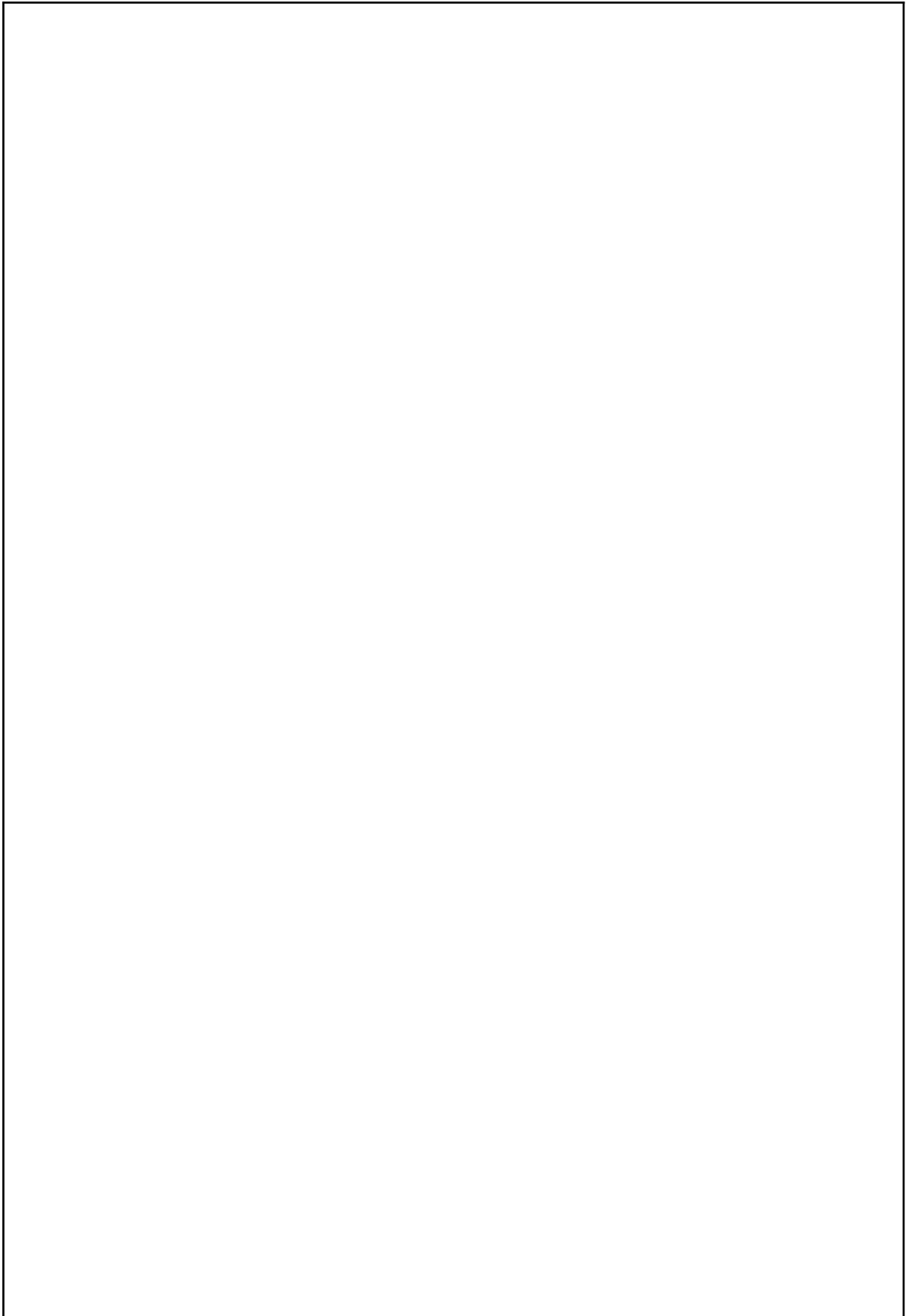


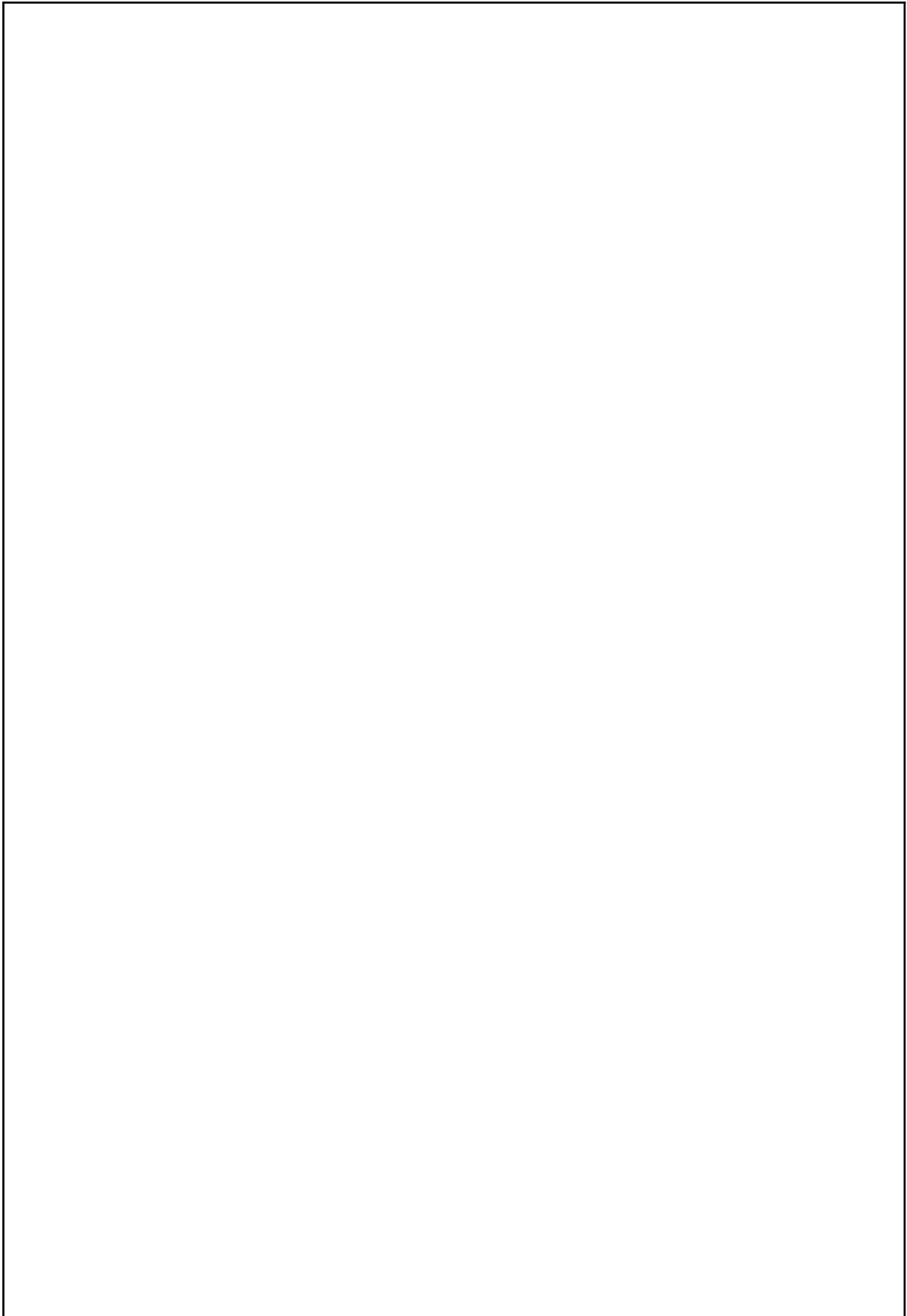


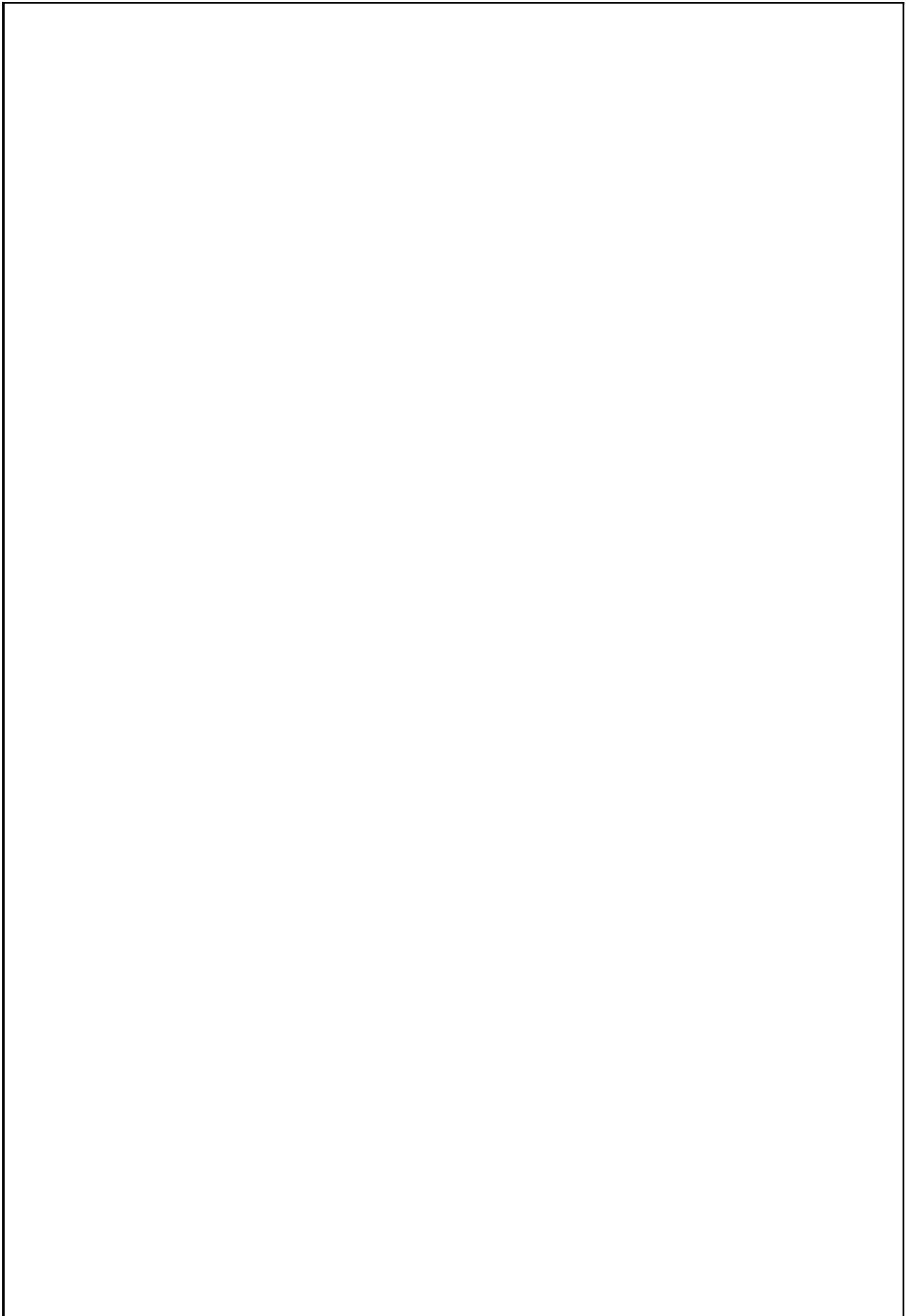
OUTPUT

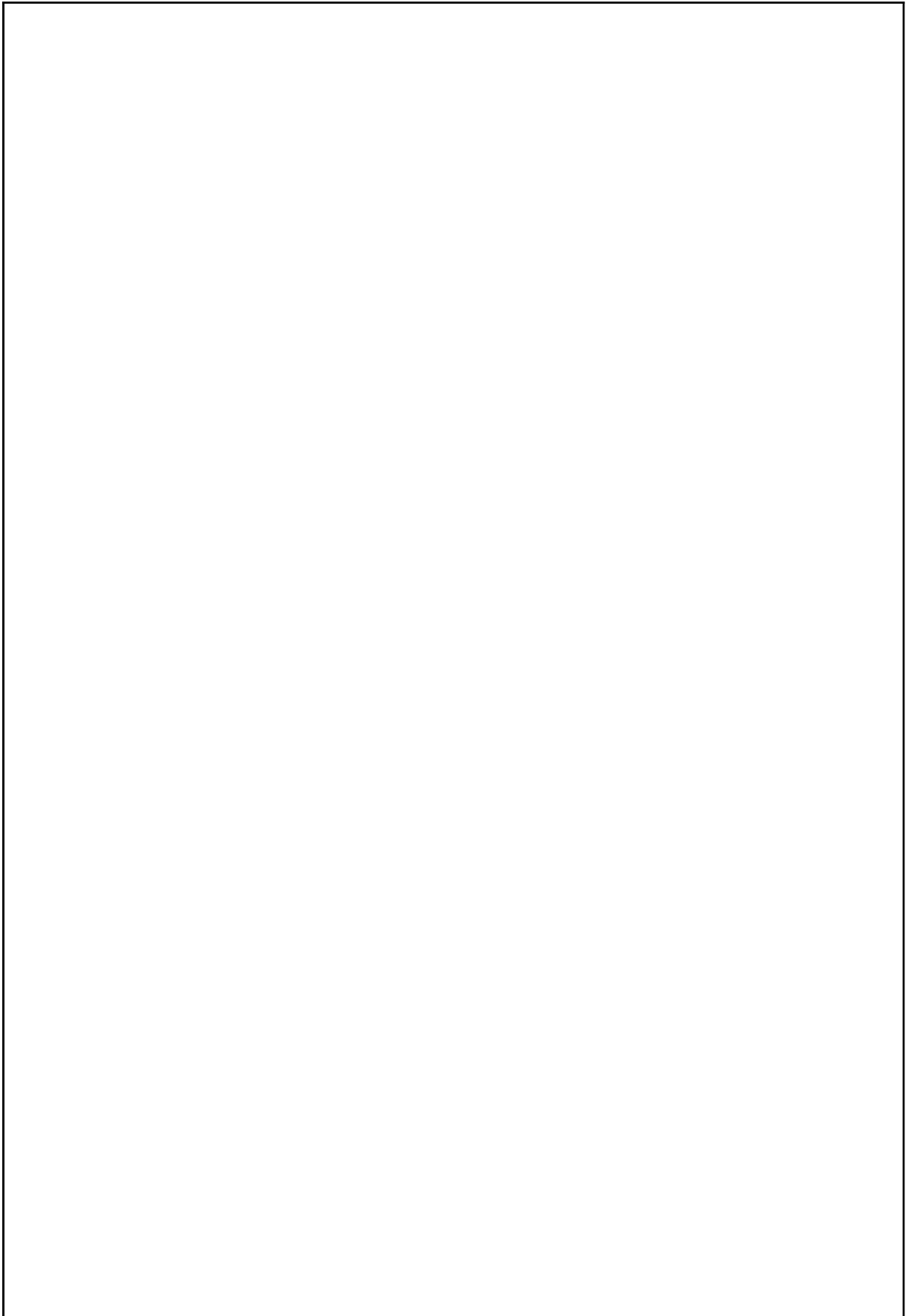
J. Practical related Exercises.

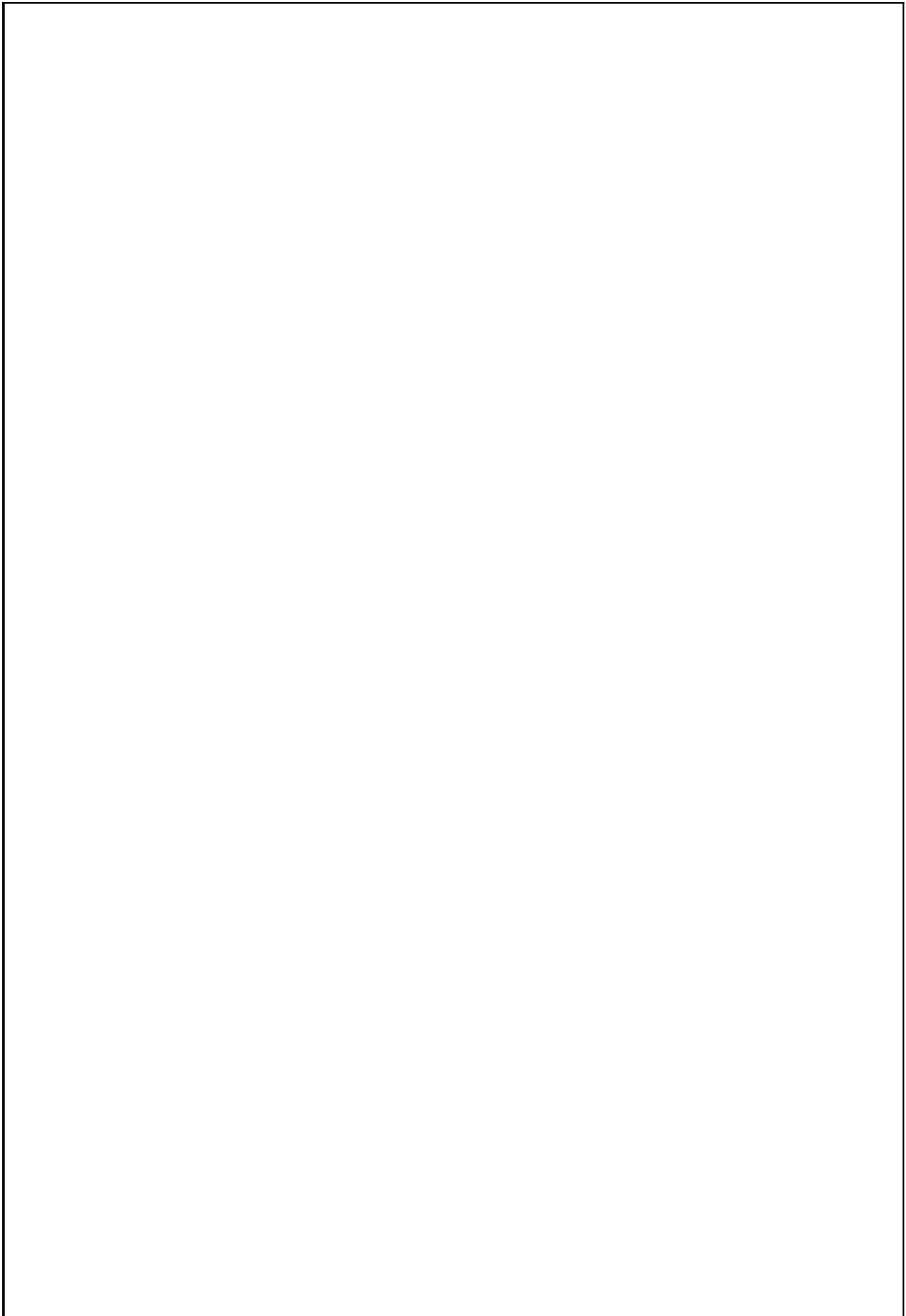
1. Design a webpage to display your name, enrolment number, college name using interpolation and photo using property binding.
2. Design a webpage to display your name, enrolment number, college name using event binding.[hint : name, number and college name values must be declared in ts file]
3. Design a login webpage with validations using ngModle Directive.

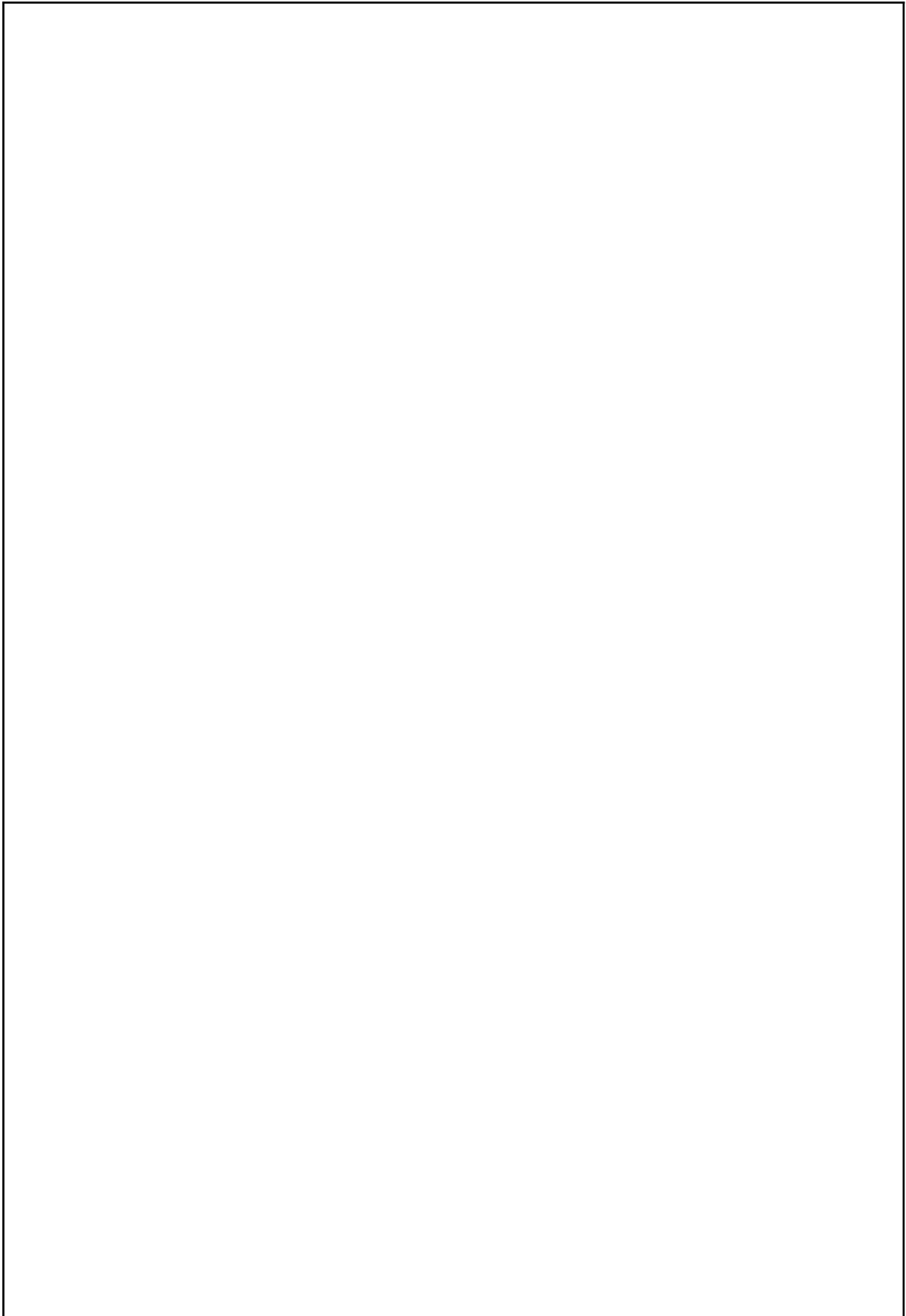


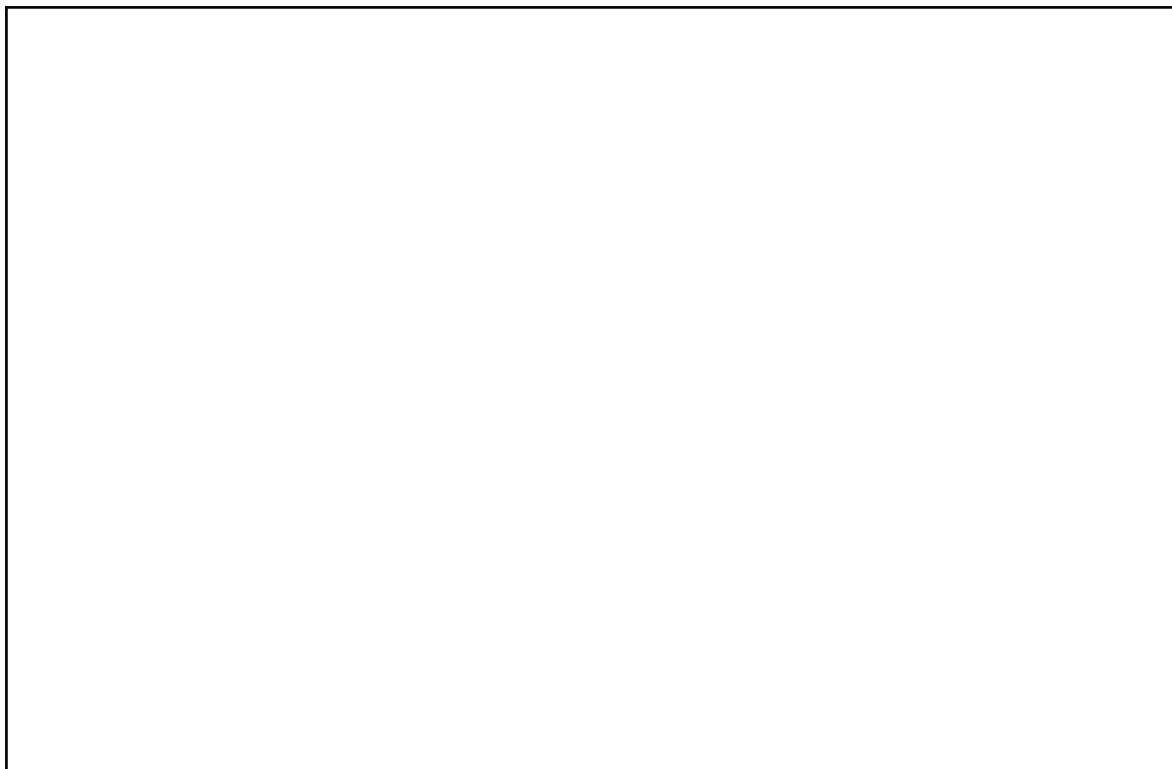












K. References Links

1. <https://www.youtube.com/watch?v=IUHId5qN0oI>
2. <https://www.w3schools.blog/angular-7-property-binding>
3. <https://www.geeksforgeeks.org/angular-forms-ngmodel-directive/>
4. <https://www.youtube.com/watch?v=bKfbzpANUFE>
5. <https://www.edureka.co/blog/angular-ngmodel/>

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of Input and output displayed (messaging and formatting).	10

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
	Total	100

Signature with Date

Practical No.4: Create various components of web page using Attribute Directives.

A. Objective:

1. Create a more modular, scalable, and maintainable web page
2. Manipulate and add functionality to HTML elements by binding custom attributes.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.
2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.
3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.

4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.

5. **Life-long learning:** Ability to analyze individual needs and engage in updating in the context of technological changes *in field of engineering*.

C. Expected Skills to be developed based on competency:

Learner can develop a wide range of technical and soft skills including angular framework knowledge, collaboration etc. making them better equipped to develop high-quality, efficient, and scalable web applications.

D. Expected Course Outcomes(Cos)

Apply angular directives, components and pipes in different web page development.

E. Practical Outcome(PRo)

Create various components of web page using Attribute Directives.

F. Expected Affective domain Outcome(ADos)

1. Follow Coding standards and practices.
2. Maintain tools and equipment.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory:

The component is the basic building block of Angular. It has a selector, template, style, and other properties, and it specifies the metadata required to process the component.

Creating a Component in Angular :

To create a component in any angular application, follow the below steps:

- [1] Get to the angular app via your terminal.
- [2] Create a component using the following command:

`ng g c <component_name>`

OR

`ng generate component <component_name>`

Following files will be created after generating the component:

- o `component_name.component.html`
- o `component_name.component.spec.ts`
- o `component_name.component.ts`

- o `component_name.component.css`

Now we are going to create a header, sidebar, and footer components in Angular, you can follow these steps:

1. Create a new Angular project:
`ng new my-app`
2. Create a header component using the following command:
`ng generate component header`
3. Create a sidebar component using the following command:
`ng generate component sidebar`
4. Create a footer component using the following command:
`ng generate component footer`

This will create the necessary files for each component, including HTML, CSS, TypeScript, and a spec file as we discussed earlier.

5. Open the `header.component.html` file and add the following code:

```
<header>
<h1>My Website</h1>
</header>
```

6. Open the `sidebar.component.html` file and add the following code:

```
<aside>
<h2>Menu</h2>
<nav>
<ul>
<li><a href="#">Home</a></li>
<li><a href="#">>About</a></li>
<li><a href="#">Contact</a></li>
</ul>
</nav>
</aside>
```

7. Open the `footer.component.html` file and add the following code:

```
<footer>
<p>My Website &copy; 2023</p>
</footer>
```

8. Open the app.component.html file and replace the existing code with the following code:

```
<app-header></app-header>
<main>
<app-sidebar></app-sidebar>
<section>
<router-outlet></router-outlet>
</section>
</main>
<app-footer></app-footer>
```

9. Open the app.component.css file and add the following styles:

```
header {
background-color: #333;
color: #fff;
padding: 20px;
}
```

```
aside {
background-color: #f2f2f2;
padding: 10px;
}
```

```
footer {
background-color: #333;
color: #fff;
padding: 20px;
text-align: center;
}
```

```
main {
display: flex;
}
```

```
section {
margin: 20px;
padding: 20px;
border: 1px solid #ccc;
flex-grow: 1;
```

```
}
```

You have designed a basic Angular app with a header, sidebar, and footer components.

ngClass directory

ngClass is a built-in Angular directive that allows you to dynamically add or remove CSS classes to an HTML element based on certain conditions. This is particularly useful when you want to apply different styles to an element based on its state or user interaction.

The **ngClass** directive can be used in two ways:

1. Using a string expression:

```
<div [ngClass]="class1 class2 class3">Example</div>
```

In this example, the **ngClass** directive applies three CSS classes to the **div** element: **class1**, **class2**, and **class3**.

2. Using an object expression:

```
<div [ngClass]="{ 'class1': condition1, 'class2': condition2 }">Example</div>
```

In this example, the **ngClass** directive applies the CSS class **class1** to the **div** element if **condition1** is true, and the CSS class **class2** if **condition2** is true. **condition1** and **condition2** value set in ts file.

Here's a more detailed example that demonstrates how to use **ngClass** to dynamically apply CSS classes based on user interaction:

```
<button [ngClass]="{ 'active': isActive }" (click)="toggleActive()">Toggle Active</button>
```

In this example, the **ngClass** directive applies the CSS class **active** to the **button** element if the **isActive** property is true. The **isActive** property is initially set to false in the component:

```
import { Component } from '@angular/core';
```

```
@Component({
  selector: 'app-button',
  template: `
```

```

<button [ngClass]="{ 'active': isActive }" (click)="toggleActive()">Toggle
Active</button>
,
}))
export class ButtonComponent {
  isActive = false;

  toggleActive() {
    this.isActive = !this.isActive;
  }
}

```

When the user clicks the button, the **toggleActive()** method is called, which toggles the value of the **isActive** property. When **isActive** is true, the **active** CSS class is added to the button, and when **isActive** is false, the **active** CSS class is removed.

ngStyledirective :

ngStyle is a built-in Angular directive that allows you to dynamically add or remove inline styles to an HTML element based on certain conditions. This is particularly useful when you want to apply different styles to an element based on its state or user interaction.

The **ngStyle** directive can be used in the following way:

```

<div [ngStyle]="{ 'property1': value1, 'property2': value2 }">Example</div>

```

In this example, the **ngStyle** directive applies the CSS styles **property1** and **property2** to the **div** element with their corresponding values **value1** and **value2**.

Here's a more detailed example that demonstrates how to use **ngStyle** to dynamically apply inline styles based on user interaction:

```

<button [ngStyle]="{ 'background-color': isActive ? 'green' : 'red' }"
(click)="toggleActive()">Toggle Active</button>

```

In this example, the **ngStyle** directive applies the inline style **background-color** to the **button** element based on the value of the **isActive** property. If **isActive** is true, the button's background color will be green, and if it is false, the background color will be red. The **isActive** property is initially set to false in the component:

```

import { Component } from '@angular/core';

```

```

@Component({
  selector: 'app-button',
  template: `
    <button [ngStyle]="{ 'background-color': isActive ? 'green' : 'red' }"
    (click)="toggleActive()">Toggle Active</button>
  `,
})
export class ButtonComponent {
  isActive = false;

  toggleActive() {
    this.isActive = !this.isActive;
  }
}

```

When the user clicks the button, the **toggleActive()** method is called, which toggles the value of the **isActive** property. When **isActive** is true, the button's background color will be green, and when **isActive** is false, the background color will be red.

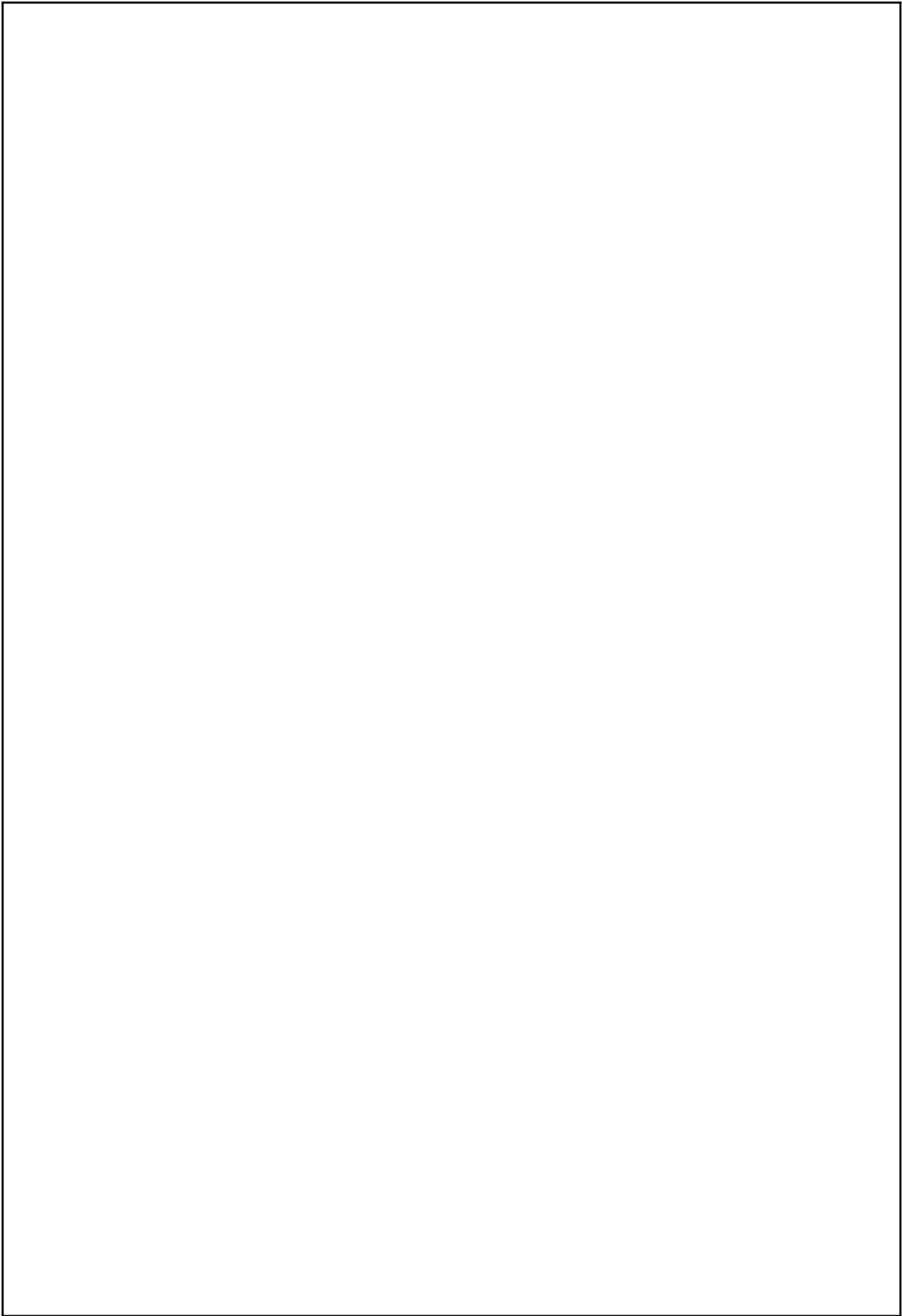
You can also use **ngStyle** in combination with other directives, such as **ngFor** or **ngIf**, to dynamically apply different inline styles to elements based on their state or data.

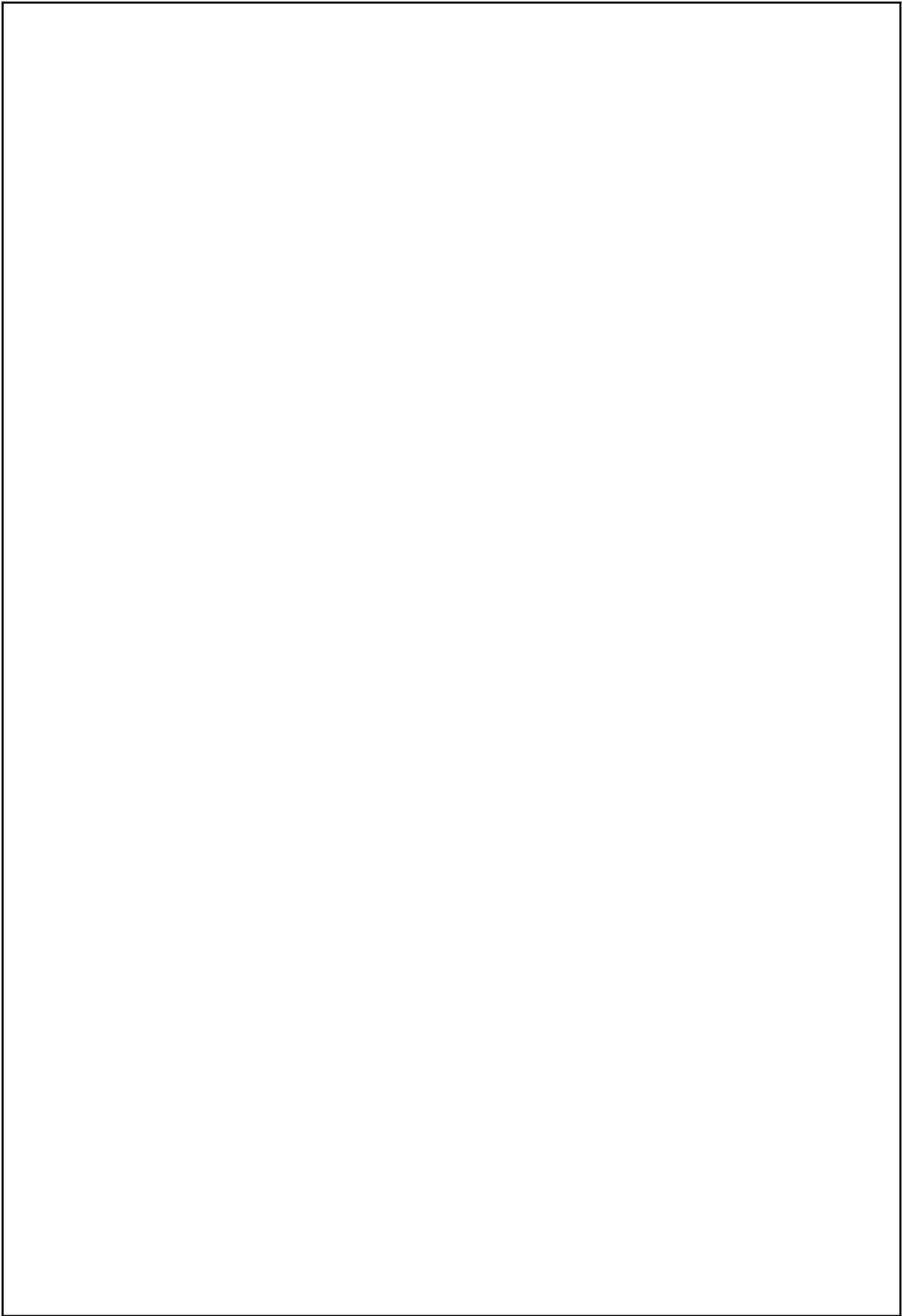
H. Resources/Equipment Required

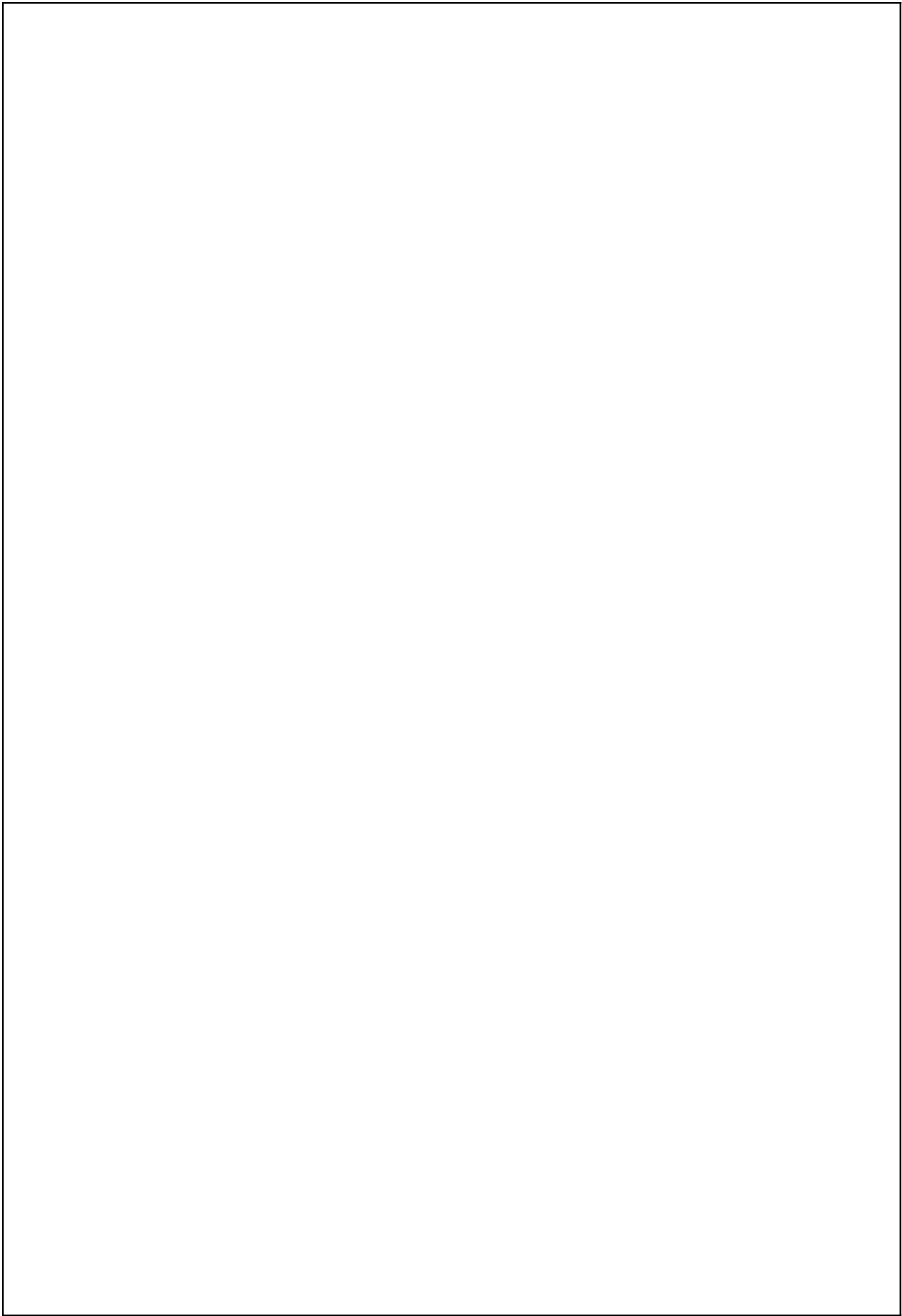
S r. N o.	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

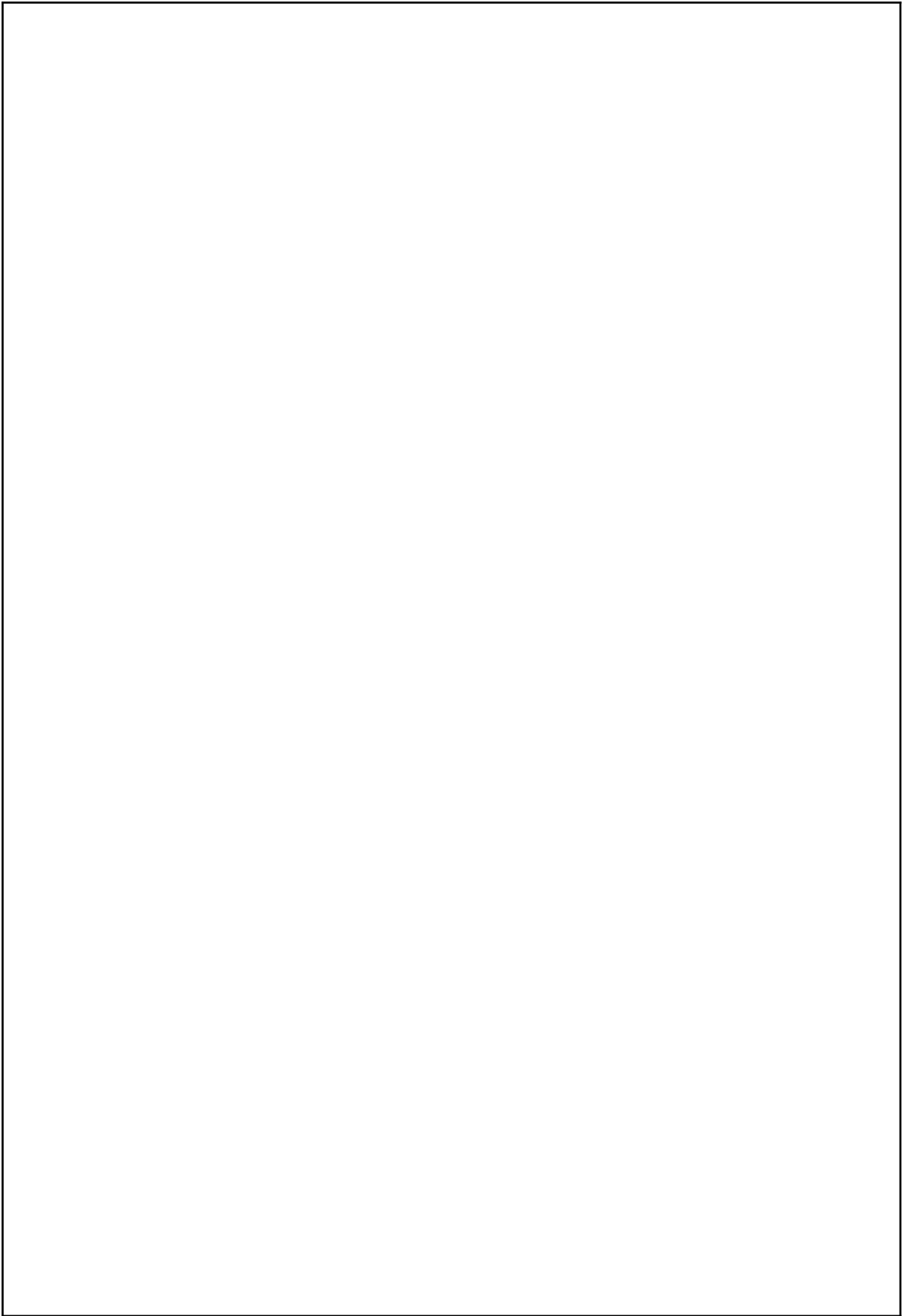
I. Program Source code Output

SOURCE CODE









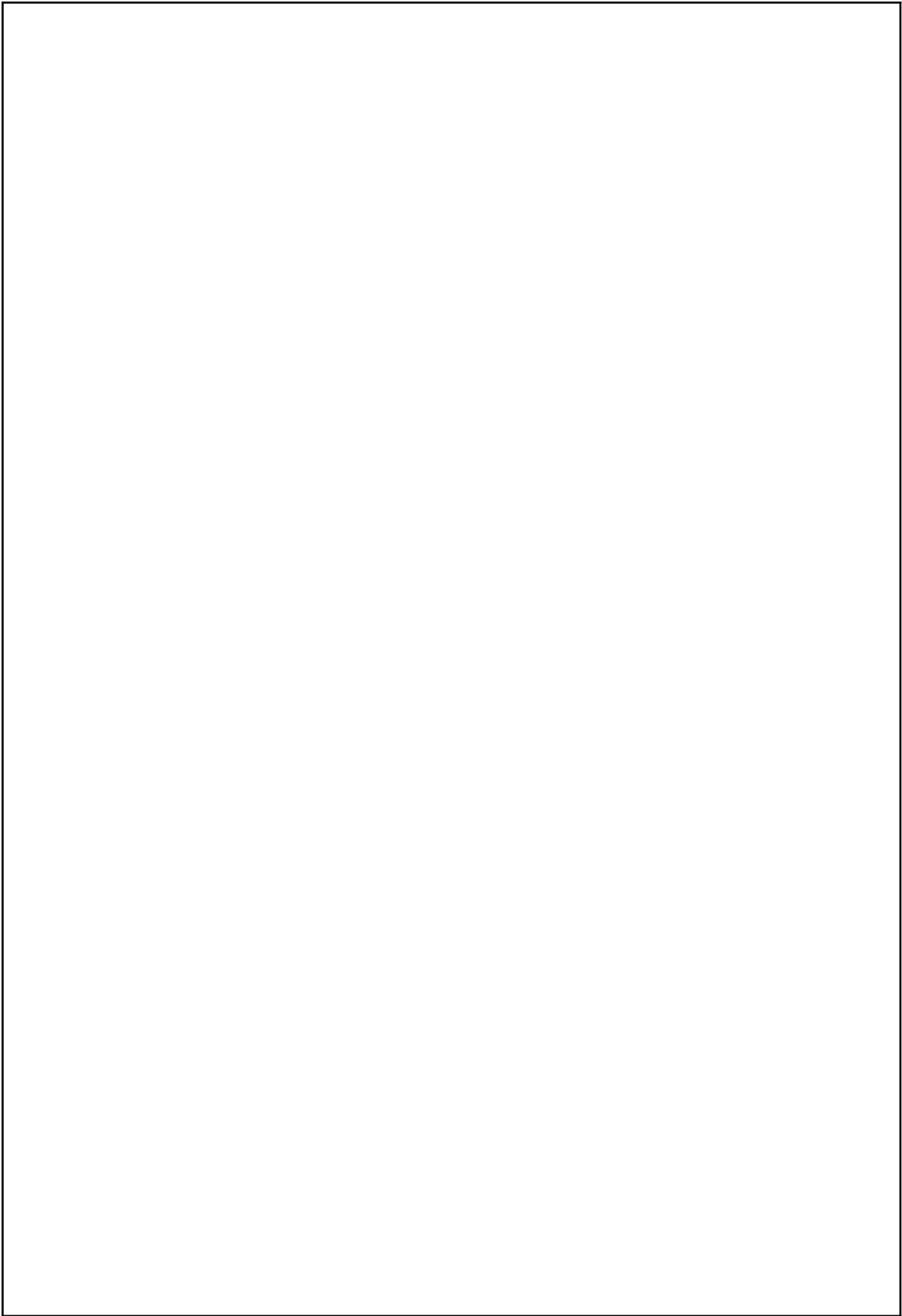
OUTPUT

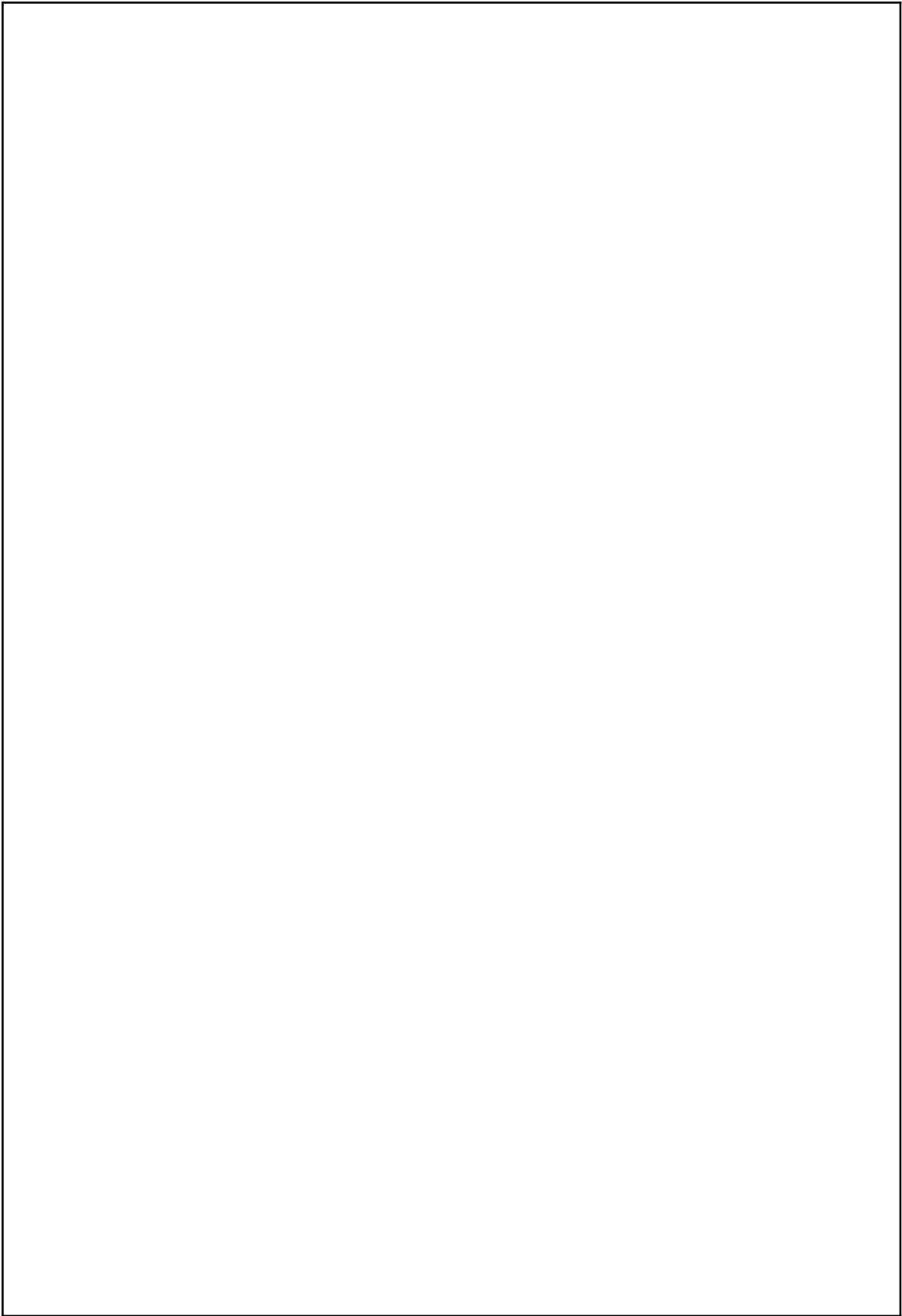
J. Practical related Exercises.

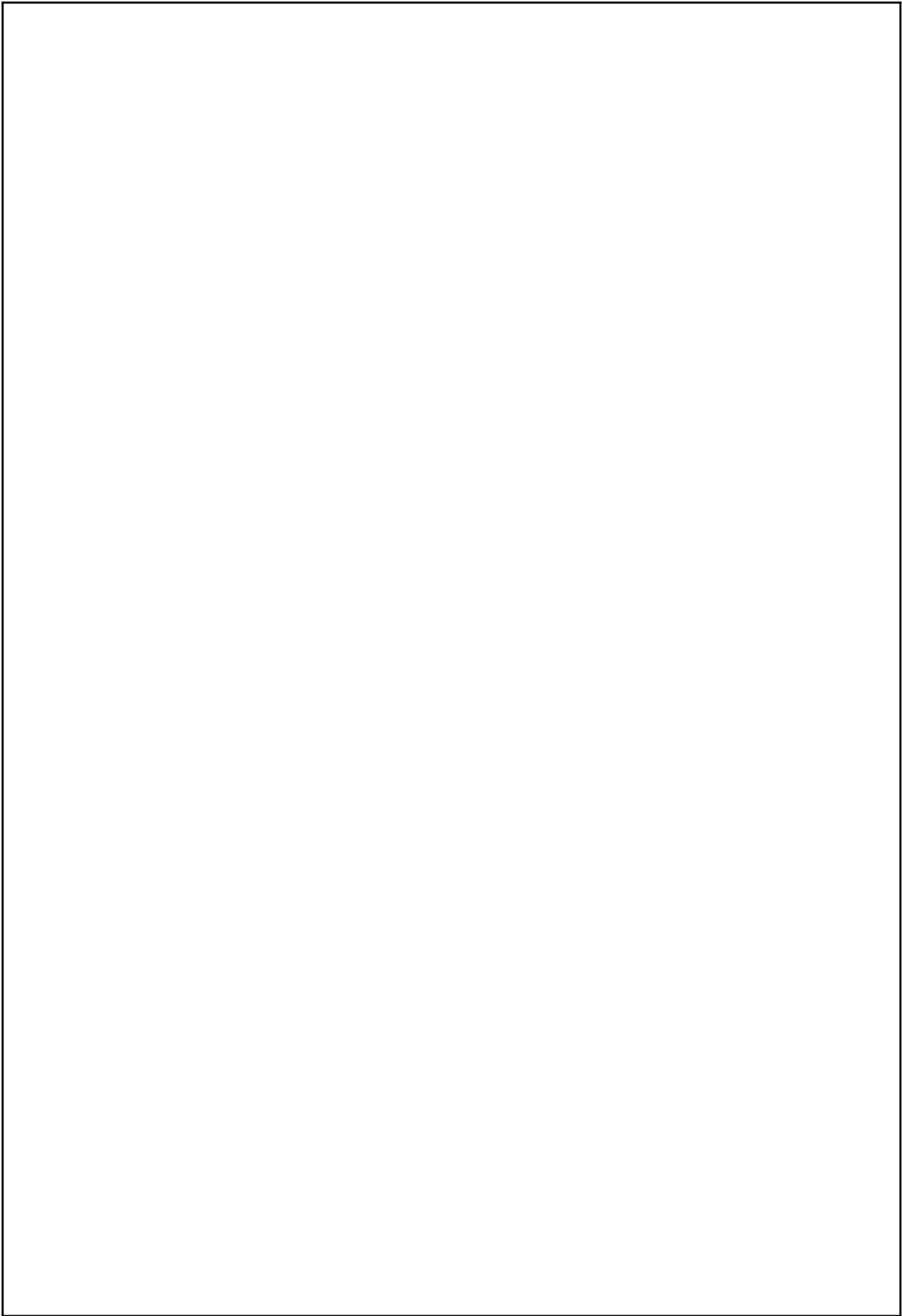
1. Design following tourist package webpage using various components of angular or teacher can assign any webpage of website.

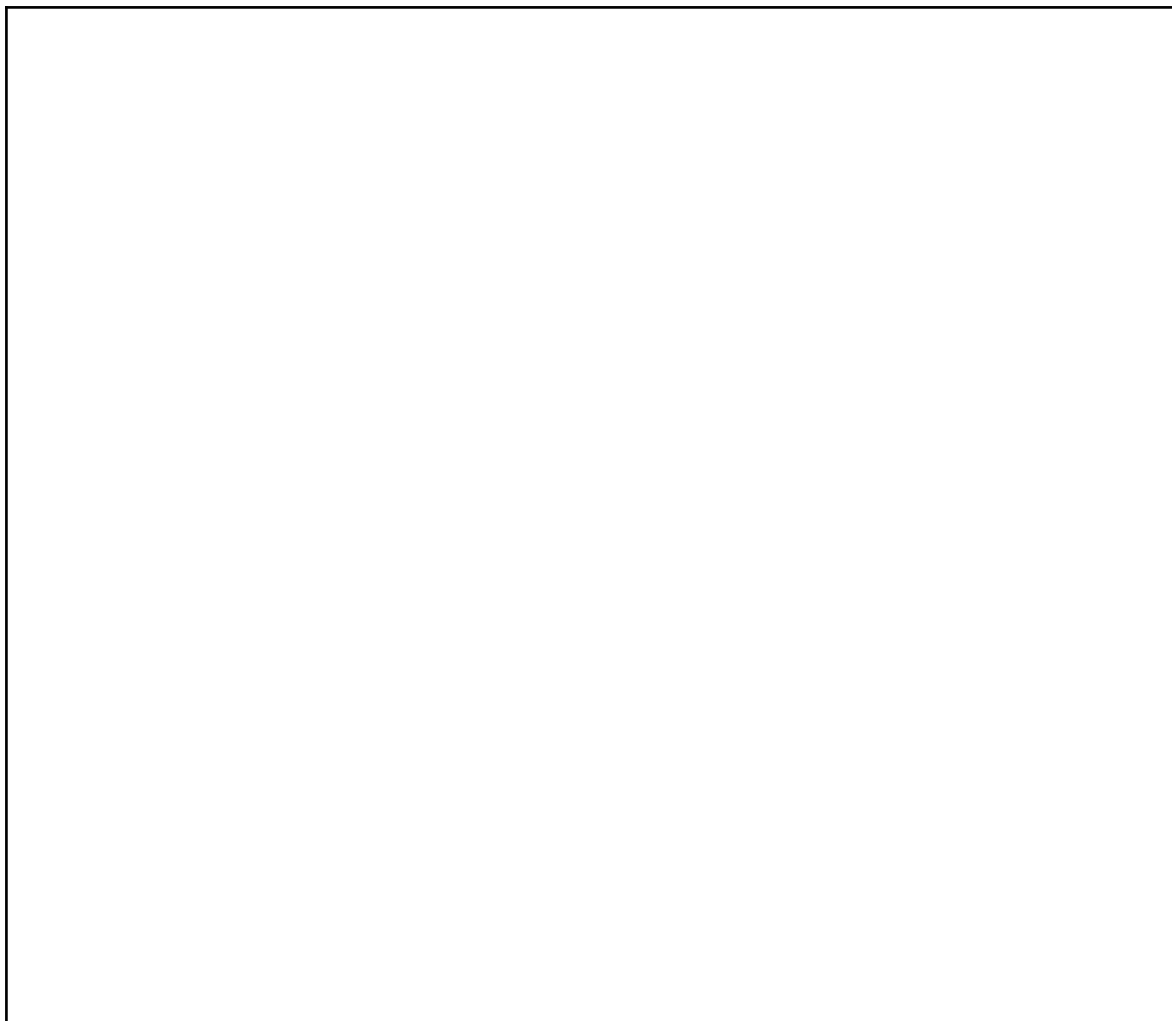


2. Create a component for header and footer that displays your department webpage information. Use the ngClass and ngStyle directive to apply different styles to the header and footer









K. References Links

1. <https://youtu.be/uxYvALNR70Y>
2. <https://www.youtube.com/watch?v=IUHId5qN0oI>
3. <https://www.youtube.com/watch?v=-duLXqM2nlo>
4. <https://www.geeksforgeeks.org/components-in-angular-8/>
5. <https://www.simplilearn.com/tutorials/angular-tutorial/angular-component-s>

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of Input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No.5: Design a web page to display student grading system in tabular format with alternate color style using ngSwitch, ngStyle Directives.

A. Objective:

Manipulate the structure of the DOM (Document Object Model) based on conditions and other logical statements.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.

2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.

3. Design/ development of solutions: Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.

4. Engineering Tools, Experimentation and Testing: Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.

5. Life-long learning: Ability to analyze individual needs and engage in updating in the context of technological changes *in field of engineering*.

C. Expected Skills to be developed based on competency:

Learner can get various skills such as understanding of basic Angular concepts, Familiarity with Type Script, Knowledge of Angular directives, Experience with Angular CLI.

D. Expected Course Outcomes(Cos)

Apply angular directives, components and pipes in different web page development.

E. Practical Outcome(PRo)

Design a web page to display student grading system in tabular format with alternate color style using ngSwitch, ngStyle Directives.

F. Expected Affective domain Outcome(ADos)

1. Follow Coding standards and practices.
2. Maintain tools and equipment.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory:

Structural directives in Angular are a type of directive that allow you to modify the structure of the DOM based on certain conditions or states. They are used to add, remove, or update elements in the DOM based on the result of an expression or the state of the application.

There are three common structural directives in Angular:

1. **ngIf:** The ngIf directive is used to conditionally add or remove an element from the DOM based on a given expression. If the expression is true, the element is added to the DOM, and if it is false, the element is removed from the DOM.
2. **ngFor:** The ngFor directive is used to repeat a section of HTML code for each item in an array or collection. It can be used to loop through an array of objects, an array of strings, or any other iterable object.
3. **ngSwitch:** The ngSwitch directive is used to conditionally display content based on a given expression. It allows you to define a set of possible values and associate each value with a template to render.

Structural directives are identified by the prefix "ng" followed by the directive name. They are enclosed in square brackets and are typically used in combination with other directives and HTML tags to create dynamic and responsive user interfaces.

***ngIf Directory Structure**

*ngIf is a structural directive in Angular that allows you to conditionally render or remove HTML elements based on an expression. The directive evaluates the expression passed to it and renders the element if the expression is truthy, otherwise it removes the element from the DOM.

In addition to *ngIf, there is also an *ngIf-else directive that allows you to specify an alternative template to render when the expression evaluates to false. The syntax for *ngIf-else is as follows:

```
<div *ngIf="condition; else elseBlock">
<!-- content to show when condition is true -->
</div>
<ng-template #elseBlock>
<!-- content to show when condition is false -->
</ng-template>
```

In this example, the **div** element will be rendered if the **condition** expression is truthy. If **condition** is falsy, the **div** element will be removed from the DOM and the **elseBlock** template will be rendered instead.

The **elseBlock** template is defined using the **ng-template** element with a template reference variable of **#elseBlock**. This allows us to refer to the template later in the ***ngIf-else** directive.

You can also use multiple ***ngIf-else** directives to conditionally render different templates based on different expressions. Here's an example:

```
<div *ngIf="condition1; else elseBlock1">
<!-- content to show when condition1 is true -->
</div>
<ng-template #elseBlock1>
<div *ngIf="condition2; else elseBlock2">
<!-- content to show when condition1 is false and condition2 is true -->
</div>
<ng-template #elseBlock2>
<!-- content to show when both condition1 and condition2 are false -->
</ng-template>
</ng-template>
```

In this example, we first check **condition1**. If it is true, we render the first **div** element. If it is false, we check **condition2**. If **condition2** is true, we render the second **div** element. If both **condition1** and **condition2** are false, we render the content in the second **ng-template**.

ngFor Directory

***ngFor** is a structural directive in Angular that iterates over a collection and creates a template for each item in the collection. It is commonly used to render a list of items or to repeat a section of a template.

The basic syntax of ***ngFor** is:

```
*ngFor="let item of collection"
```

where **item** is a variable that represents each item in the collection, and **collection** is an array or any iterable object.

Here are some examples of how ***ngFor** can be used in Angular:

1. Rendering a list of items

```
<ul>
<li *ngFor="let item of items">{{ item }}</li>
</ul>
```

```
items=['apple', 'banana', 'orange']
```

In this example, **items** is an array of strings that is declared in ts file, and ***ngFor** creates a **li** element for each item in the array.

2. Looping over an array of objects

```
<table>
<tr *ngFor="let user of users">
<td>{{ user.name }}</td>
<td>{{ user.age }}</td>
</tr>
</table>
```

```
users = [
  {
    name: 'chintan',
    age: 36
  },
  {
    name: 'mohit',
    age: 45
  },
  {
    name: 'hiral',
    age: 25
  },
]
```

In this example, **users** is an array of objects that is declared in ts file, and ***ngFor** creates a **tr** element for each object in the array. The **td** elements contain the **name** and **age** properties of each object.

3: Using index for conditional rendering

```
<div *ngFor="let item of items; index as i">
<div *ngIf="i % 2 == 0" class="even-item">{{ item }}</div>
```

```
<div *ngIf="i % 2 !== 0" class="odd-item">{{item}}</div>
</div>
```

In this example, we are using the "index as i" syntax to access the current index of the loop iteration and use it for conditional rendering. We are displaying each item on a new line, but alternating the background color based on whether the index is even or odd.

Overall, `*ngFor` with index is a powerful tool in Angular that allows developers to loop through collections and access the current index of the loop iteration for various use cases.

***ngSwitch**

The `*ngSwitch` directive is used in Angular to conditionally display content based on a specified expression. It's similar to a switch statement in programming languages. Here's an example of how to use `*ngSwitch` in an Angular template file:

```
<div [ngSwitch]="dayOfWeek">
<div *ngSwitchCase="Monday">It's Monday!</div>
<div *ngSwitchCase="Tuesday">It's Tuesday!</div>
<div *ngSwitchCase="Wednesday">It's Wednesday!</div>
<div *ngSwitchCase="Thursday">It's Thursday!</div>
<div *ngSwitchCase="Friday">It's Friday!</div>
<div *ngSwitchCase="Saturday">It's Saturday!</div>
<div *ngSwitchCase="Sunday">It's Sunday!</div>
<div *ngSwitchDefault>Invalid day of the week!</div>
</div>
```

In the above example, we have a div with the attribute `[ngSwitch]="dayOfWeek"`, where **dayOfWeek** is a property in the component class. Inside this div, we have several divs that use the `*ngSwitchCase` directive to display different content based on the value of **dayOfWeek**. The `*ngSwitchDefault` directive is used to specify the default case if none of the `*ngSwitchCase` expressions match.

```
import { Component } from '@angular/core';
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  dayOfWeek = 'Monday';
```



```
}

```

In the component class, we have a property called **dayOfWeek** that is set to '**Monday**'. Based on this value, the template will display the div with the ***ngSwitchCase="'Monday'"** directive.

H. Resources/Equipment Required

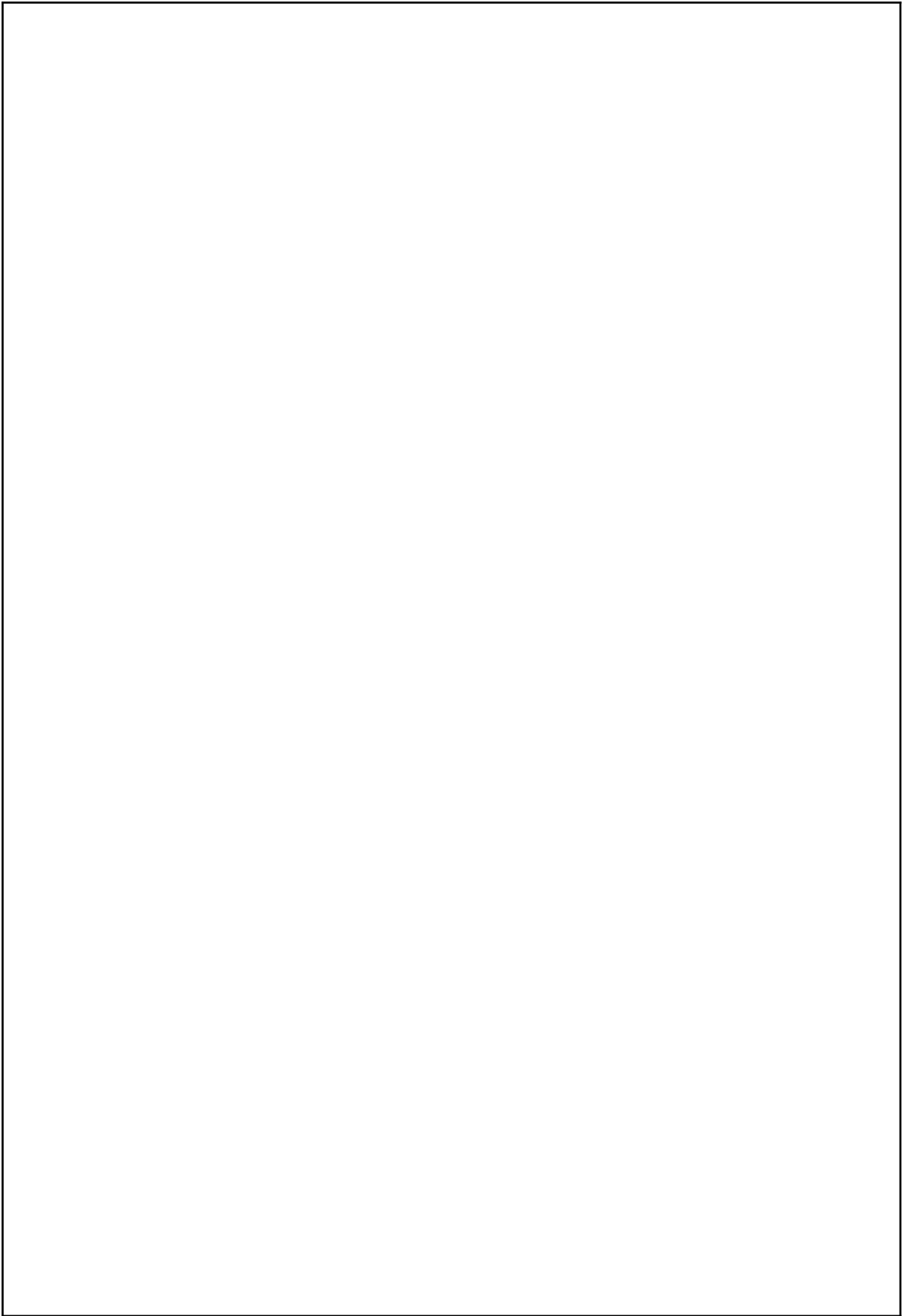
S r. N o.	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

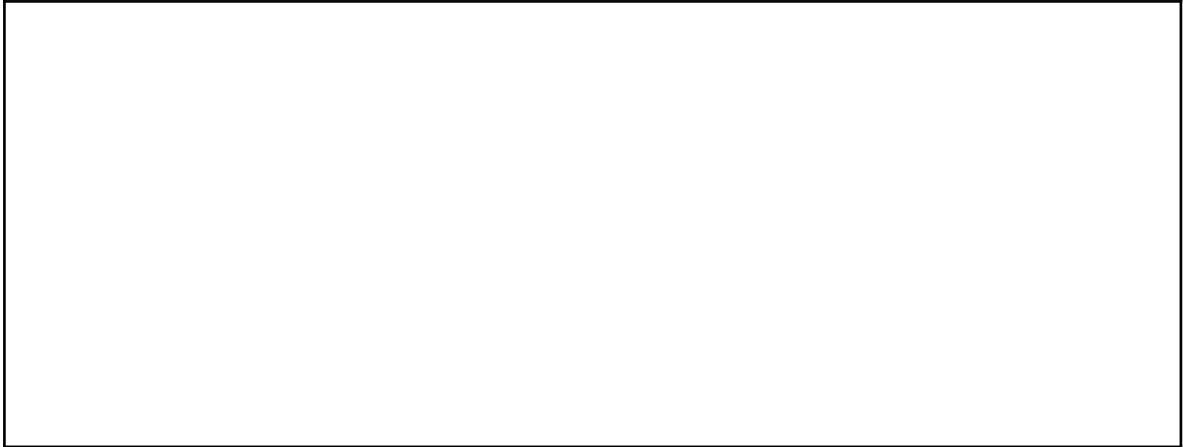
I. Program Source code Output

SOURCE CODE

A large, empty rectangular box with a thin black border, occupying the majority of the page. It is intended for the main content of the document.

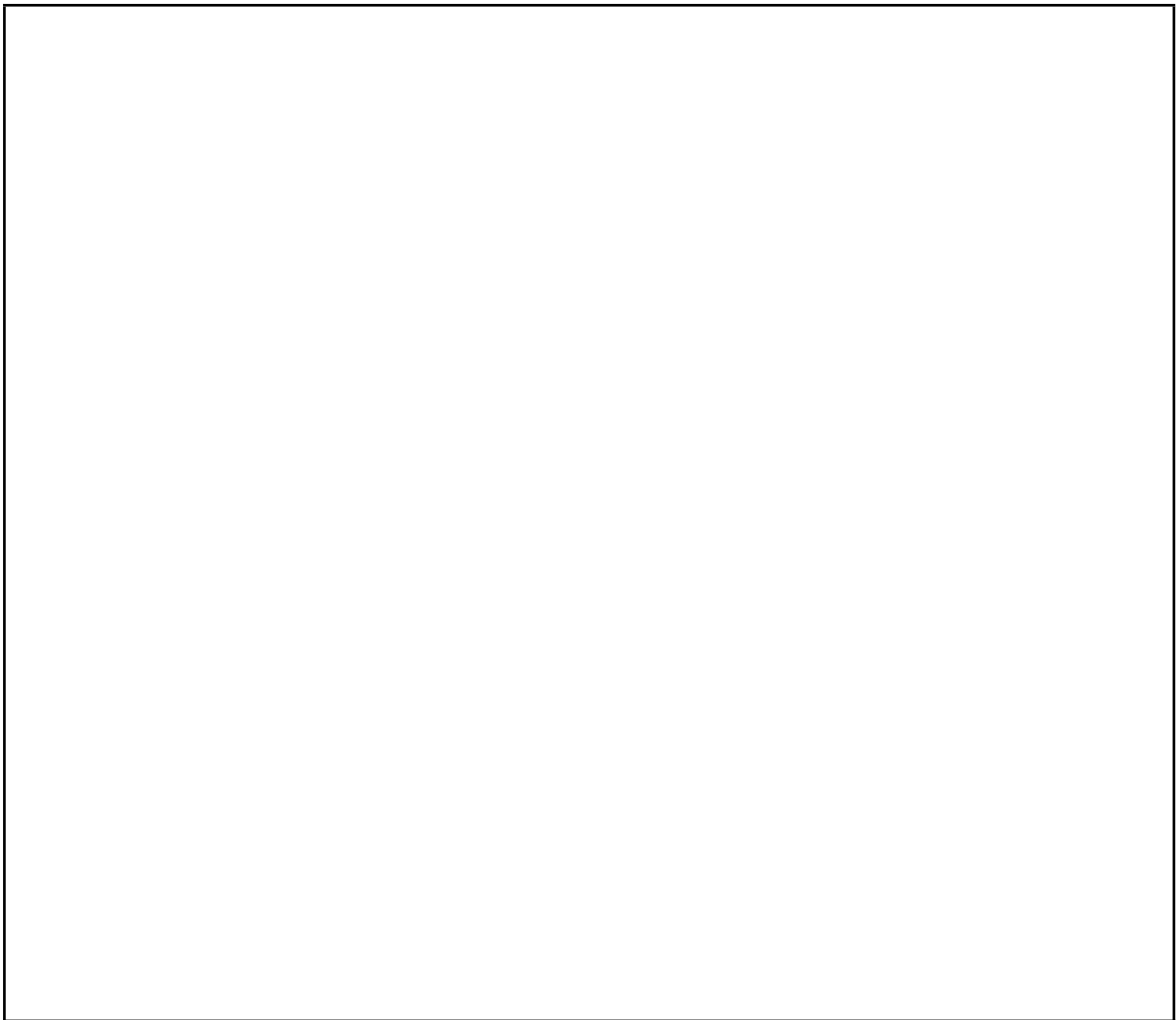
OUTPUT

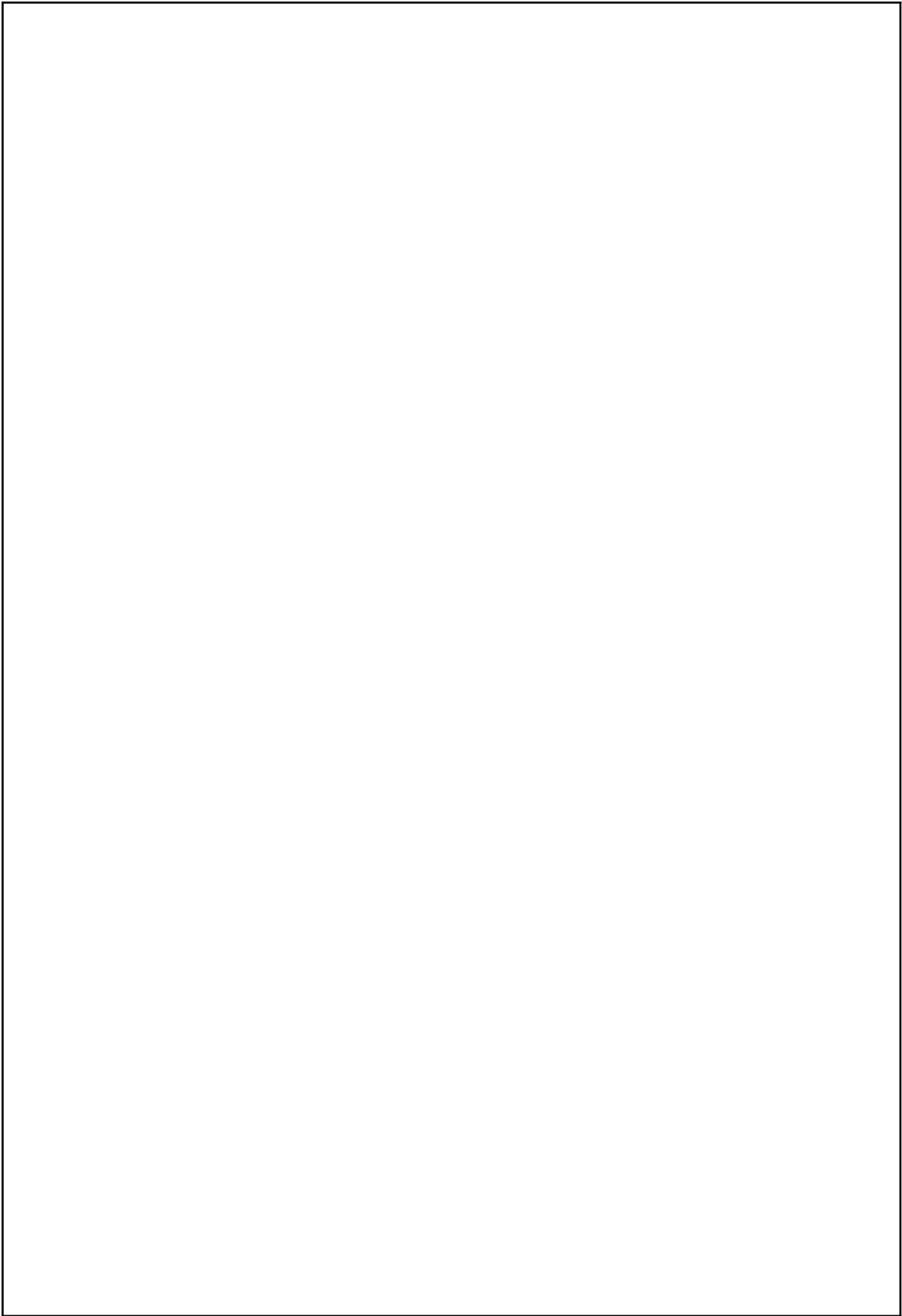


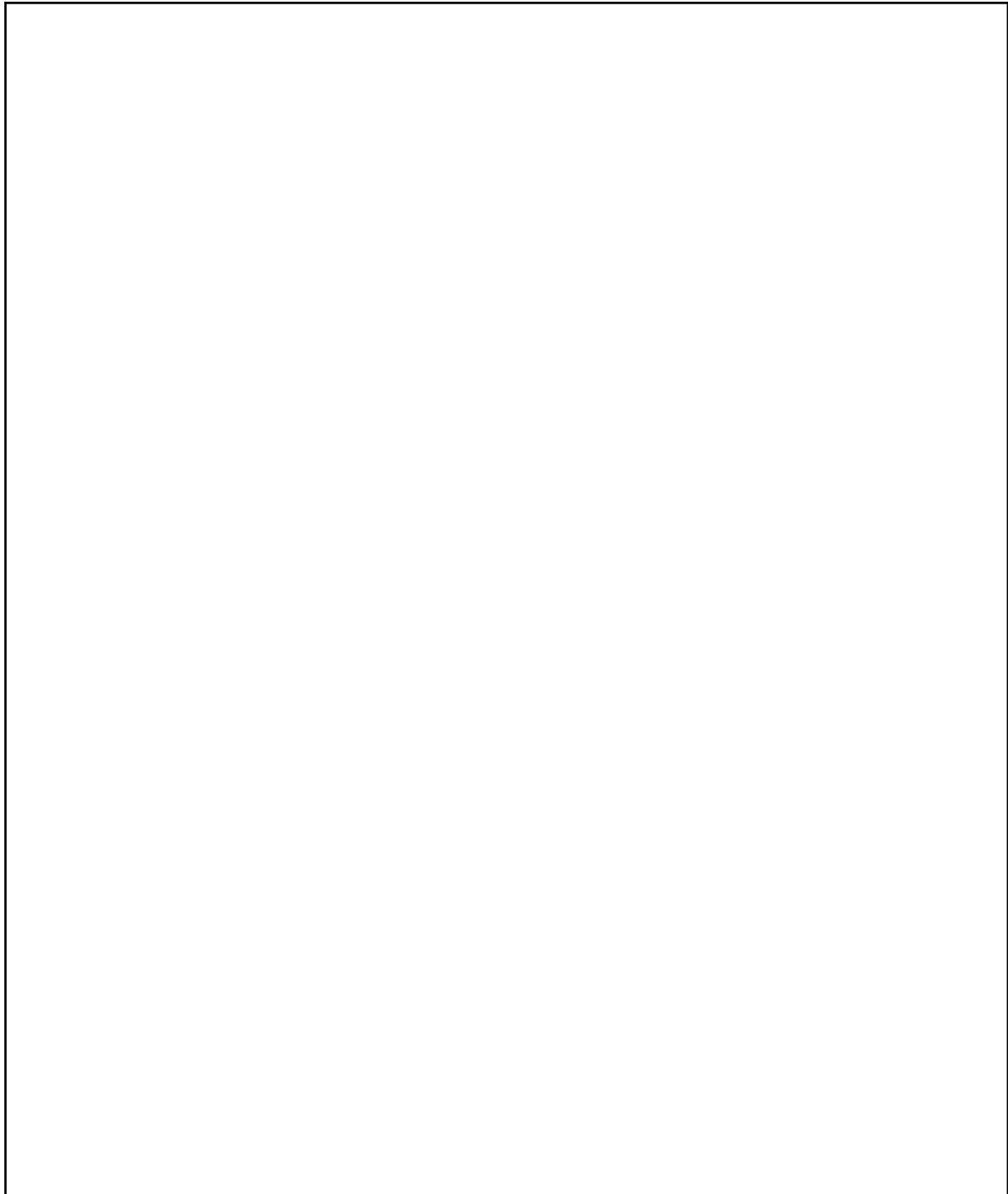


J. Practical related Exercises.

1. Design webpage to displaying different images based on a selected option.
2. Design webpage using *ngIf,*ngFor,*ngswitch and ngStyle Directives.







K. References Links

1. <https://www.geeksforgeeks.org/structural-directives-in-angular/>
2. <https://docs.angular.io/guide/structural-directives>
3. <https://www.tektutorialshub.com/angular/angular-ngswitch-directive/>
4. <https://youtu.be/ILgnVV3IDI0>
5. <https://youtu.be/jJqwb44B974>

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of Input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No.6: Design component to perform following tasks

- [1] To Add or Remove number of students using textbox and button controls and display it in tabular structure format.**
- [2] Give row level remove button option to student table and record should be deleted when click on it.**

A. Objective:

The learner will help to manipulate data on web page based on event , conditions and generate desired outputs.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.

2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.

3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.

4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.

5. **Life-long learning:** Ability to analyze individual needs and engage in updating in the context of technological changes *in field of engineering*.

C. Expected Skills to be developed based on competency:

Learner should have knowledge of how to Dynamic rendering of lists, efficiently adding and removing elements on web page and Familiarity with template syntax.

D. Expected Course Outcomes(Cos)

Apply angular directives, components and pipes in different web page development.

E. Practical Outcome(PRo)

Learner will be able to Add or Remove number of students using textbox and button controls and display it in tabular structure format along with give row level remove button option to student table and record should be deleted when click on it.

.

F. Expected Affective domain Outcome(ADos)

1. Follow Coding standards and practices.
2. Maintain tools and equipment.
3. Follow safety practices.

4. Follow ethical practices

G. Prerequisite Theory:

Template reference variable

Template reference variable is a variable that you can assign to a template element or component using the # symbol. This allows you to reference the element or component in your template code and in your component code.

Here's an example of using a template reference variable to reference an input element in a template:

```
<input type="text" #nameInput>  
  
<button (click)="greet(nameInput.value)">Greet</button>
```

In this template, we're using the **#nameInput** syntax to create a template reference variable that we can use to reference the **input** element. We're also using an event binding to call a **greet** method on our component when a button is clicked.

In our component, we can access the value of the input element using the template reference variable:

```
import { Component } from '@angular/core';
```

```
@Component({  
  selector: 'app-greeter',  
  templateUrl: './greeter.component.html',  
  styleUrls: ['./greeter.component.css']  
})  
export class GreeterComponent {  
  greet(name: string) {  
    alert(`Hello, ${name}!`);  
  }  
}
```

In this component, we've defined a **greet** method that takes a **name** argument and displays an alert message with the name. When the button in the template is clicked, the **greet** method is called with the value of the **nameInput** template reference variable as the argument.

PUSH method

The **push** method in Angular is used to add elements to an array. It's a built-in method of the JavaScript **Array** object, and it can be used in Angular components to manipulate arrays and update the view.

Here's an example of using the **push** method in an Angular component:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-product-list',
  templateUrl: './product-list.component.html',
  styleUrls: ['./product-list.component.css']
})
export class ProductListComponent {
  products: string[] = [];

  addProduct(productName: string) {
    this.products.push(productName);
  }
}
```

In this example, we've defined a **products** array in our component and initialized it to an empty array. We've also defined an **addProduct** method that takes a **productName** argument and adds it to the **products** array using the **push** method.

Now, in our component's template, we can use **ngFor** to iterate over the **products** array and display each product in a list:

```
<ul>
<li *ngFor="let product of products">{{ product }}</li>
</ul>

<input type="text" #productName>
```

```
<button (click)="addProduct(productName.value)">Add Product</button>
```

In this template, we're using **ngFor** to iterate over the **products** array and display each product as a list item. We're also using an input and a button to capture new product names and call the **addProduct** method when the button is clicked.

When the **addProduct** method is called, it uses the **push** method to add the new product to the **products** array. Because the view is bound to the **products** array using **ngFor**, the view is automatically updated to show the new product in the list.

Splice method

The splice method in Angular is used to add or remove elements from an array at a specified index. It's a built-in method of the JavaScript Array object, and it can be used in Angular components to manipulate arrays and update the view.

Here's an example of using the splice method in an Angular component:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-product-list',
  templateUrl: './product-list.component.html',
  styleUrls: ['./product-list.component.css']
})
export class ProductListComponent {
  products: string[] = ['apple', 'banana', 'orange'];

  removeProduct(index: number) {
    this.products.splice(index, 1);
  }
}
```

In this example, we've defined a **products** array in our component and initialized it with three products. We've also defined a **removeProduct** method that takes an **index** argument and removes the product at that index from the **products** array using the **splice** method.

Now, in our component's template, we can use **ngFor** to iterate over the **products** array and display each product in a list. We can also use a button to call the **removeProduct** method with the index of the product we want to remove:

```

<ul>
<li *ngFor="let product of products; let i = index">{{ product }}
<button (click)="removeProduct(i)">Remove</button>
</li>
</ul>

```

In this template, we're using **ngFor** to iterate over the **products** array and display each product as a list item. We're also using a button to call the **removeProduct** method with the index of the product we want to remove.

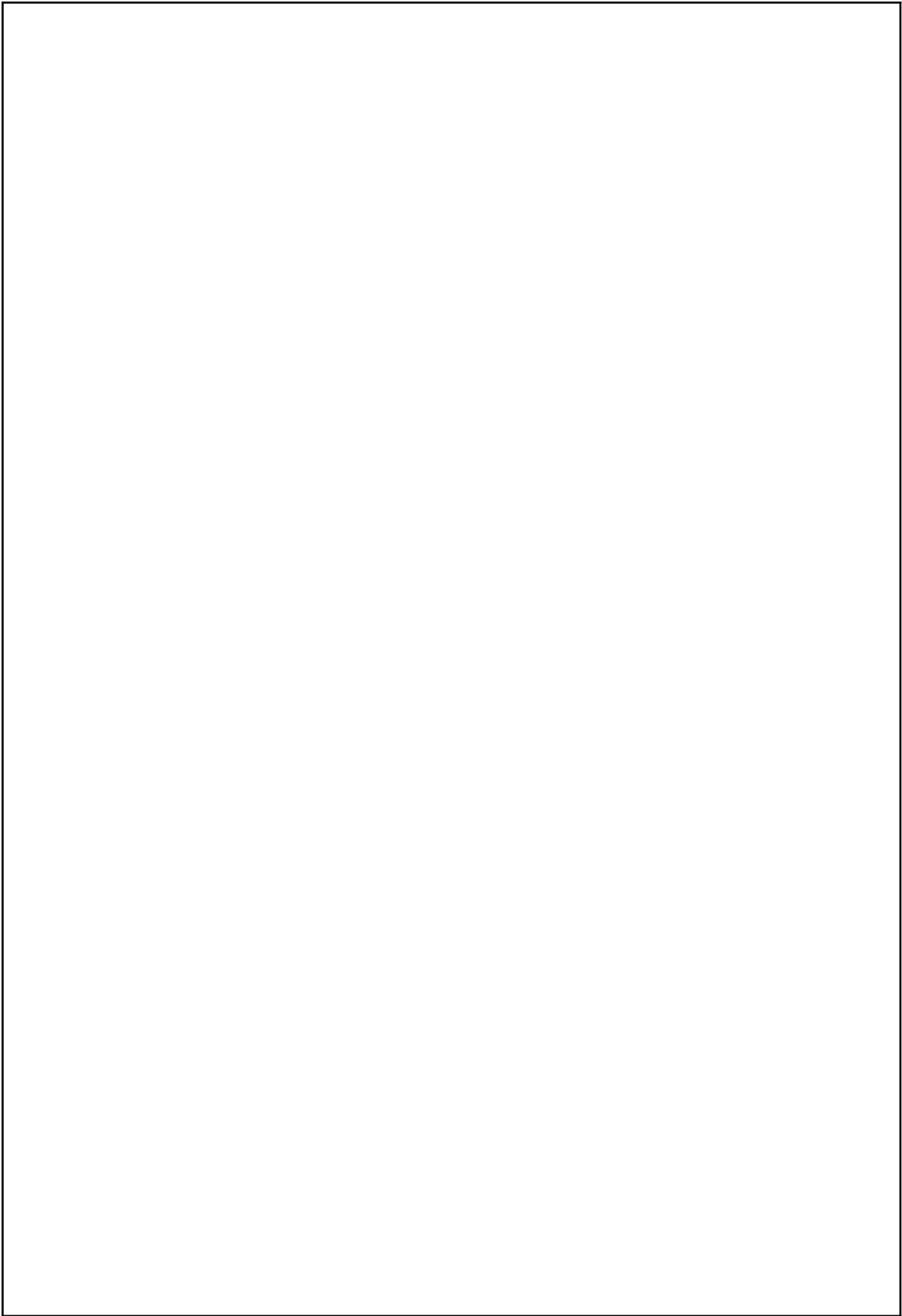
When the **removeProduct** method is called, it uses the **splice** method to remove the product at the specified index from the **products** array. Because the view is bound to the **products** array using **ngFor**, the view is automatically updated to remove the product from the list.

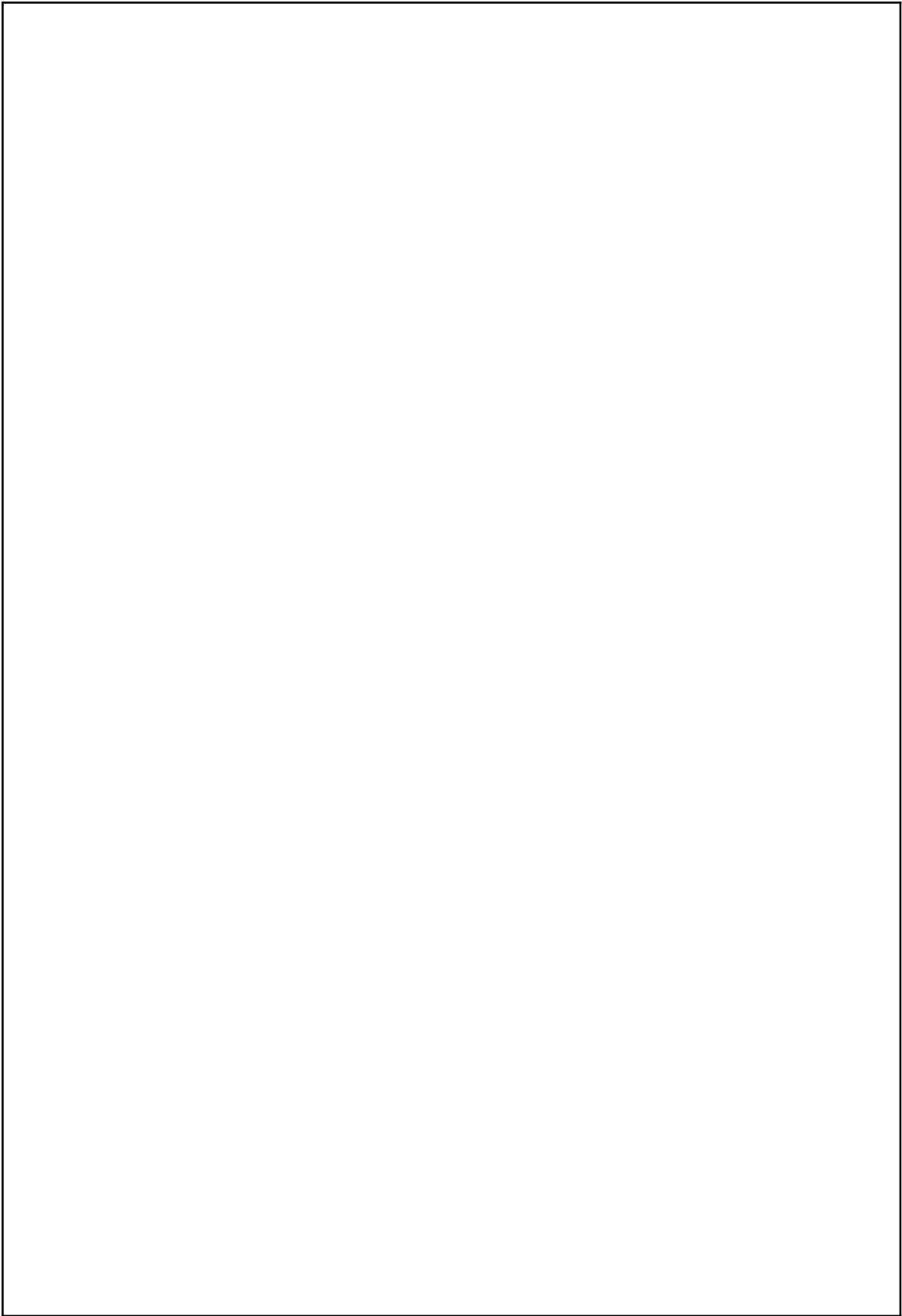
H. Resources/Equipment Required

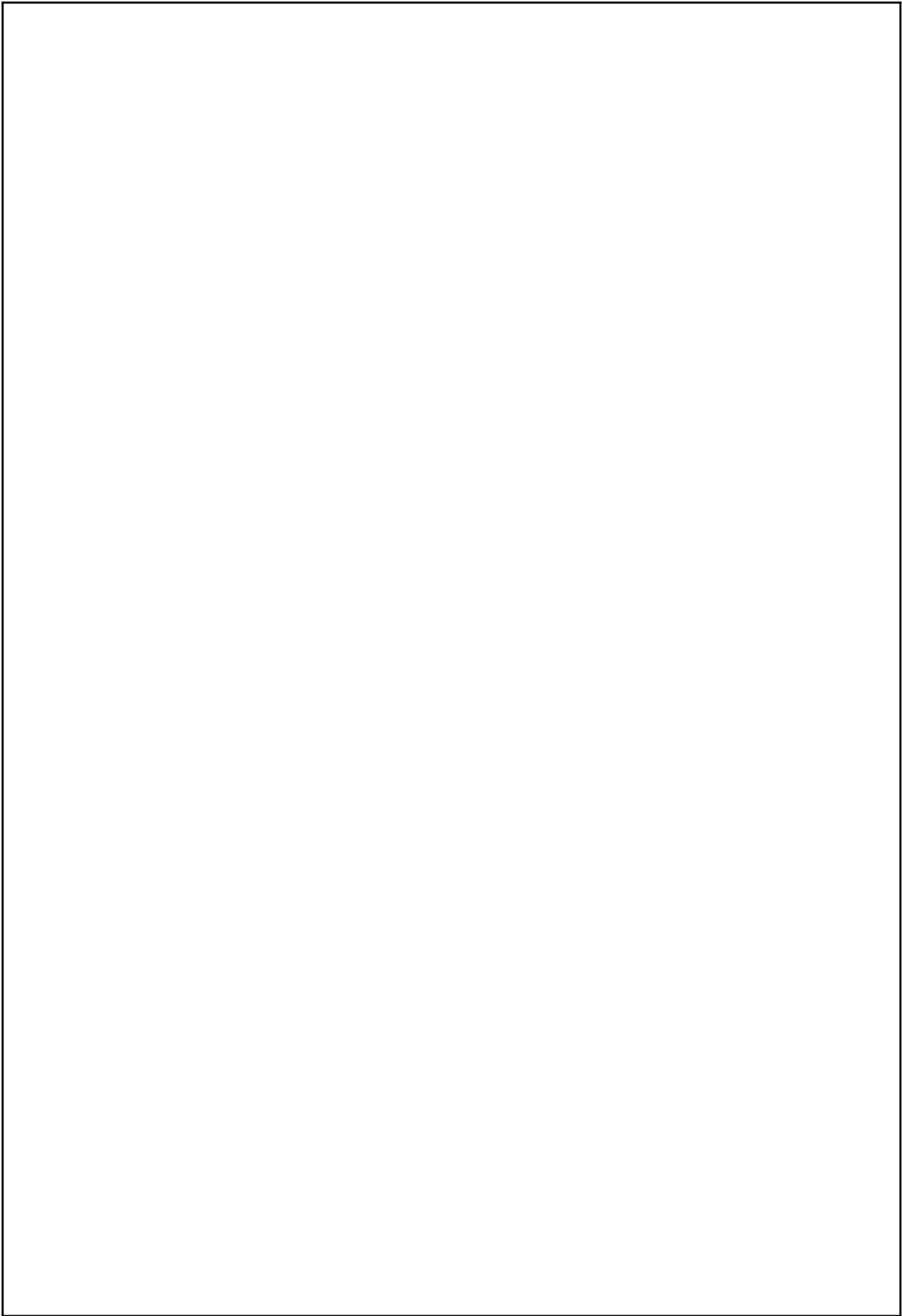
S r. N o.	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

I. Program Source code Output

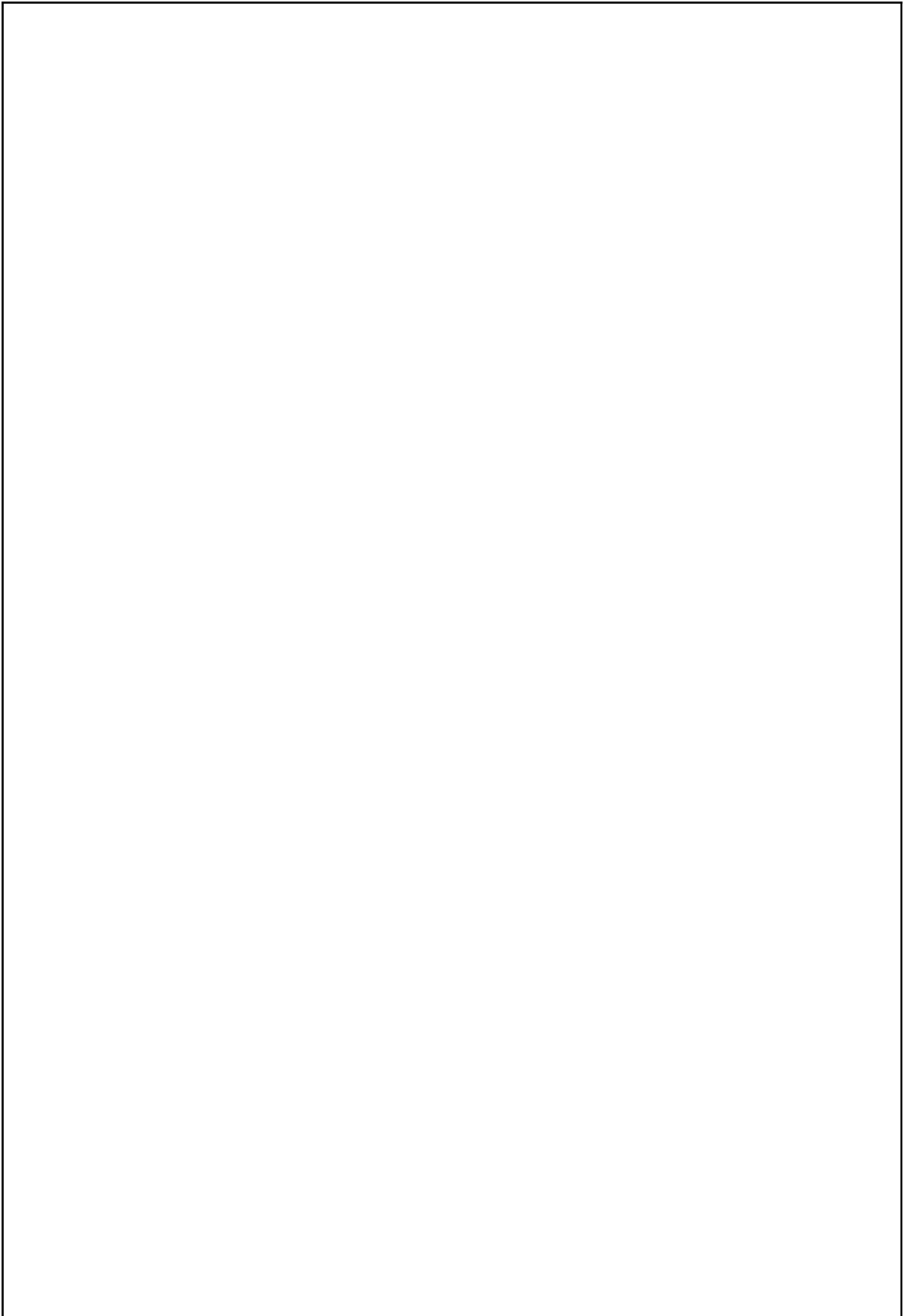
SOURCE CODE







OUTPUT

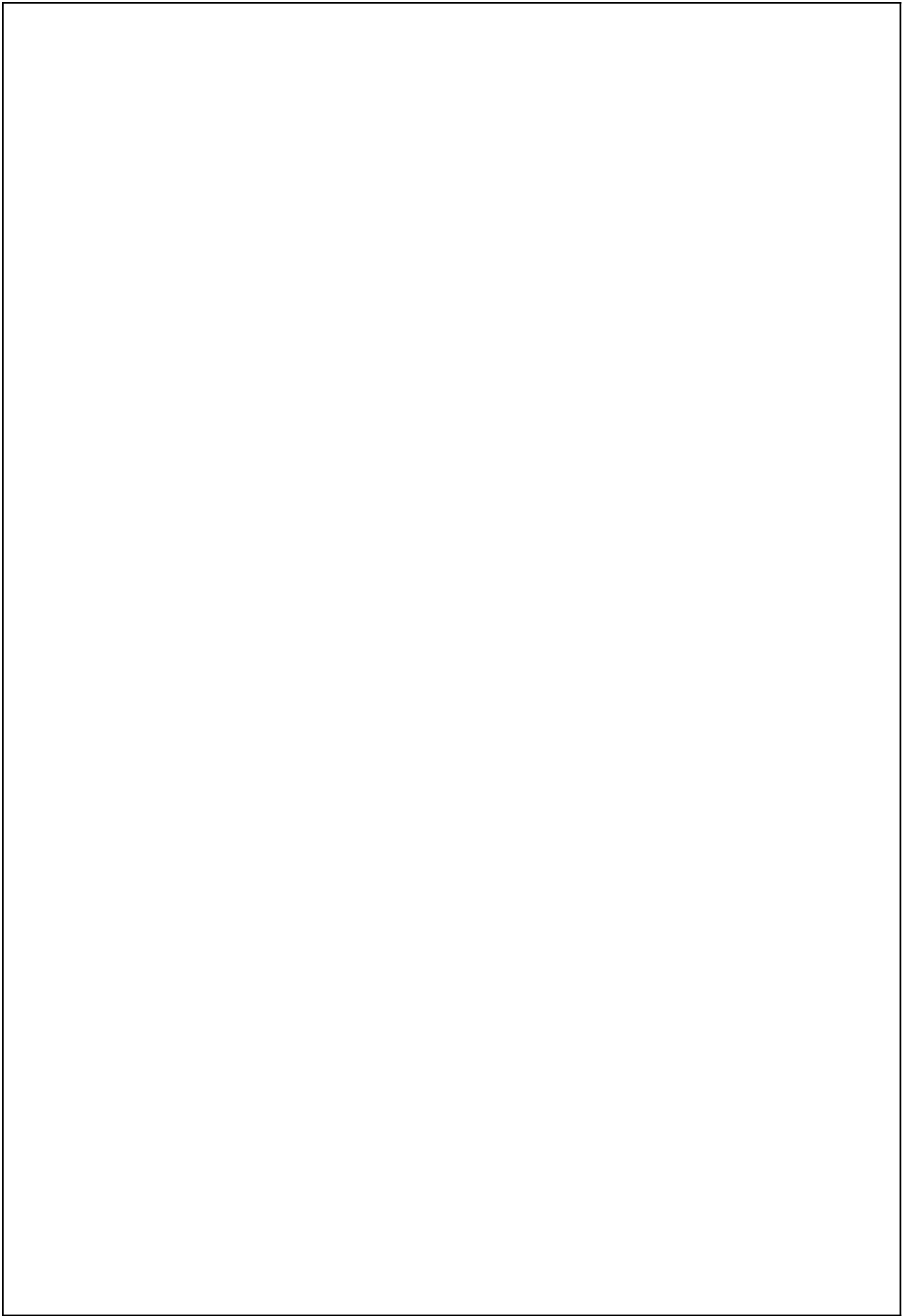


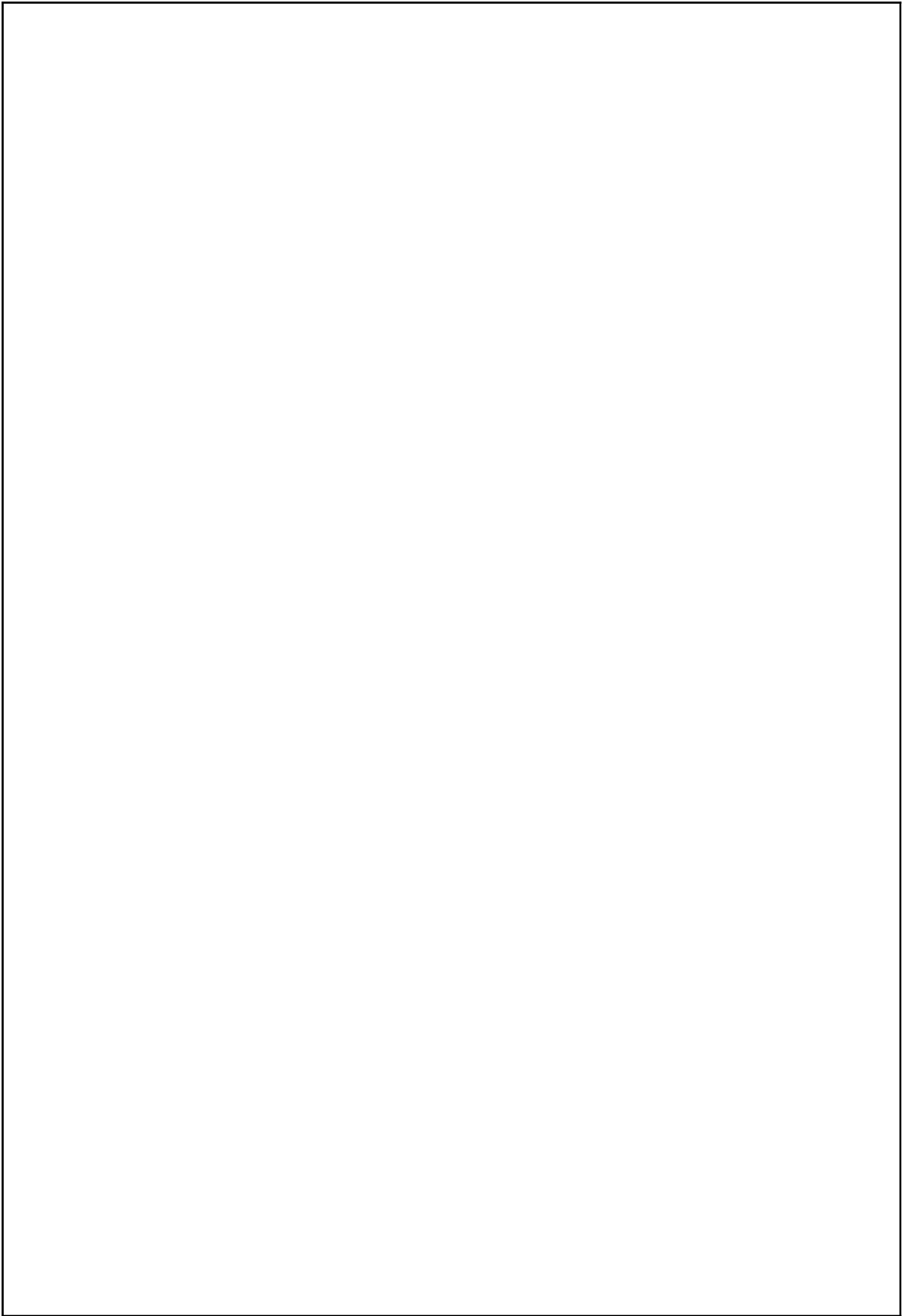


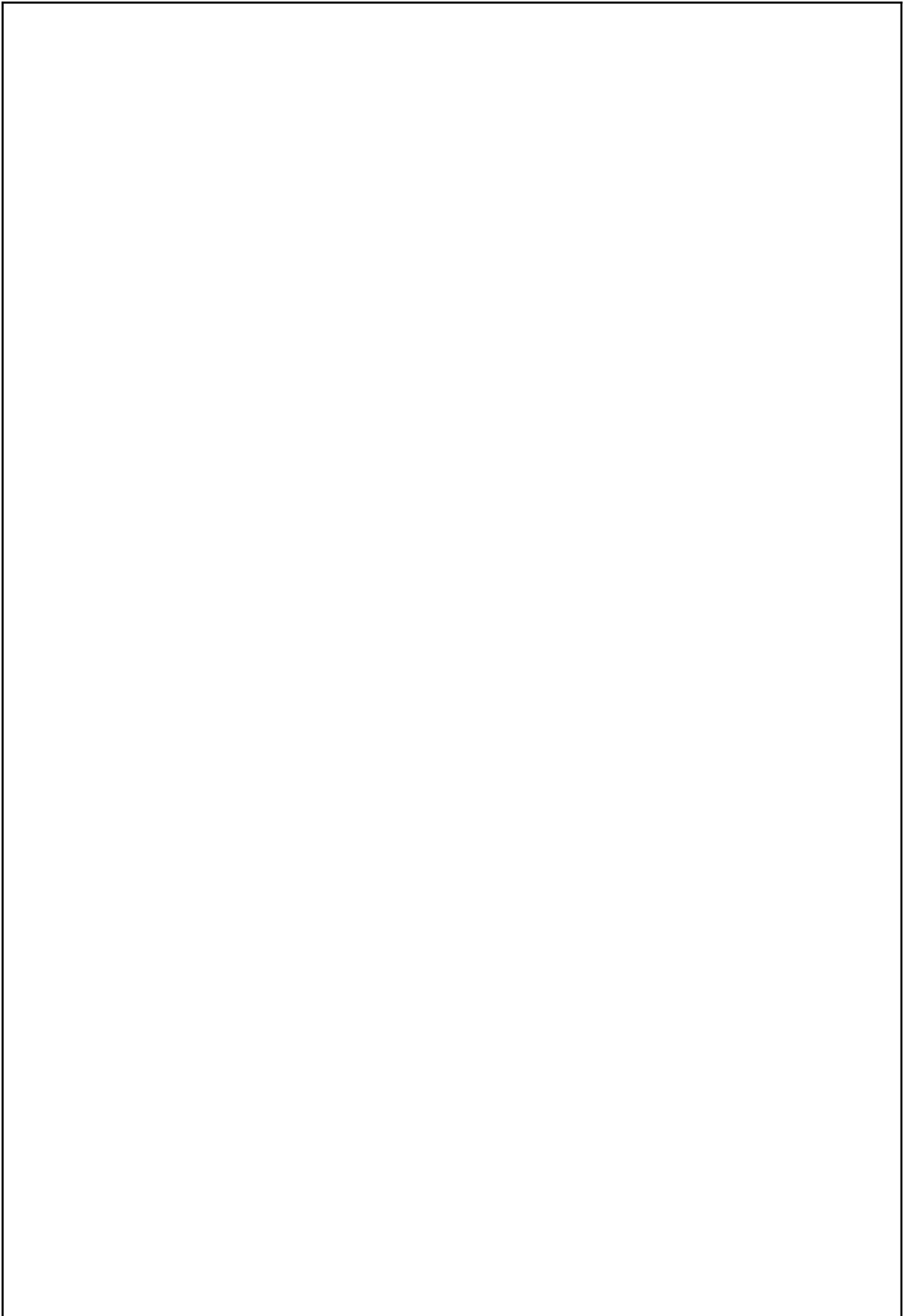
J. Practical related Exercises.

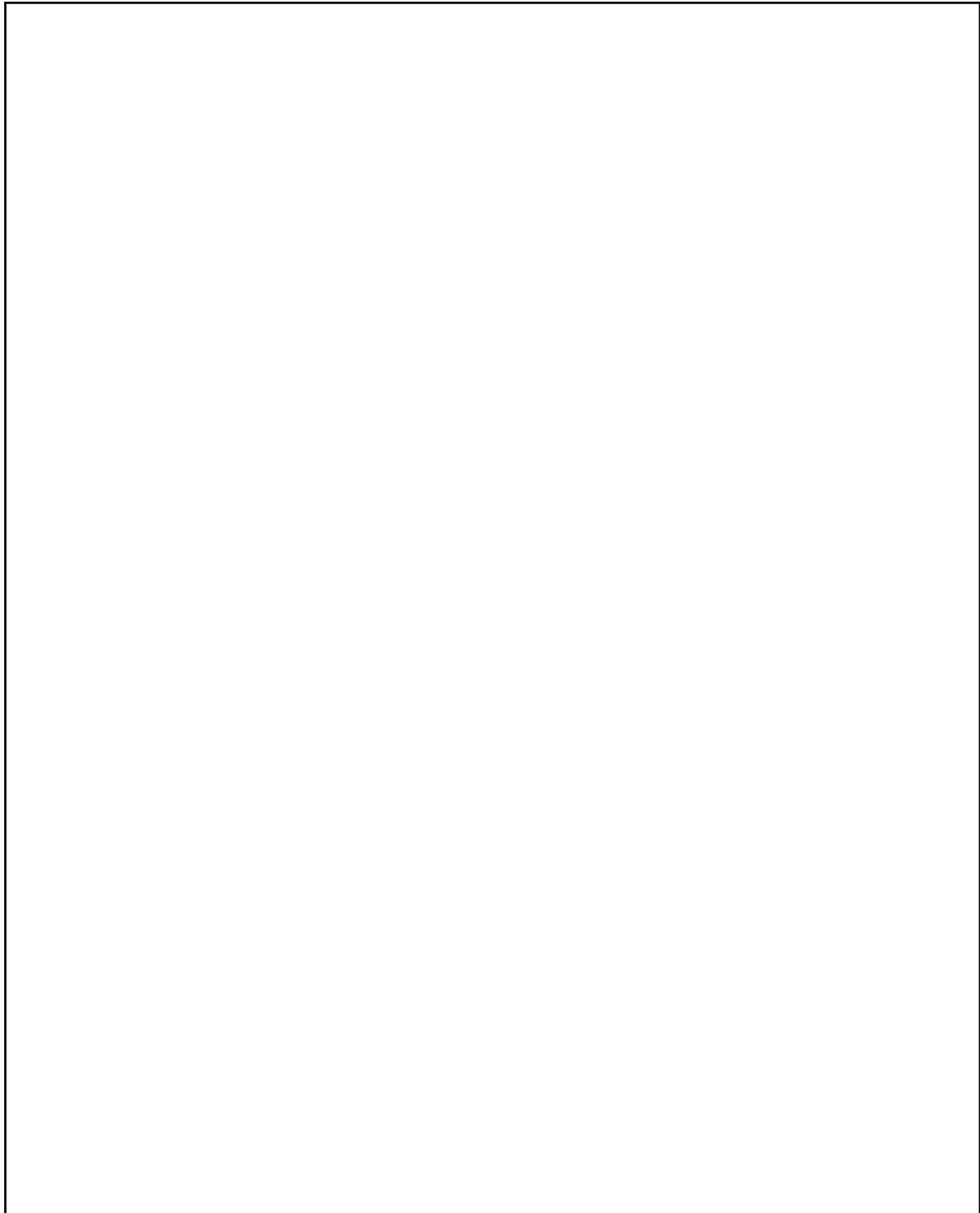
1. Design a component to add maximum 10 students name from user input and display data in tabular format. When user cross maximum limit and add new student name, there must be display message as “Maximum limit Reached”.
2. Design a component to display product list with removeitem facilities. When user click on remove item button, there should be removed current item and immediate next item from product list.











K. References Links

1. https://www.tutorialspoint.com/typescript/typescript_array_splice.htm
2. <https://www.educative.io/answers/how-to-use-splice-in-angular>
3. https://www.tutorialspoint.com/typescript/typescript_array_push.htm
4. <https://youtu.be/dMf7LPnFBEA>

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of Input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No 7: Create a component to display a products list from array. the product component should display a product Id, name, purchase date, price, and image for the product and search using various pipes.

A. Objective:

Simplify the process of displaying data in a format that is appropriate for the user by using custom pipe and you can reduce the amount of code needed in your

template , improve the performance and usability of your application by using filter pipe.

.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.

2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.

3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.

4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.

5. **Life-long learning:** Ability to analyze individual needs and engage in updating in the context of technological changes *in field of engineering*.

C. Expected Skills to be developed based on competency:

Learner can develop skills in data filtering, working with arrays and objects, applying filter conditions. These skills are valuable for building robust and efficient Angular applications.

D. Expected Course Outcomes(Cos)

Apply angular directives, components and pipes in different web page development.

E. Practical Outcome(PRo)

Learner will be able to display a products list from array and search required product using filter, custom pipes

.

F. Expected Affective domain Outcome(ADos)

1. Follow Coding standards and practices.
2. Maintain tools and equipment.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory:

Pipes are used to format and transform data in a template. Pipes take in an input value, transform it, and then return the transformed value. There are two types of pipes available in Angular:

1. Built-in pipes - These are provided by Angular.

Here is a table listing the built-in pipes in Angular along with a brief description of each pipe:

Pipe	Description
DatePipe	Used to format a date according to a specified format string and locale.
DecimalPipe	Used to format a number with a fixed number of digits before and after the decimal point.
LowerCasePipe	Used to convert a string to lowercase.
PercentPipe	Used to format a number as a percentage.
SlicePipe	Used to create a new array or string containing a subset of the elements of the input array or string.
TitleCasePipe	Used to convert a string to title case, where the first letter of each word is capitalized.
UpperCasePipe	Used to convert a string to uppercase.

2. Custom pipes - These are created by developers and can be used to transform data in a specific way.

Custom pipes are useful when you need to transform data in a specific way that is not provided by the built-in pipes. In addition to custom pipes, Angular also provides filter pipes. Filter pipes are used to filter data based on a specific criteria.

Here is an example of using the built-in UpperCasePipe in Angular:

```
<!-- app.component.html -->
<p>{{ 'hello world' | uppercase }}</p>
```

In this example, the string 'hello world' is passed to the uppercase pipe, which transforms it to 'HELLO WORLD'. The transformed string is then displayed in the template using string interpolation.

Here is an example of creating a custom pipe that appends an exclamation mark to a string:

```
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { AddExclamationPipe } from './add-exclamation.pipe';

@NgModule({
  declarations: [
    AppComponent,
    AddExclamationPipe
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

```
// add-exclamation.pipe.ts
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'addExclamation'
})
export class AddExclamationPipe implements PipeTransform {
  transform(value: string): string {
    return value + '!';
  }
}
```

```
<!-- app.component.html -->
<p>{{ 'hello world' | addExclamation }}</p>
```

In this example, a custom pipe called `AddExclamationPipe` is created using the `Pipe` decorator. The pipe takes in a string value, appends an exclamation mark to it, and then returns the transformed value. The custom pipe is then used in the template by passing the string 'hello world' to it.

here's an example of how to use the **filter** pipe in Angular to filter an array of objects, including the relevant HTML, TypeScript, and pipe files, as well as the app module file:

```
<input type="text" [(ngModel)]="searchQuery">
<ul>
<li *ngFor="let item of items | filter: searchQuery">
  {{ item.name }}
</li>
</ul>
```

In this example, we have an input field where the user can type in a search query. We also have a list of items that we want to filter based on the search query. We use the **filter** pipe to filter the items based on the search query.

TypeScript:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-item-list',
  templateUrl: './item-list.component.html',
  styleUrls: ['./item-list.component.css']
})
export class ItemListComponent {
  items = [
    { id: 1, name: 'Item 1' },
    { id: 2, name: 'Item 2' },
    { id: 3, name: 'Item 3' },
    { id: 4, name: 'Item 4' },
    { id: 5, name: 'Item 5' }
  ];
  searchQuery: string = "";
}
```

In the TypeScript file, we define an array of items that we want to filter. We also define a **searchQuery** variable to hold the search query entered by the user.

Filter Pipe:

```
import { Pipe, PipeTransform } from '@angular/core';
```

```
@Pipe({
  name: 'filter'
})
export class FilterPipe implements PipeTransform {
  transform(items: any[], searchText: string): any[] {
    if (!items) {
      return [];
    }
    if (!searchText) {
      return items;
    }
    searchText = searchText.toLowerCase();
    return items.filter(item => {
      return item.name.toLowerCase().includes(searchText);
    });
  }
}
```

In the filter pipe, we define a custom **filter** pipe to filter the items based on the search query. The pipe takes in an array of items and a search query, and returns a filtered array based on the search query. The filter function checks if each item's name includes the search query and returns only those items that match.

App Module:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { FilterPipe } from './filter.pipe';
```

```
@NgModule({
  declarations: [
    FilterPipe,
    ItemListComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
```

```

    ],
    providers: [],
    bootstrap: []
  })
  export class AppModule { }

```

In the app module, we import the **FilterPipe** and declare them in the declarations array.

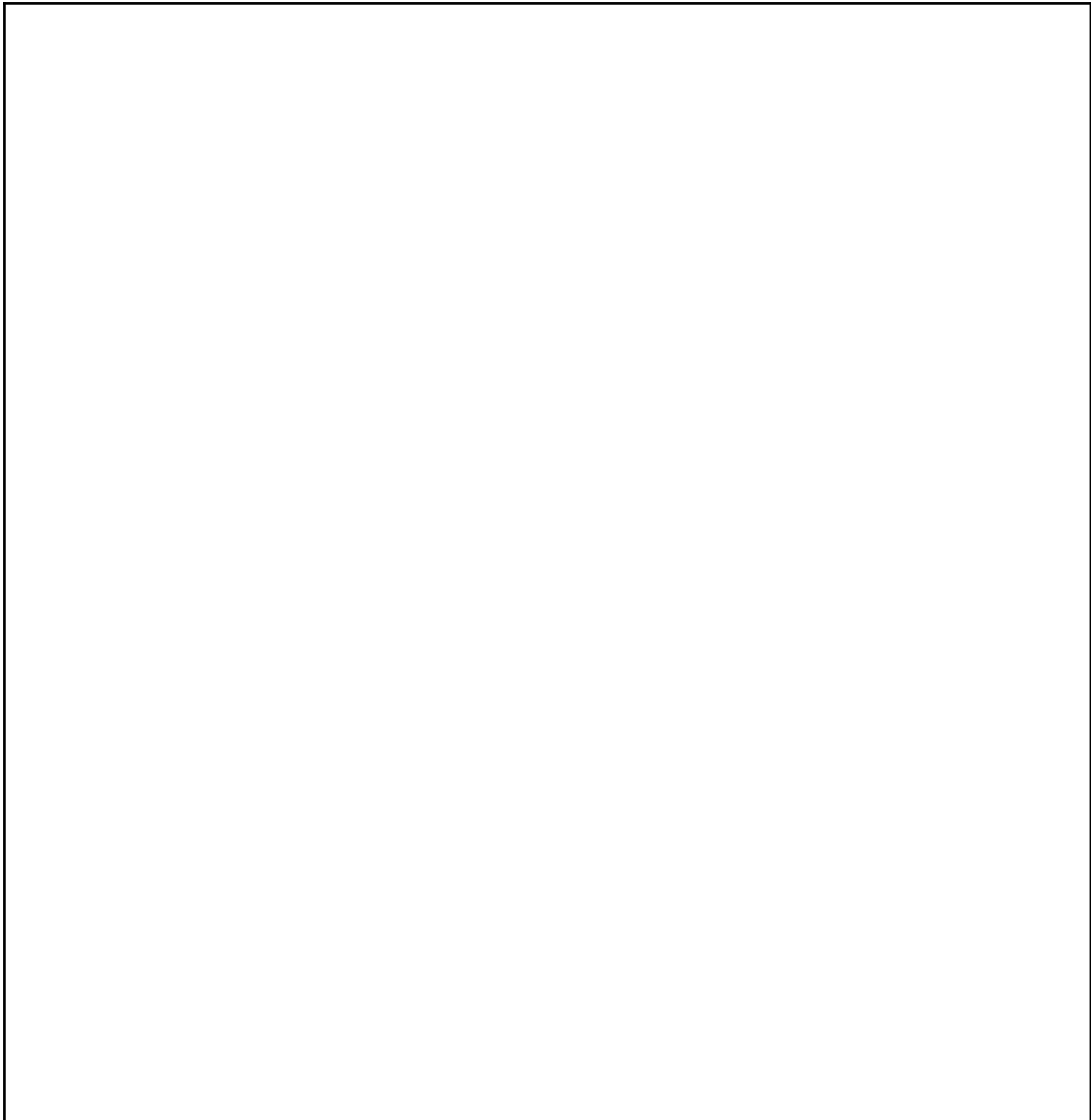
H. Resources/Equipment Required

S r. N o.	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

I. Program Source code Output

SOURCE CODE

OUTPUT

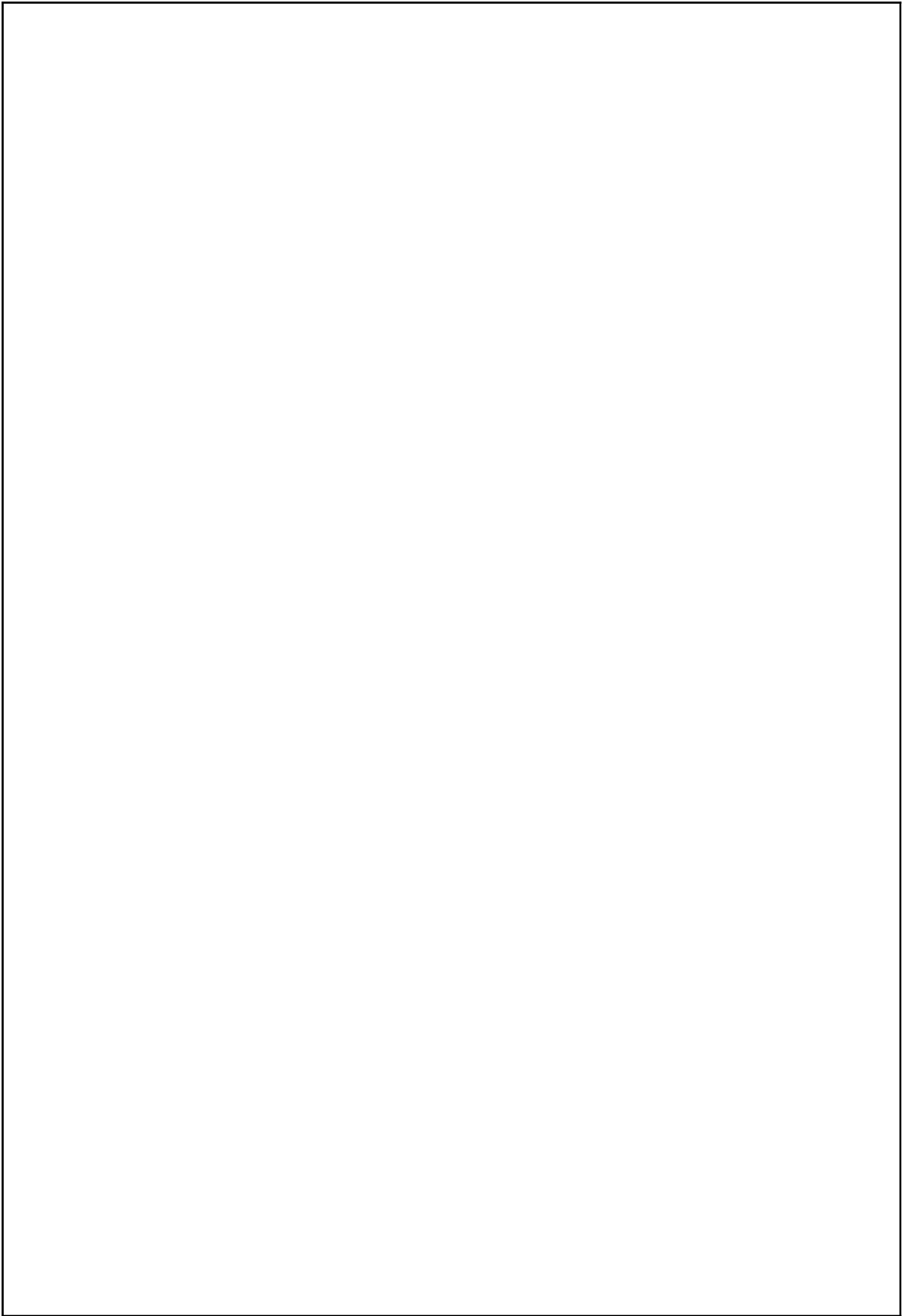


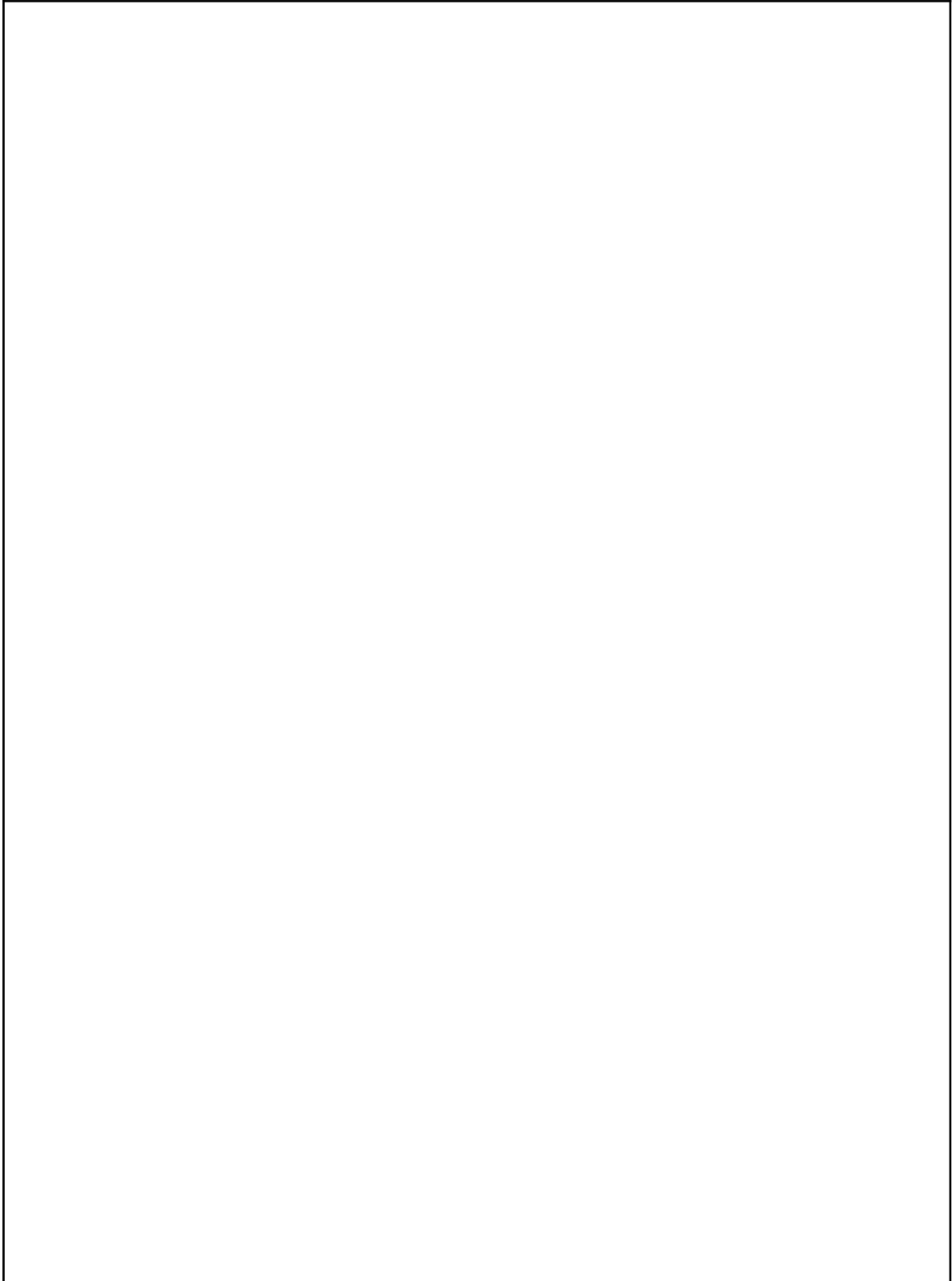
J. Practical related Exercises.

1. Design a component to apply various built in pipe on data.
2. Design a component to display student list that contains student id, name, address, semester.if student address length is more than 20 character then student address length must be display only 20 characters and append '...' [3 dot characters] at the end of address.



A large, empty rectangular box with a thin black border, occupying the central portion of the page. It is intended for the main content of the document.





K. References Links

1. <https://angular.io/guide/pipes>

2. <https://www.simplilearn.com/tutorials/angular-tutorial/angular-pipes>
3. <https://www.javatpoint.com/angular-7-pipes>
4. <https://www.geeksforgeeks.org/explain-pure-and-impure-pipe-in-angular/>
5. <https://youtu.be/uUQfi1-g92Q>

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of Input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No 8 :Design a student registration page using template driven form approach and utilize different form and controls level ng validation classes.

A. Objective:

To create template driven form that is used in design small to medium level application. It allows us to create custom form controls classes, apply validate error messages and quickly create forms with minimal coding to submit data.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.
2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.
3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.
4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.

C. Expected Skills to be developed based on competency:

Learner can develop skills related to HTML, Angular directives, TypeScript, two-way data binding, custom form controls, and form validation. These skills can be valuable for building forms and various types of Angular applications.

D. Expected Course Outcomes(Cos)

Utilize angular template driven and reactive forms in different problem solutions.

E. Practical Outcome(Pro)

Learner will be able to create template driven form approach for angular application and apply different validation classes, custom validation message, respond to form submission.

F. Expected Affective domain Outcome(ADos)

1. Follow Coding standards and practices.
2. Maintain tools and equipment.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory:

Template-driven forms is a way to create forms in Angular using templates, directives, and data binding. With template-driven forms, the form layout and behavior are defined in the HTML template, using Angular directives, rather than in the component class.

Template-driven forms are ideal for creating simple forms that don't require complex form logic or dynamic form controls. They are easy to use and require minimal coding, making them a good choice for prototyping or creating small forms.

FormsModule :

The **FormsModule** is a built-in module that provides a set of directives, services, and pipes for creating and managing forms. It is an essential module for working with forms in Angular and is included by default in most Angular applications.

The **FormsModule** provides several features that make it easy to work with forms in Angular, including:

1. Two-way data binding: The **FormsModule** provides the **ngModel** directive, which enables two-way data binding between form controls and component properties.
2. Validation: The **FormsModule** provides a set of validation directives, such as **ngRequired**, **ngMinLength**, and **ngPattern**, that can be used to validate form inputs.
3. Form submission: The **FormsModule** provides the **ngSubmit** directive, which is used to handle form submission events and trigger form validation.
4. Form control status tracking: The **FormsModule** provides the **ngModelGroup** and **ngForm** directives, which can be used to group form controls and track their status, such as validity and touched status.

To use the **FormsModule** in an Angular application, you need to import it in the root module of your application, typically in the **app.module.ts** file, like this:

```
import { FormsModule } from '@angular/forms';
```

```
@NgModule({  
  imports: [  
    // other modules  
    FormsModule  
  ],  
  // other configuration  
})  
export class AppModule { }
```

Once imported, you can use the **FormsModule** directives, services, and pipes in your components to create and manage forms in Angular.

ngModel, ngForm, and ngSubmit directives

let's look at the **ngModel**, **ngForm**, and **ngSubmit** directives purposes are given below.

1. **ngModel** Directive: The **ngModel** directive is used to bind the form control values to the component properties, enabling two-way data binding between the form and the component. In the example, we use the **ngModel** directive to bind the input values to the **name** and **email** component properties.
2. **ngForm** Directive: The **ngForm** directive is used to define the form and track its state, such as validity and touched status. In the example, we use the **ngForm** directive with the **#myForm** template reference variable to define the form and track its state.
3. **ngSubmit** Directive: The **ngSubmit** directive is used to handle the form submission event and trigger form validation. In the example, we use the **ngSubmit** directive to call the **submitForm()** method when the form is submitted.

Overall, the combination of these directives enables us to create and manage forms in Angular with minimal coding and powerful functionality.

let's take an example of the **ngModel**, **ngForm**, and **ngSubmit** directives.

HTML Template:

```
<form #myForm="ngForm" (ngSubmit)="submitForm(myForm)">
<div>
<label for="name">Name:</label>
<input type="text" name="name" [(ngModel)]="name" required>
</div>
<div>
<label for="email">Email:</label>
<input type="email" name="email" [(ngModel)]="email" required>
</div>
<button type="submit">Submit</button>
</form>
```

In this example, the form contains two form controls, a text input and an email input, both of which are required. The **ngModel** directive is used to bind the input values to the **name**

and **email** component properties, enabling two-way data binding. The **ngForm** directive is used to define the form and track its state. Finally, the **ngSubmit** directive is used to handle the form submission event and trigger form validation.

TypeScript File:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-my-form',
  templateUrl: './my-form.component.html',
  styleUrls: ['./my-form.component.css']
})
export class MyFormComponent {
  name: string;
  email: string;

  submitForm(form: any): void {
    console.log(form);
  }
}
```

In this example, we have created a component called **MyFormComponent** that defines the component's behavior and interacts with the template. The component has two properties, **name** and **email**, which are used to store the input values. The **submitForm()** method is called when the form is submitted and receives the **myForm** object as a parameter, which is an instance of the **NgForm** class. The method logs the form object to the console.

form validation state

In template-driven forms, form validation is an essential feature that ensures that user inputs are correct before submitting the form. The validation process involves tracking the state of each form control and providing feedback to the user about the validity of their inputs.

There are several states that a form control can be in, including:

1. **untouched**: The form control has not been touched by the user yet.
2. **touched**: The form control has been touched by the user.
3. **pristine**: The form control has not been modified by the user yet.

4. **dirty**: The form control has been modified by the user.
5. **valid**: The form control value is valid according to the validation rules.
6. **invalid**: The form control value is invalid according to the validation rules.

To display the validation state of a form control, you can use CSS classes that reflect the control's state. Angular provides several CSS classes for this purpose, including **ng-untouched**, **ng-touched**, **ng-pristine**, **ng-dirty**, **ng-valid**, and **ng-invalid**.

HTML Template

The HTML template is where we define the form layout and validation directives that will be used to validate the form input. In the following example, we have a simple form that requires a name and an email, with validation for both fields.

```
<form #myForm="ngForm" (ngSubmit)="onSubmit(myForm)">
<div>
<label for="name">Name:</label>
<input type="text" name="name" [(ngModel)]="model.name" required minlength="3"
maxlength="50" #nameInput="ngModel">
<div *ngIf="nameInput.invalid && (nameInput.dirty || nameInput.touched)">
<div *ngIf="nameInput.errors.required">Name is required.</div>
<div *ngIf="nameInput.errors.minlength">Name must be at least 3 characters long.</div>
<div *ngIf="nameInput.errors.maxlength">Name cannot be longer than 50
characters.</div>
</div>
</div>
<div>
<label for="email">Email:</label>
<input type="email" name="email" [(ngModel)]="model.email" required email
#emailInput="ngModel">
<div *ngIf="emailInput.invalid && (emailInput.dirty || emailInput.touched)">
<div *ngIf="emailInput.errors.required">Email is required.</div>
<div *ngIf="emailInput.errors.email">Email must be a valid email address.</div>
</div>
</div>
<button type="submit" [disabled]="myForm.invalid">Submit</button>
</form>
```

In this example, we define the form using the **ngForm** directive, which allows us to track the form state and handle the form submission. We then define two input fields, one for the name and one for the email, using the **ngModel** directive to bind the input values to the

model object. We also add several validation directives to each input field, including **required**, **minlength**, **maxlength**, and **email**.

To display the validation messages, we use the ***ngIf** directive to check if the input field is invalid and has been touched or is dirty. If this condition is met, we display the appropriate validation message.

Finally, we disable the submit button if the form is invalid, using the **[disabled]** directive.

CSS Styling

The CSS styling for the form validation is entirely optional, but it can help to provide visual feedback to the user when the form is in an invalid state. In the following example, we add some simple styling to display the validation messages in red text.

```
input.ng-invalid {  
  border-color: red;  
}
```

```
.error-message {  
  color: red;  
}
```

This CSS styling changes the border color of an input field when it is in an invalid state, making it more visually distinct. It also applies a red color to the validation messages, making them more noticeable.

TypeScript Code

Finally, we need to add some TypeScript code to handle the form submission and set the **model** object. In the following example, we define a **model** object in the component class and add an **onSubmit()** method to handle the form submission.

```
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-form',  
  templateUrl: './form.component.html',  
  styleUrls: ['./form.component.css']  
})  
export class FormComponent {  
  model = {  
    name: "",  
    email: ""  
  }  
}
```

```

    };
    onSubmit (form: NgForm) {
    if (form.valid) {
        // form is valid, submit data
    }
    }
}

```

In this example, we define a model object with **name** and **email** properties in the component class. We use two-way data binding with the **[(ngModel)]** directive to bind the input values to the properties of the model object.

We also define an **onSubmit()** method in the component class that is called when the form is submitted. This method can perform other actions such as submitting the form data to a server.

Note that we use the **NgForm** class to access the form data in the **onSubmit()** method. We pass the **myForm** template reference variable to the method to access the form data.

Form Submission Data

In template-driven forms in Angular, you can submit form data and display it in the template using template reference variables and data binding.

Here is an example of how to submit form data and display it in the template:

```

<form #myForm="ngForm" (ngSubmit)="onSubmit(myForm)">
<div class="form-group">
<label for="name">Name:</label>
<input type="text" name="name" [(ngModel)]="model.name" required>
</div>
<div class="form-group">
<label for="email">Email:</label>
<input type="email" name="email" [(ngModel)]="model.email" required email>
</div>
<button type="submit">Submit</button>
</form>

<div *ngIf="submitted">
<h3>Form Data</h3>
<p>Name: {{ model.name }}</p>

```

```
<p>Email: {{ model.email }}</p>
</div>
```

Typescript file

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-my-form',
  templateUrl: './my-form.component.html',
  styleUrls: ['./my-form.component.css']
})
export class MyFormComponent {
  model = {
    name: "",
    email: ""
  };
  submitted = false;

  onSubmit(form: NgForm) {
    console.log(this.model);
    // submit form data to server or perform other actions
    this.submitted = true;
  }
}
```

In this example, we define a **submitted** property in the component class that is initially set to **false**. We also add a ***ngIf** directive to the **div** element that displays the form data to only display it when **submitted** is **true**.

In the **onSubmit()** method, we set **submitted** to **true** after performing any necessary actions with the form data.

When the form is submitted, the **submitted** property is set to **true** and the form data is displayed in the template. The **{{ }}** syntax is used for data binding to display the values of the **name** and **email** properties of the **model** object.

Note that this is just one example of how to submit and display form data in template-driven forms in Angular. There are other ways to achieve this functionality depending on your specific requirements.

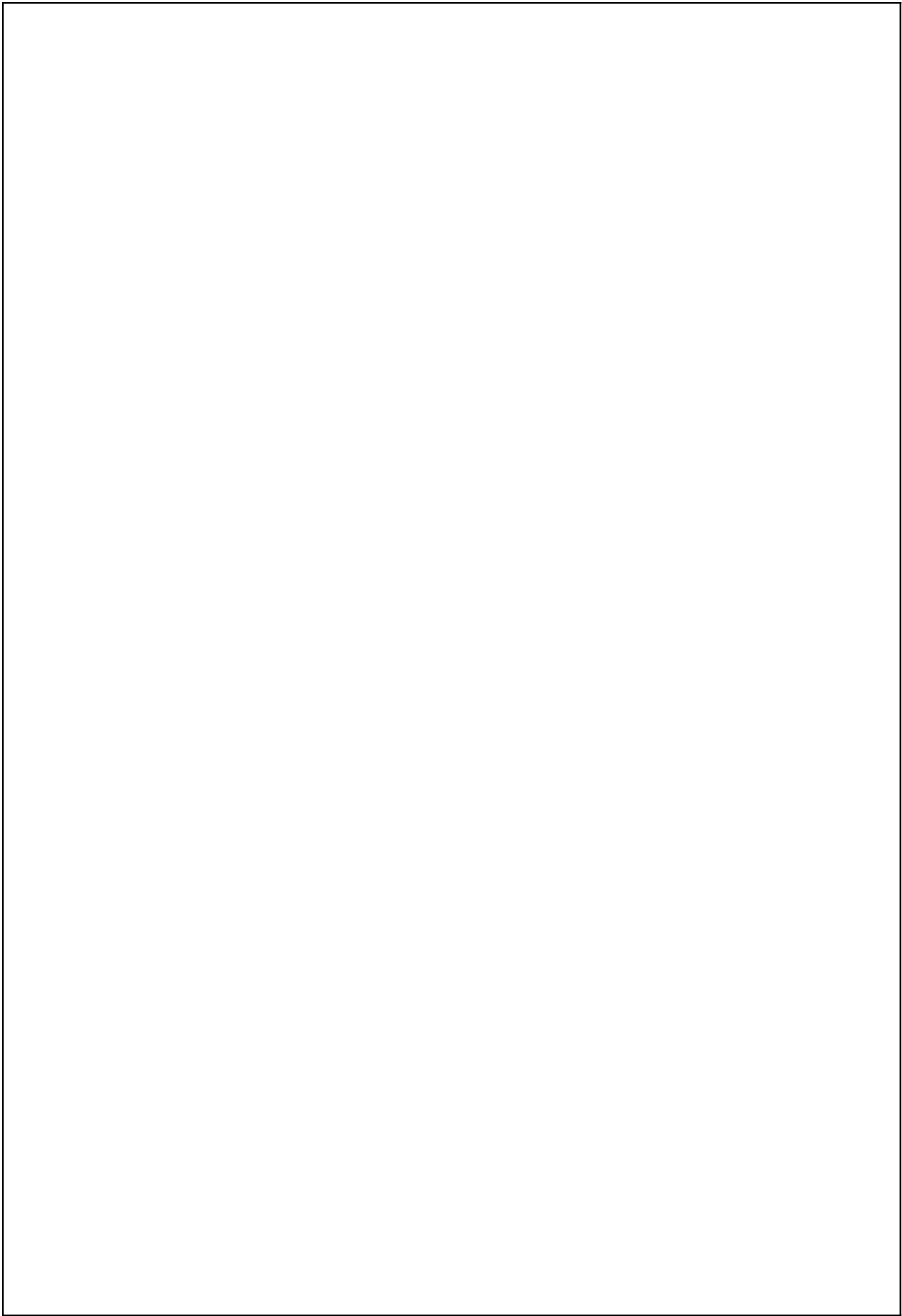
H. Resources/Equipment Required

S r. N o.	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

I. Program Source code Output

SOURCE CODE

A large, empty rectangular box with a thin black border, occupying the central portion of the page. It is intended for a drawing or a detailed written response.



OUTPUT



J. Practical related Exercises.

1. Design a component to display student result data as given below by using directives like ngForm, ngModel, and custom validation.

Student Result data	Student Result data
Enrollment No : <input type="text" value="123"/>	Enrollment No : <input type="text" value="123"/>
Web Developmemnt : <input type="text" value="10"/>	Web Developmemnt : <input type="text" value="78"/>
Computer Networ : <input type="text" value="20"/>	Computer Networ : <input type="text" value="82"/>
Modern Practical Tool (Angular) : <input type="text" value="20"/>	Modern Practical Tool (Angular) : <input type="text" value="50"/>
<input type="button" value="Get Reusult"/>	<input type="button" value="Get Result"/>
Sorry!! You have not cleared Exam...!!	Enrollment No : 123 WDT : 78 CN : 82 Angular : 50 Grade:BB Congratulations!!You have cleared Exam..

2. Design component to create feedback form of any application given by respected teacher.

A large, empty rectangular box with a thin black border, occupying the central portion of the page. It is intended for a drawing or a detailed written response.

K. References Links

1. <https://angular.io/guide/forms>
2. <https://blog.angular-university.io/introduction-to-angular-2-forms-template-driven-vs-model-driven/>
3. <https://www.w3resource.com/angular/form-validation.php>
4. <https://www.javatpoint.com/angular-template-driven-forms>
5. https://www.tutorialspoint.com/angular8/angular8_forms.html
6. <https://youtu.be/eLbttIXsrrc>

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of Input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No 9 :Design component to enter faculty details like Code, Name, Email, Type, Faculty Status (Active, Inactive), Subjects Teaching (with option to add multiple subjects dynamically) using reactive form with various types of validation of form and controls.

A. Objective:

The practical will help to manage complex forms, implement custom validation rules, and dynamically modify the form based on user input.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.

2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.

3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.

4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.

C. Expected Skills to be developed based on competency:

Learner can develop Strong understanding of Angular components and modules, Familiarity with reactive programming concepts and reactive extensions, Ability to manage complex forms, Knowledge of form state management and reactive programming concepts, Understanding of how to create custom form controls and validators, Ability to use reactive forms to implement dynamic and responsive user interfaces

D. Expected Course Outcomes(Cos)

Utilize angular template driven and reactive forms in different problem solutions.

E. Practical Outcome(PRo)

Learner will be able to create faculty detail page using reactive form with various types of validations.

F. Expected Affective domain Outcome(ADos)

1. Follow Coding standards and practices.
2. Maintain tools and equipment.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory:

ReactiveFormsModule is a built-in Angular module that provides support for reactive forms in an Angular application. Here most of the code manage in ts file rather than html. This module is imported from the **@angular/forms** package and must be included in the **imports** array of your Angular module.

Import the ReactiveFormsModule into your app.Module.ts file

```
import { NgModule } from '@angular/core';
import { ReactiveFormsModule } from '@angular/forms';
```

```
@NgModule({
  imports: [
    // other imports...
    ReactiveFormsModule
  ],
  // other module properties...
})
export class AppModule { }
```

In Angular, a **FormGroup** is a collection of **FormControl** objects that represents a form. A **FormControl** represents an individual form control, such as an input field or select box.

Here's an example of how to use **FormGroup** and **FormControl** in Angular:

```
import { Component } from '@angular/core';
import { FormGroup, FormControl } from '@angular/forms';
```

```
@Component({
  selector: 'app-example',
  template: `
    <form [formGroup]="exampleForm">
    <label>
      Name:
    <input formControlName="name">
    </label>
```

```

<label>
  Email:
<input formControlName="email">
</label>
</form>
,
}))
export class ExampleComponent {
  exampleForm = new FormGroup({
    name: new FormControl(),
    email: new FormControl()
  });
}

```

In the above example, we've created a **FormGroup** object called **exampleForm**. We've then added two **FormControl** objects to this group, **name** and **email**. We've bound these controls to input fields in our template using the **formControlName** directive.

In Angular Reactive Forms, validators are functions that are used to validate form input fields. Validators can be built-in functions provided by Angular, or custom validators created by the developer.

Here's an example of how to use built-in validators in Angular Reactive Forms:

HTML:

```

<form [formGroup]="form" (ngSubmit)="onSubmit()">
<div>
<label for="name">Name:</label>
<input type="text" id="name" formControlName="name">
<div *ngIf="form.controls['name'].invalid && (form.controls['name'].dirty ||
form.controls['name'].touched)">
<div *ngIf="form.controls['name'].errors.required">Name is required.</div>
<div *ngIf="form.controls['name'].errors.minlength">Name must be at least 3 characters
long.</div>
</div>
</div>
<div>
<label for="email">Email:</label>
<input type="email" id="email" formControlName="email">
<div *ngIf="form.controls['email'].invalid && (form.controls['email'].dirty ||
form.controls['email'].touched)">
<div *ngIf="form.controls['email'].errors.required">Email is required.</div>

```

```
<div *ngIf="form.controls['email'].errors.email">Email must be a valid email
address.</div>
</div>
</div>
<button type="submit">Submit</button>
</form>
```

TS:

```
import { Component } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/forms';

@Component({
  selector: 'app-example',
  templateUrl: './example.component.html',
  styleUrls: ['./example.component.css']
})
export class ExampleComponent {
  form = new FormGroup({
    name: new FormControl("", [Validators.required, Validators.minLength(3)]),
    email: new FormControl("", [Validators.required, Validators.email])
  });

  onSubmit() {
    console.log(this.form.value);
  }
}
```

CSS:

```
input.ng-invalid.ng-touched,
input.ng-invalid.ng-dirty {
  border-color: red;
}
```

In the above example, we've created a form with two form controls: **name** and **email**. We've added built-in validators to the **FormControl** objects. The **name** control is required and must have a minimum length of 3 characters. The **email** control is also required and must be a valid email address.

In the HTML template, we're using Angular's built-in ***ngIf** directive to display error messages when the form inputs are invalid. We're also applying a red border to invalid inputs using a custom CSS class.

In the TypeScript code, we're creating a **FormGroup** object to represent the form and binding it to the form element using the **[formGroup]** directive. We're also creating an **onSubmit()** method to handle form submissions.

Overall, validators are a powerful feature of Angular Reactive Forms that enable developers to easily validate and handle user input.

Custom validators are a way to add additional validation logic to Angular reactive forms beyond the built-in validators. In Angular, a custom validator is simply a function that takes a form control as input and returns either **null** if the validation passes, or an object containing an error message if the validation fails.

Here's an example of how to create a custom validator for a reactive form in Angular:

In the HTML file, create a form with an input field for the user's age:

```
<form [formGroup]="myForm">
<label for="age">Age:</label>
<input type="number" id="age" formControlName="age" />
<div *ngIf="myForm.get('age').invalid && myForm.get('age').touched">
<div *ngIf="myForm.get('age').errors.required">
  Age is required.
</div>
<div *ngIf="myForm.get('age').errors.invalidAge">
  Age must be between 18 and 60.
</div>
</div>
</form>
```

In the component's TS file, import the **FormControl** and **FormGroup** classes from **@angular/forms**, and create a form with a control for the age input:

```
import { Component } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-example',
  templateUrl: './example.component.html',
  styleUrls: ['./example.component.css']
})
```

```

    })
    export class ExampleComponent {
      myForm = new FormGroup({
        age: new FormControl("", [Validators.required, this.ageValidator])
      });

      ageValidator(control: FormControl) {
        const age = control.value;
        if (age < 18 || age > 60) {
          return { invalidAge: true };
        }
        return null;
      }
    }
  }

```

In the CSS file, you can add styles to the form and error messages as desired:

In this example, we created a custom validator called **ageValidator** that checks whether the user's age is between 18 and 60. If the age is outside that range, the function returns an object with a key of **invalidAge**. The **ageValidator** function is then passed as a second argument to the **FormControl** constructor, along with the built-in **Validators.required** validator.

The HTML file uses the ***ngIf** directive to show error messages if the age input is invalid and has been touched by the user. The **myForm.get('age').errors** object is used to access the error messages returned by the validators.

The CSS file applies styles to the input field if it is invalid, and to the error messages if they are displayed.

FormArray in reactive forms is a way to manage an array of form controls in Angular. It is used when you have a dynamic number of form fields, such as a list of items that can be added or removed from a form. You can also add validation to the FormArray to ensure that the input values are correct.

Here's an example of creating a FormArray with validation in a reactive form in Angular:

In the HTML file, create a form with a FormArray to manage a list of items:

```

<form [formGroup]="myForm">
  <div formArrayName="items">

```



```

<div *ngFor="let item of items.controls; let i = index">
  <div [formGroupName]="i">
    <label for="name{{i}}">Name:</label>
    <input type="text" id="name{{i}}" formControlName="name" />
    <label for="price{{i}}">Price:</label>
    <input type="number" id="price{{i}}" formControlName="price" />
    <div *ngIf="item.get('price').invalid && item.get('price').touched">
      <div *ngIf="item.get('price').errors.required">
        Price is required.
      </div>
      <div *ngIf="item.get('price').errors.min">
        Price must be greater than zero.
      </div>
    </div>
    <button (click)="removeItem(i)">Remove</button>
  </div>
</div>
<button (click)="addItem()">Add Item</button>
</form>

```

In the component's TS file, import the FormControl, FormGroup, and FormArray classes from @angular/forms, and create a FormArray with validation for the list of items:

```

import { Component } from '@angular/core';
import { FormControl, FormGroup, Validators, FormArray } from '@angular/forms';

```

```

@Component({
  selector: 'app-example',
  templateUrl: './example.component.html',
  styleUrls: ['./example.component.css']
})
export class ExampleComponent {
  myForm = new FormGroup({
    items: new FormArray([
      this.createItem()
    ])
  });
}

```

```

createItem(): FormGroup {
  return new FormGroup({

```

```
name: new FormControl("", Validators.required),
price: new FormControl(0, [Validators.required, Validators.min(0.01)])
  });
}

get items(): FormArray {
return this.myForm.get('items') as FormArray;
}

addItem() {
this.items.push(this.createItem());
}

removeItem(index: number) {
this.items.removeAt(index);
}
}
```

In the CSS file, you can add styles to the form and error messages as desired:

```
input.ng-invalid {
border-color: red;
}

div.error {
color: red;
font-size: 0.8em;
}
```

In this example, we created a FormArray called **items** that manages an array of items, each represented by a FormGroup with **name** and **price** form controls. The **createItem** function returns a new FormGroup with validation rules for the **name** and **price** form controls.

The **get items()** method is used to access the **items** FormArray in the template.

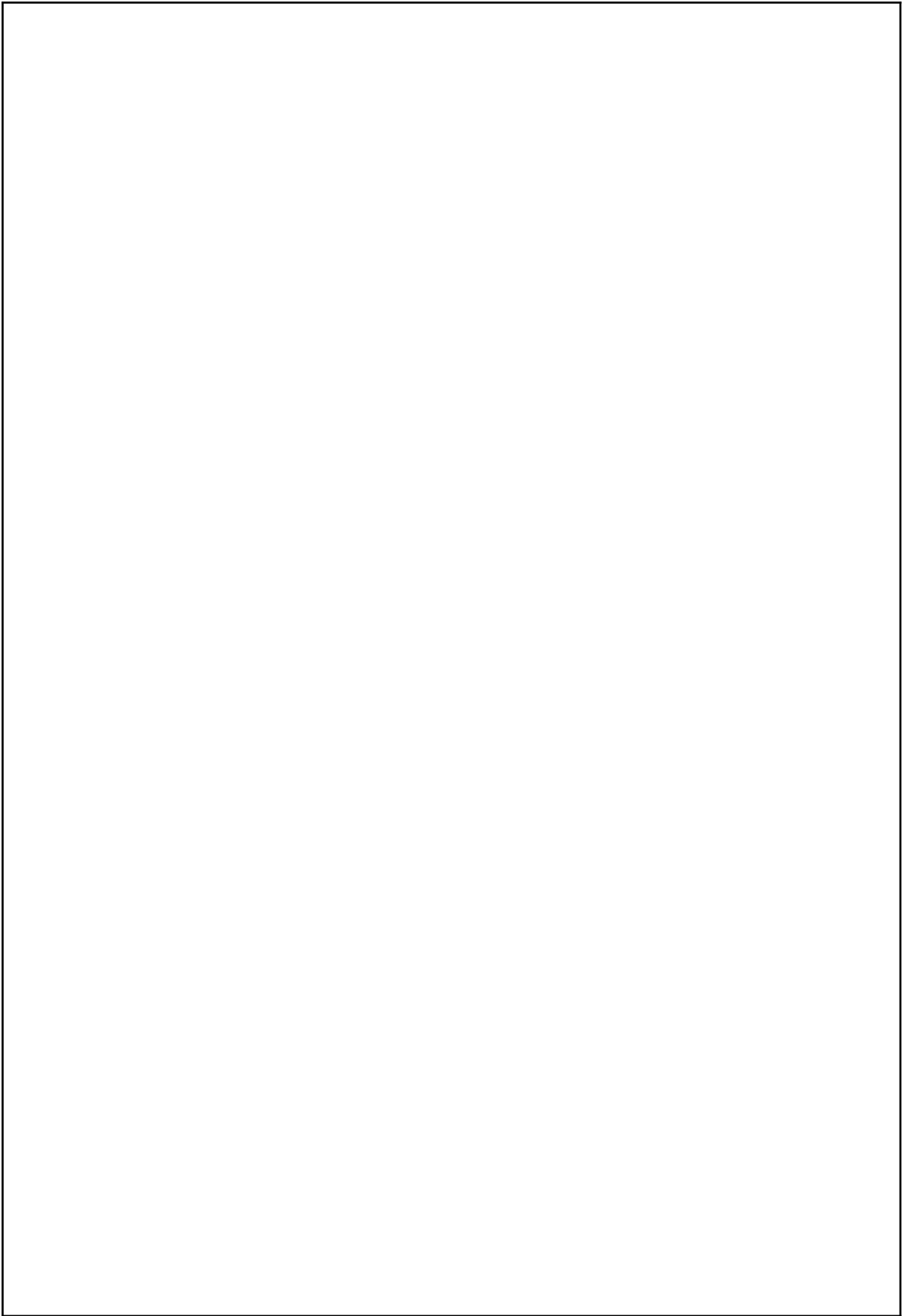
The **addItem** method is used to add a new item to the **items** FormArray, and the **removeItem** method is used to remove an item from the **items** FormArray.

The HTML file uses the ***ngFor** directive to loop through the **items** FormArray and display each item's form controls

H. Resources/Equipment Required

S r. N o.	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

I. Program Source code Output**SOURCE CODE**



A large, empty rectangular box with a thin black border, occupying the central portion of the page. It is intended for a drawing or a detailed written response.

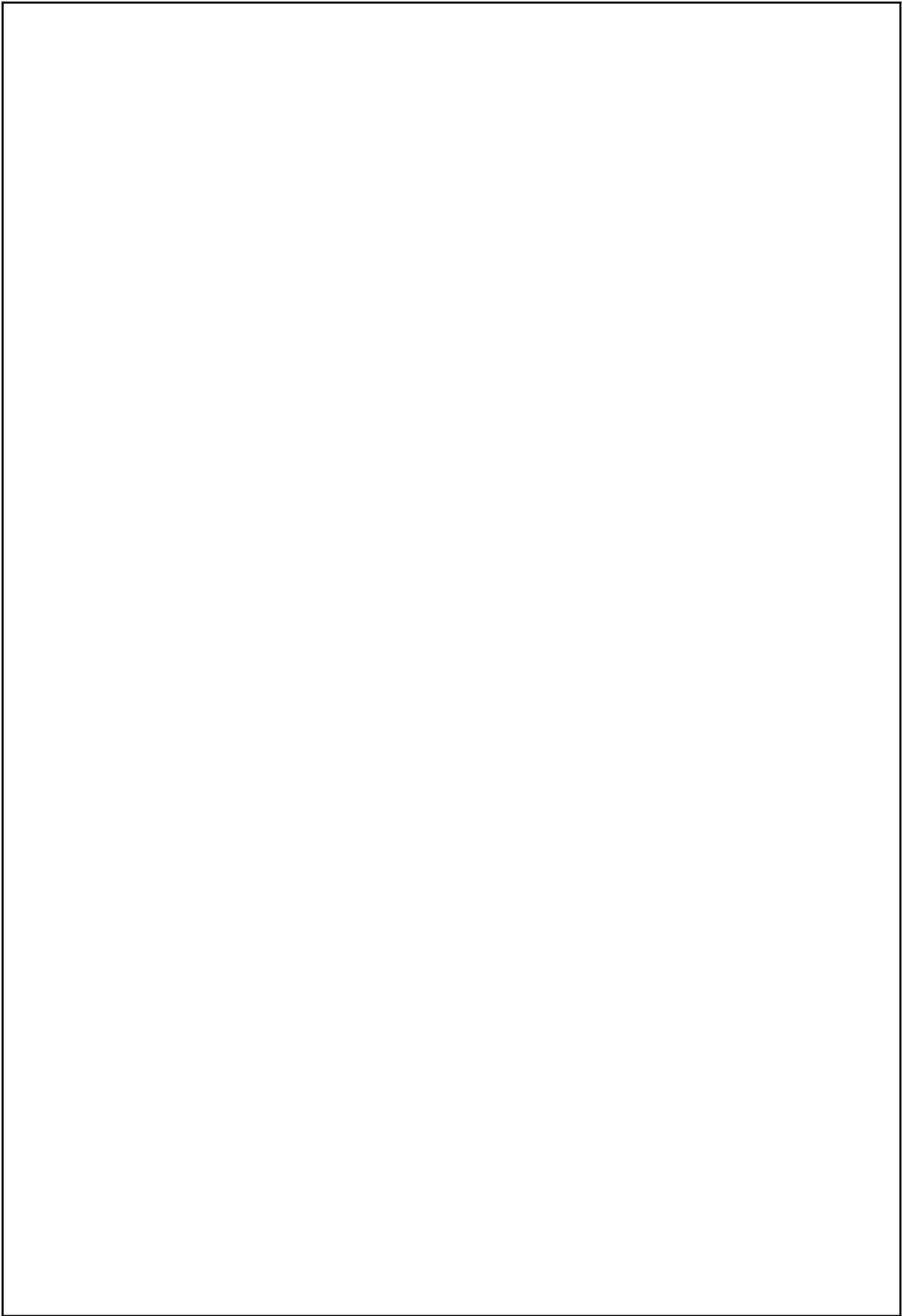
OUTPUT

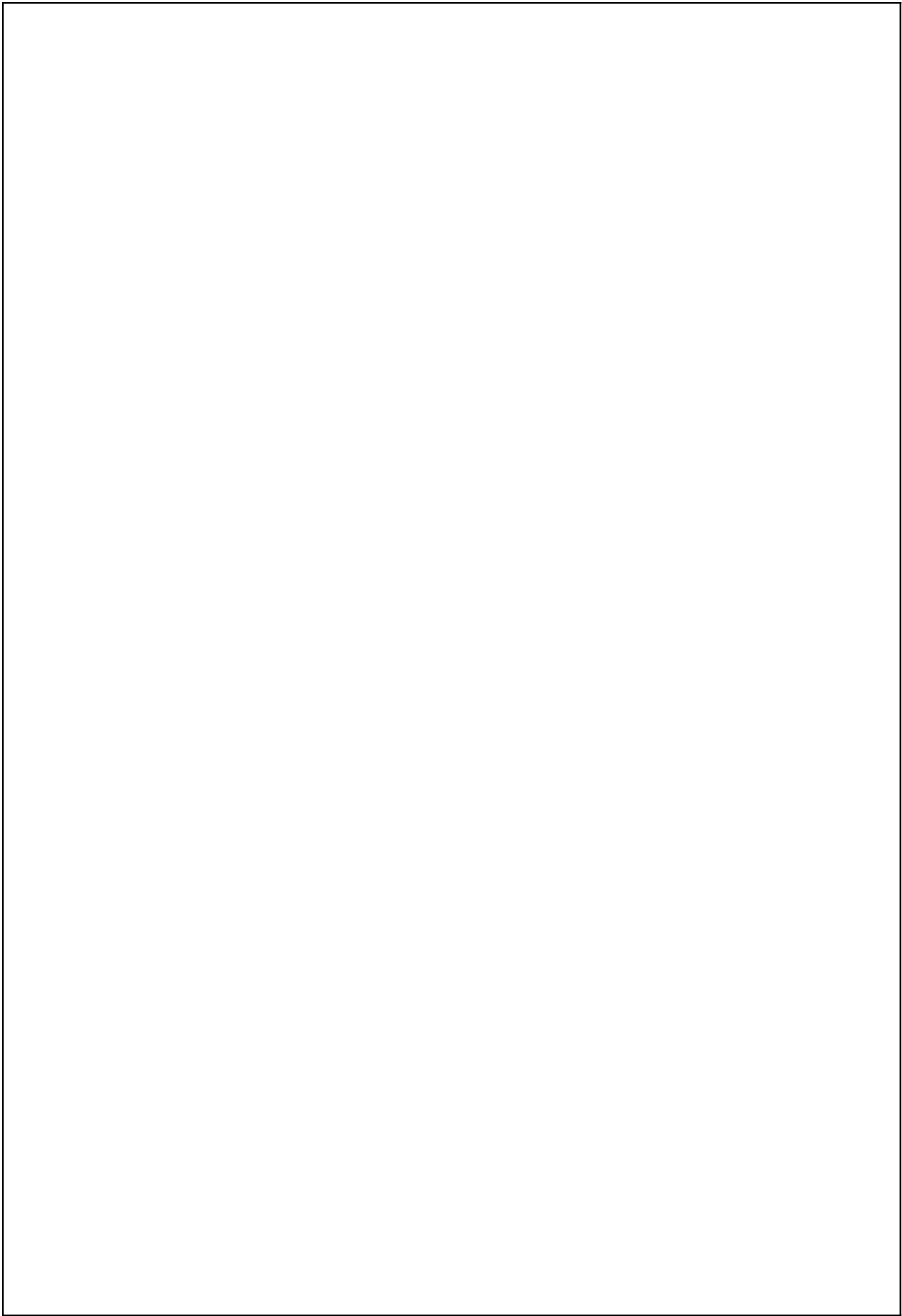
--

J. Practical related Exercises.

1. Write down difference between template driven form and reactive form.
2. Design form with multiple fields such as name, email, phone number, and password, that collects user information and validates it..

--





--

K. References Links

1. <https://angular.io/guide/reactive-forms>
2. <https://www.digitalocean.com/community/tutorials/angular-reactive-forms-introduction>
3. <https://www.geeksforgeeks.org/how-to-check-whether-a-form-or-a-control-is-touched-or-not-in-angular-10/>
4. <https://www.javatpoint.com/dynamically-add-and-remove-fields-in-angular>
5. <https://youtu.be/0sGMNeK8o78>
6. <https://youtu.be/wktyHrIS-dk>

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of Input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No 10: Design a page to implement Add to Cart functionality using decorators, custom properties, custom events of component communication.

A. Objective:

- Modify the behaviour of a class, property, method, or parameter
- Pass data from a parent component to a child component
- Modify the behavior of a child component based on data passed from the parent component and Pass data from the child component to the parent component.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.

2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.

3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.

4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.

C. Expected Skills to be developed based on competency:

Learner can develop a deeper understanding of how decorators work, how to encapsulate and share data across different parts of your application, help you to create more powerful and dynamic applications.

D. Expected Course Outcomes(Cos)

Utilize angular template driven and reactive forms in different problem solutions.

E. Practical Outcome(Pro)

Learner will be able to design module using decorators, custom properties, custom events of component communication to make real time user interactive application.

F. Expected Affective domain Outcome(ADos)

1. Follow Coding standards and practices.
2. Maintain tools and equipment.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory:

In Angular, a decorator is a function that is used to modify a class, method, property, or parameter. Decorators provide a way to add metadata to your code, which can be used by the Angular framework to configure your application.

Angular provides several built-in decorators that you can use to modify the behavior of your components, services, directives, and other classes.

By using decorators, you can easily add features such as dependency injection, event handling, and custom directives to your components and services.

@Input decorator

In Angular, the @Input decorator is used to pass data from a parent component to its child component. Here's an example of how to use @Input decorator and custom properties:

Create a child component called child-component using the Angular CLI command: `ng generate component child-component`.

In the child component's TypeScript file (child-component.ts), import the Input decorator from @angular/core.

```
import { Component, Input } from '@angular/core';
```

```
@Component({  
  selector: 'app-child-component',  
  templateUrl: './child-component.html',  
  styleUrls: ['./child-component.css']  
})  
export class ChildComponent {  
  @Input() customProperty: string;  
}
```

In this example, we create a custom property called customProperty using the @Input decorator. This property can be set by the parent component.

In the child component's HTML file (child-component.html), display the value of the custom property using interpolation syntax.

```
<p>Custom property value: {{ customProperty }}</p>
```

In the parent component's TypeScript file (parent-component.ts), set the value of the custom property.

```
import { Component } from '@angular/core';
```

```
@Component({  
  selector: 'app-parent-component',  
  templateUrl: './parent-component.html',  
  styleUrls: ['./parent-component.css']  
})  
export class ParentComponent {  
  customPropertyValue = 'Hello, world!';  
}
```

In this example, we define the customPropertyValue variable and set it to a string.

In the parent component's HTML file (parent-component.html), add the child component and bind to the custom property using the [customProperty] syntax.

```
<app-child-component  
[customProperty]="customPropertyValue"></app-child-component>
```

In this example, we bind to the customProperty using the [customProperty] syntax and pass the customPropertyValue variable as the value.

That's how you can use @Input decorator and custom properties in Angular to pass data from a parent component to its child component.

@Output decorator

In Angular, the @Output decorator is used to create custom events that can be emitted from a child component to its parent component. Here's an example of how to use @Output decorator and custom events:

Create a child component called child-component using the Angular CLI command: `ng generate component child-component`.

In the child component's TypeScript file (`child-component.ts`), import the Output decorator from `@angular/core` and create an instance of the `EventEmitter` class.

```
import { Component, Output, EventEmitter } from '@angular/core';
```

```
@Component({
  selector: 'app-child-component',
  templateUrl: './child-component.html',
  styleUrls: ['./child-component.css']
})
export class ChildComponent {
  @Output() customEvent = new EventEmitter<string>();

  handleClick() {
    this.customEvent.emit('Custom event was triggered!');
  }
}
```

In this example, we create a custom event called `customEvent` using the `@Output` decorator. We also create an instance of the `EventEmitter` class, which is used to emit the custom event when a button is clicked. The `handleClick()` method is called when the button is clicked, which triggers the custom event using `this.customEvent.emit()`.

In the child component's HTML file (`child-component.html`), add a button that calls the `handleClick()` method when clicked.

```
<button (click)="handleClick()">Click me!</button>
```

In the parent component's TypeScript file (`parent-component.ts`), import the `ViewChild` decorator from `@angular/core` and the child component.

```
import { Component, ViewChild } from '@angular/core';
import { ChildComponent } from './child-component/child-component.component';
```

```
@Component({
  selector: 'app-parent-component',
  templateUrl: './parent-component.html',
  styleUrls: ['./parent-component.css']
})
```



```
export class ParentComponent {  
  @ViewChild(ChildComponent) childComponent: ChildComponent;  
  
  handleCustomEvent(event: string) {  
    console.log(event);  
  }  
}
```

In this example, we use the ViewChild decorator to get a reference to the child component. We also define the handleCustomEvent() method to handle the custom event.

In the parent component's HTML file (parent-component.html), add the child component and bind to the custom event using the (customEvent) syntax.

```
<app-child-component  
(customEvent)="handleCustomEvent($event)"></app-child-component>
```

In this example, we bind to the customEvent using the (customEvent) syntax and call the handleCustomEvent() method when the custom event is triggered.

That's how you can use @Output decorator and custom events in Angular to emit events from a child component to its parent component.

@input and @output decorator

In Angular, you can use both @Input and @Output decorators to create a two-way data binding between a parent component and its child component. Here's an example of how to use @Input and @Output decorators together:

Create a child component called child-component using the Angular CLI command: ng generate component child-component.

In the child component's TypeScript file (child-component.ts), import the Input and Output decorators from @angular/core.

```
import { Component, Input, Output, EventEmitter } from '@angular/core';
```

```
@Component({  
  selector: 'app-child-component',  
  templateUrl: './child-component.html',  
  styleUrls: ['./child-component.css']  
})
```

```

    })
    export class ChildComponent {
      @Input() customProperty: string;
      @Output() customEvent = new EventEmitter<string>();

      handleClick() {
        this.customEvent.emit('Custom event was triggered!');
      }
    }
  }

```

In this example, we create a custom property called `customProperty` using the `@Input` decorator. We also create a custom event called `customEvent` using the `@Output` decorator and an instance of the `EventEmitter` class. The `handleClick()` method is called when a button is clicked, which triggers the custom event using `this.customEvent.emit()`.

In the child component's HTML file (`child-component.html`), display the value of the custom property using interpolation syntax and add a button that calls the `handleClick()` method when clicked.

```

<p>Custom property value: {{ customProperty }}</p>
<button (click)="handleClick()">Click me!</button>

```

In the parent component's TypeScript file (`parent-component.ts`), define the custom property value and the method that handles the custom event.

```

import { Component } from '@angular/core';

@Component({
  selector: 'app-parent-component',
  templateUrl: './parent-component.html',
  styleUrls: ['./parent-component.css']
})
export class ParentComponent {
  customPropertyValue = 'Hello, world!';

  handleCustomEvent(event: string) {
    console.log(event);
  }
}

```

In this example, we define the `customPropertyValue` variable and set it to a string. We also define the `handleCustomEvent()` method to handle the custom event.

In the parent component's HTML file (`parent-component.html`), add the child component and bind to the custom property using the `[customProperty]` syntax. Also, bind to the custom event using the `(customEvent)` syntax.

```
<app-child-component [customProperty]="customPropertyValue"
(customEvent)="handleCustomEvent($event)"></app-child-component>
```

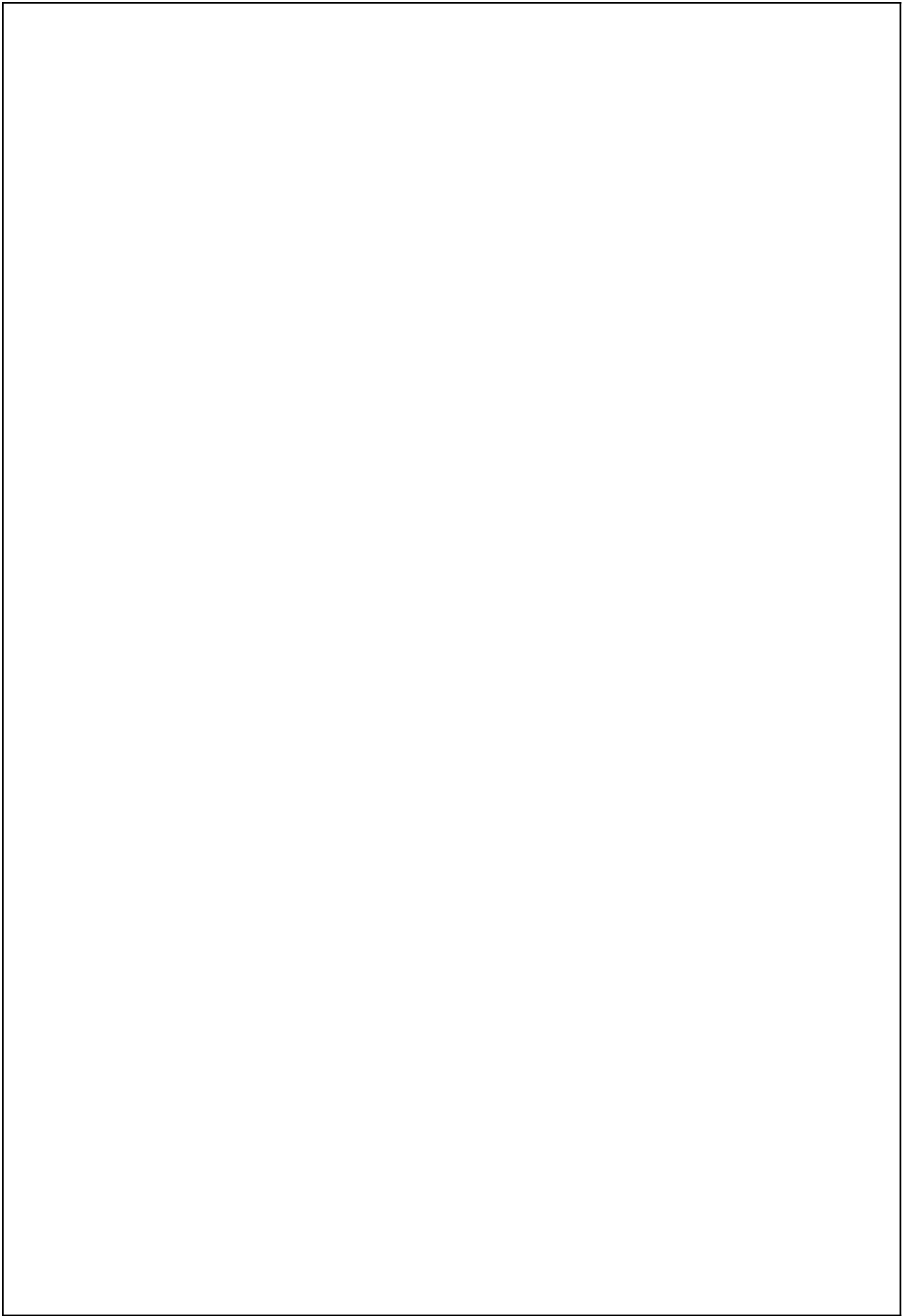
In this example, we bind to the `customProperty` using the `[customProperty]` syntax and pass the `customPropertyValue` variable as the value. We also bind to the `customEvent` using the `(customEvent)` syntax and call the `handleCustomEvent()` method when the custom event is triggered.

H. Resources/Equipment Required

S r. N o.	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

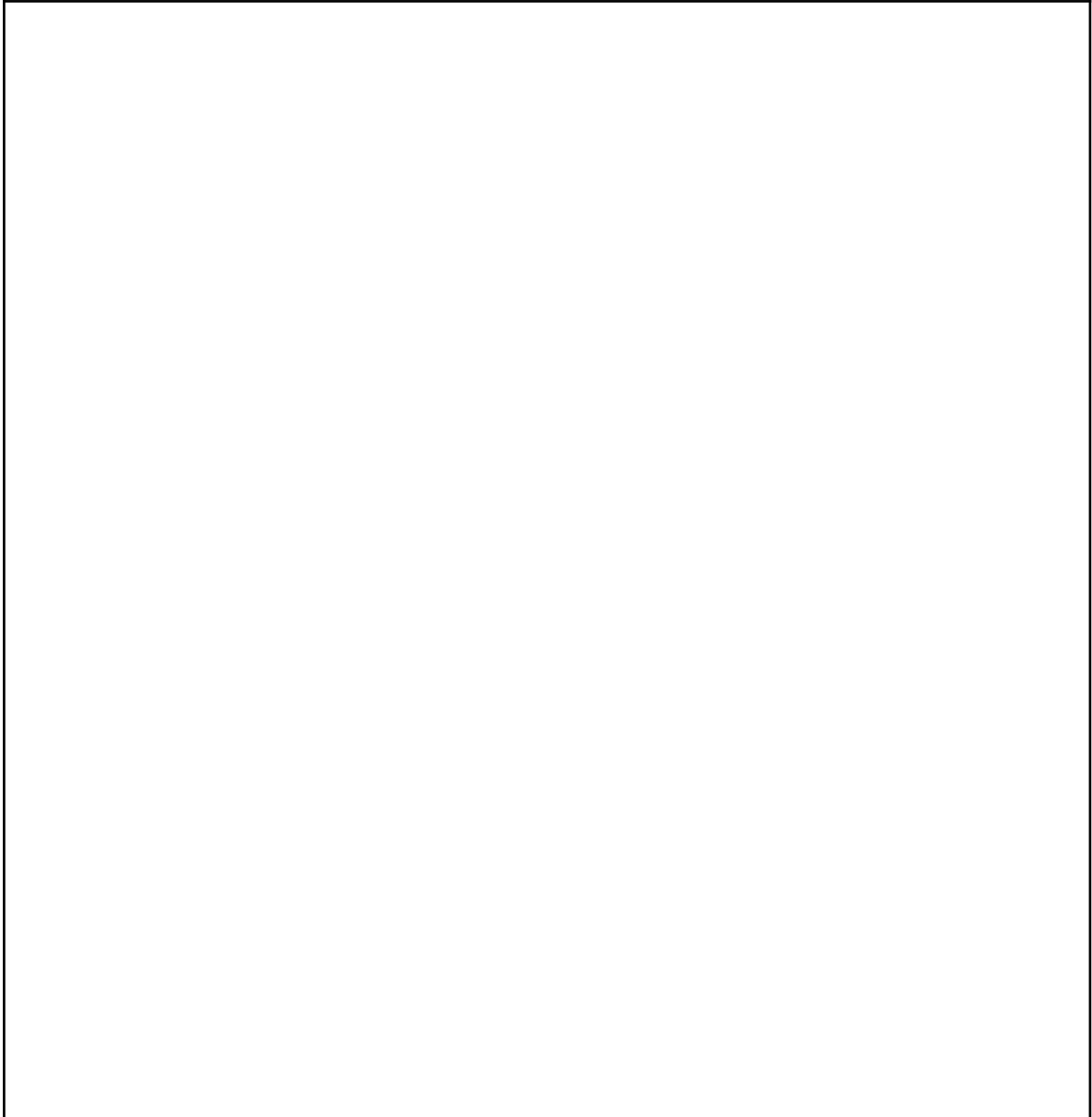
I. Source code:

SOURCE CODE



A large, empty rectangular box with a thin black border, occupying the central portion of the page. It is intended for the main content of the document.

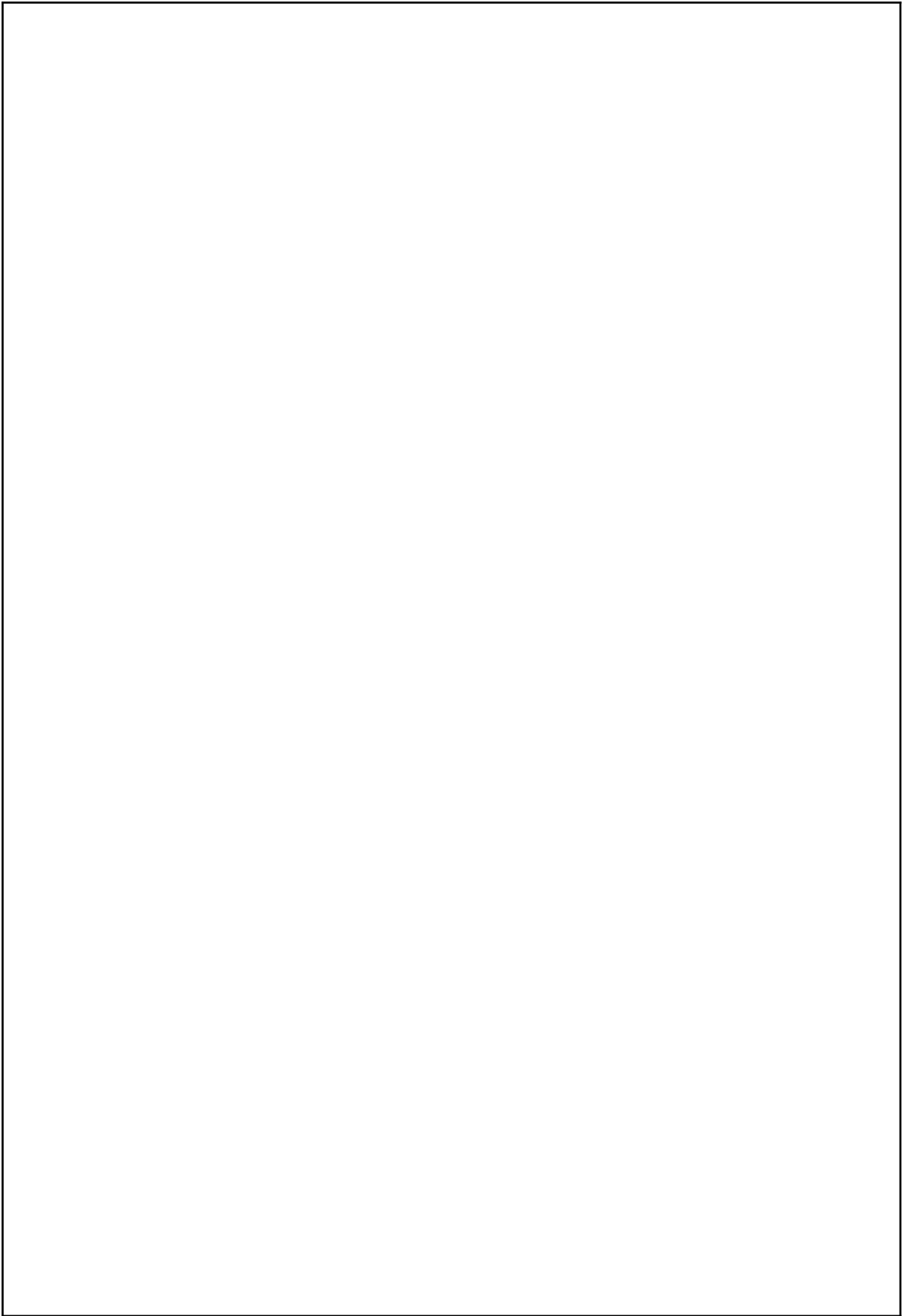
OUTPUT



J. Practical related Exercises.

1. Design page to implement Add to cart functionality for restaurant.
2. Design page to send one message using button control from parent control to child control and vice-versa using component communication concept.





K. References Links

1. <https://angular.io/guide/component-interaction>
2. <https://angular.io/guide/inputs-outputs>
3. <https://www.netjstech.com/2020/04/angular-custom-property-binding-using-input-decorator.html>
4. <https://www.c-sharpcorner.com/blogs/how-components-communicate-with-each-other-in-angular>
5. <https://www.youtube.com/watch?v=I8Z8g9APaDY>
6. <https://www.youtube.com/watch?v=5V77yVzzv7A>

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of Input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No 11: Develop page to demonstrate different methods of angular component lifecycle.

A. Objective: Main objective of angular component lifecycle are component initialization to initialize component before it renders to page, Change Detection mechanism to detect changes in the component's properties and update the view accordingly, Content Projection used to project content into their views using content projection, View Initialization and Update and Destruction of component.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.

2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.

3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.

4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.

C. Expected Skills to be developed based on competency:

D. Expected Course Outcomes (Cos)

Utilize angular template driven and reactive forms in different problem solutions.

E. Practical Outcome (PRo)

Learner will be able to utilize different methods of the angular component lifecycle.

F. Expected Affective domain Outcome (ADos)

1. Follow Coding standards and practices.
2. Maintain tools and equipment.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory:

Angular components have a series of lifecycle events that are executed in a specific order during their creation, rendering, and destruction. These lifecycle events allow developers to control and respond to changes in the state of the component.

The following is the order of the Angular component lifecycle events:

ngOnChanges: This event is executed when the input properties of a component change. It receives a `SimpleChanges` object that contains the current and previous values of the input properties.

ngOnInit: This event is executed once, immediately after the first `ngOnChanges` event. It is used to initialize the component's properties.

ngDoCheck: This event is executed every time Angular performs change detection. It allows developers to detect and respond to changes that Angular may have missed.

ngAfterContentInit: This event is executed after the component's content has been projected into its view. It is used to perform initialization tasks that depend on the component's content.

ngAfterContentChecked: This event is executed every time Angular checks the content of the component. It allows developers to perform additional checks or updates after the content has been checked.

ngAfterViewInit: This event is executed after the component's view has been initialized. It is used to perform initialization tasks that depend on the component's view.

ngAfterViewChecked: This event is executed every time Angular checks the view of the component. It allows developers to perform additional checks or updates after the view has been checked.

ngOnDestroy: This event is executed just before the component is destroyed. It is used to perform cleanup tasks such as unsubscribing from observables or removing event listeners.

By using these lifecycle events, developers can build more robust and responsive Angular applications.

Below example for implementing the Angular component lifecycle:

Create a new Angular project using the Angular CLI.

Create a new component called "LifecycleComponent" using the CLI command: `ng generate component Lifecycle`

In the "LifecycleComponent" class, implement the following lifecycle hooks:

`ngOnInit`

`ngOnChanges`

`ngDoCheck`

`ngAfterContentInit`

`ngAfterContentChecked`

`ngAfterViewInit`

`ngAfterViewChecked`

`ngOnDestroy`

In each lifecycle hook method, add a `console.log` statement to output a message indicating which hook has been called.

Add the "LifecycleComponent" to the "AppComponent" template.

Test the component by running the application and observing the console output.

Here's an example implementation of the "LifecycleComponent" class:

```
import { Component, OnInit, OnChanges, DoCheck, AfterContentInit,
AfterContentChecked, AfterViewInit, AfterViewChecked, OnDestroy } from
'@angular/core';
```

```
@Component({
  selector: 'app-lifecycle',
  templateUrl: './lifecycle.component.html',
  styleUrls: ['./lifecycle.component.css']
})
export class LifecycleComponent implements OnInit, OnChanges, DoCheck,
AfterContentInit, AfterContentChecked, AfterViewInit, AfterViewChecked,
OnDestroy {
```

```
  constructor() {}
```

```
  ngOnInit(): void {
    console.log('ngOnInit');
  }
```

```
  ngOnChanges(): void {
    console.log('ngOnChanges');
  }
```

```
  ngDoCheck(): void {
```

```
    console.log('ngDoCheck');
  }

  ngAfterContentInit(): void {
    console.log('ngAfterContentInit');
  }

  ngAfterContentChecked(): void {
    console.log('ngAfterContentChecked');
  }

  ngAfterViewInit(): void {
    console.log('ngAfterViewInit');
  }

  ngAfterViewChecked(): void {
    console.log('ngAfterViewChecked');
  }

  ngOnDestroy(): void {
    console.log('ngOnDestroy');
  }
}
```

And here's an example implementation of the "AppComponent" template:

```
<div>
  <h1>Angular Component Lifecycle Exercise</h1>
  <app-lifecycle></app-lifecycle>
</div>
```

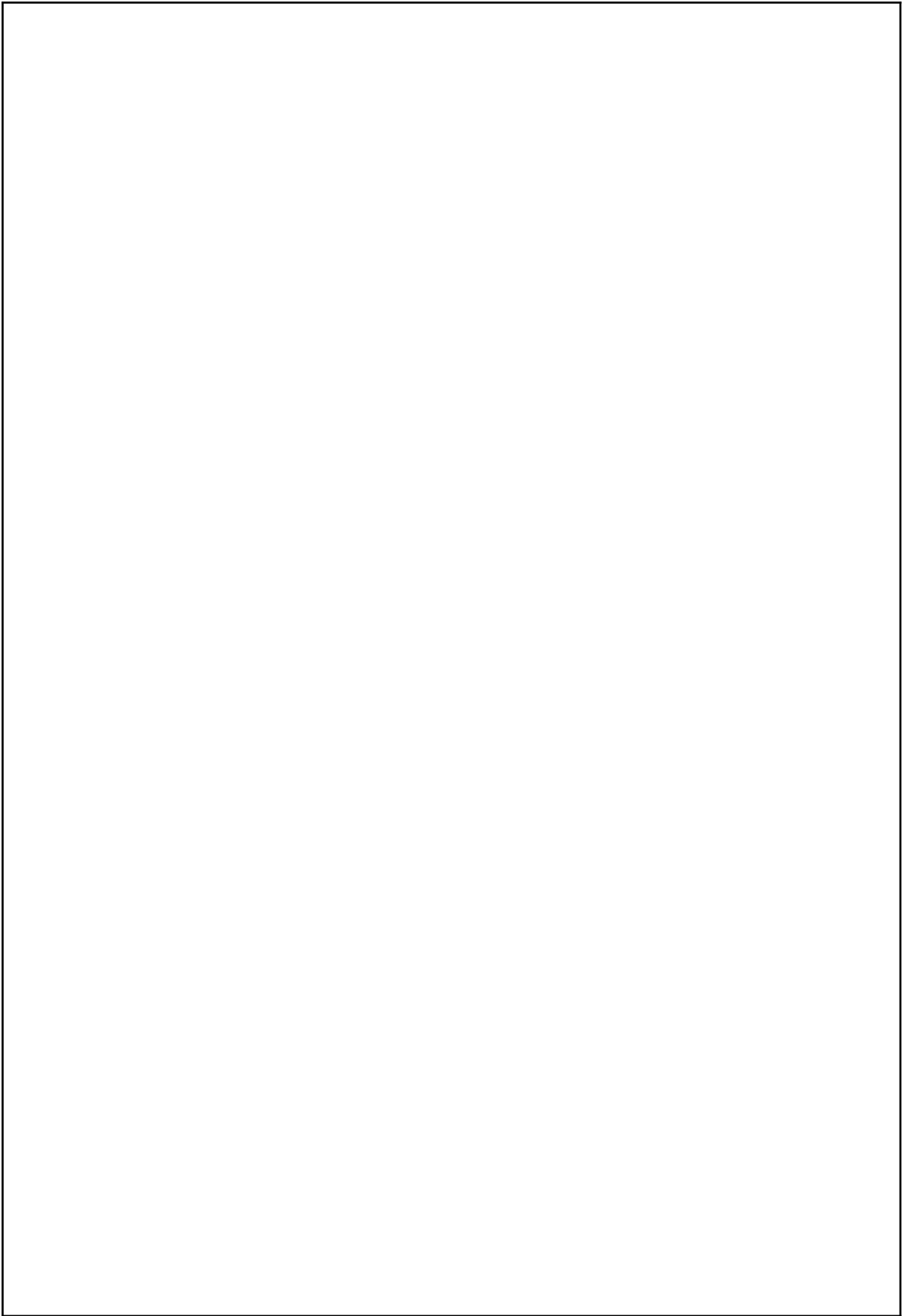
H. Resources/Equipment Required

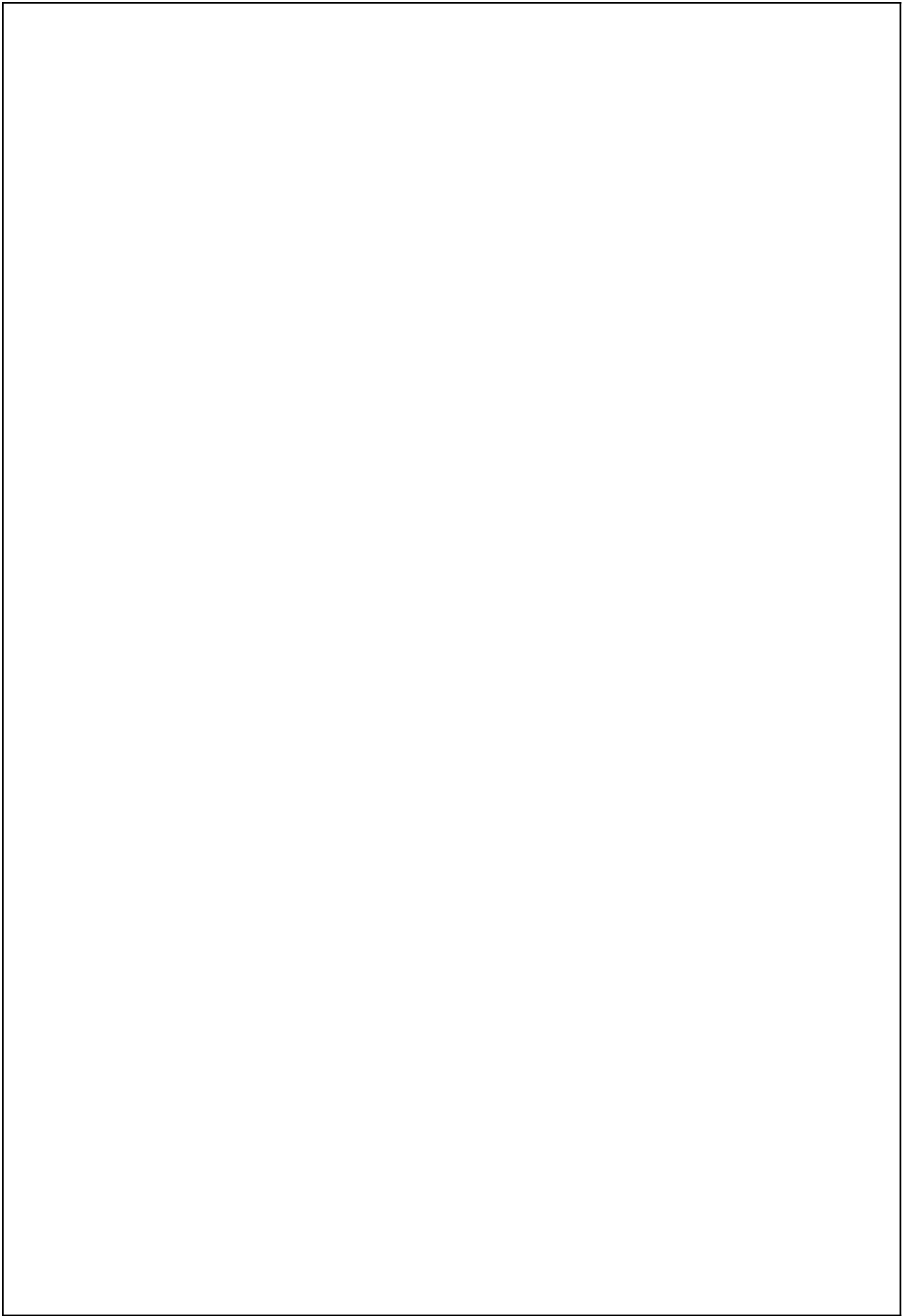
S r. N o.	Equipment/ Software Resources	Specification

1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

I. Source code:

SOURCE CODE

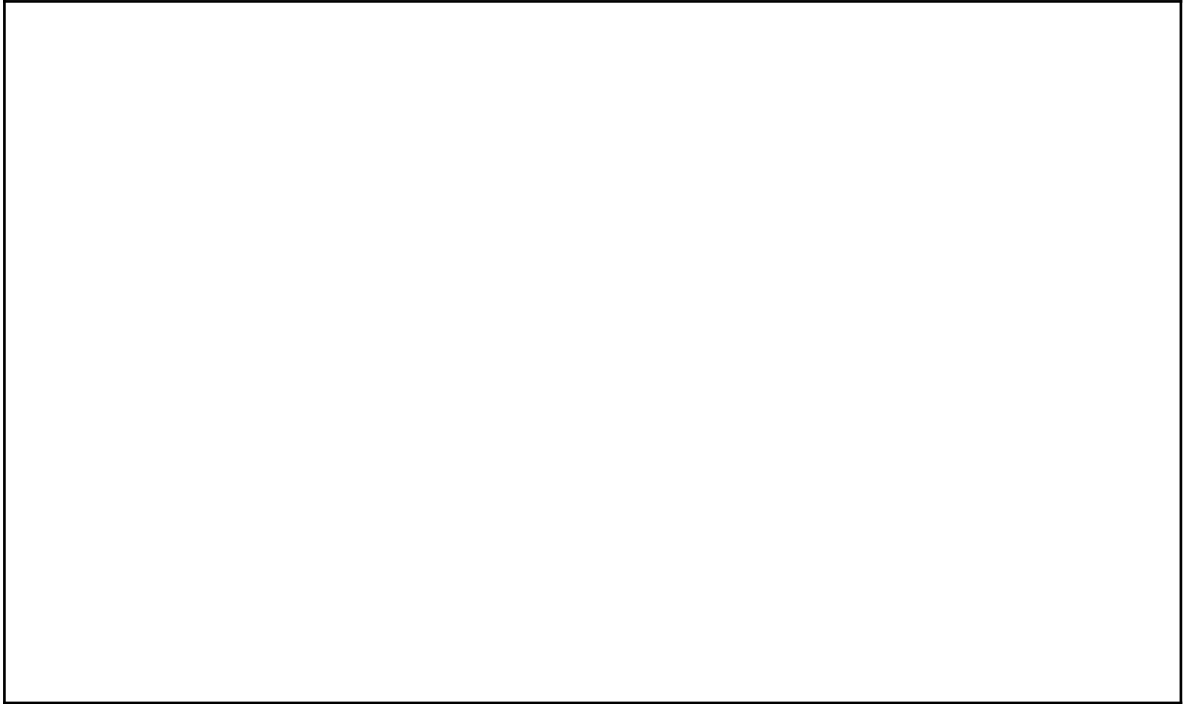




A large, empty rectangular box with a thin black border, occupying the central portion of the page. It is intended for a drawing or a detailed written response.

A large, empty rectangular box with a thin black border, occupying the central portion of the page. It is intended for a drawing or a detailed written response.

OUTPUT



J. Practical related Exercises.

1. Design a page to demonstrate usage of angular component Lifecycle.
2. Design a component to calculate area of circle use different methods to initialize default values and radius of circle.



A large, empty rectangular box with a thin black border, occupying the central portion of the page. It is intended for the main content of the document.

A large, empty rectangular box with a thin black border, occupying the central portion of the page. It is intended for a drawing or a detailed written response.

A large, empty rectangular box with a thin black border, occupying the central portion of the page. It is intended for a drawing or a detailed written response.

K. References Links

1. <https://angular.io/guide/lifecycle-hooks>
2. <https://www.c-sharpcorner.com/article/life-cycle-of-angular-components/>
3. <https://codecraft.tv/courses/angular/components/lifecycle-hooks/>
4. <https://www.javatpoint.com/angular-lifecycle>
5. <https://indepth.dev/posts/1494/complete-guide-angular-lifecycle-hooks>

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of Input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No 12: Design an e-commerce product page and product details page that displays product details when clicking on any particular product.

A. Objective:

Services are an important part of Angular development, providing a way to encapsulate functionality, manage dependencies, and promote code reusability. the Router, ActivatedRoute, and param parameters allowing for the implementation of dynamic routing, state management, access to route parameters and enabling URL management.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.
2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.
3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.
4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.

C. Expected Skills to be developed based on competency:

Learner can manage Service, router and Active Router facilities and to pass data from one component to another via the URL that can handle different user inputs and scenarios.

D. Expected Course Outcomes(Cos)

Utilize angular template driven and reactive forms in different problem solutions.

E. Practical Outcome(Pro)

Learner will be able to encapsulate and modularize functionality that can be used across components and modules, making it easier to reuse code and avoid duplication, enables navigation from one component to another, without reloading the page, managing URLs and handling redirects to make real time user interactive application.

F. Expected Affective domain Outcome(ADos)

1. Follow Coding standards and practices.

2. Maintain tools and equipment.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory:

Service Concept:

Service is a class that is used to encapsulate functionality that can be shared across components. Services are typically used to handle business logic, API calls, and other non-UI related tasks. Here's an example of a service with two methods and how it can be used in a component:

Service: UserService

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  private users = [
    { id: 1, name: 'abc', email: 'abc@example.com' },
    { id: 2, name: 'pqr', email: 'pqr@example.com' },
    { id: 3, name: 'xyz', email: 'xyz@example.com' },
  ];

  getUsers() {
    return this.users;
  }

  getUserById(id: number) {
    return this.users.find(user => user.id === id);
  }
}
```

This UserService has two methods, getUsers() and getUserById(), which respectively return all the users and a specific user by ID.

Component: UserComponent

```
import { Component, OnInit } from '@angular/core';
import { UserService } from './user.service';
```

```

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrls: ['./user.component.css']
})
export class UserComponent implements OnInit {
  users = [];

  constructor(private userService: UserService) { }

  ngOnInit(): void {
    this.users = this.userService.getUsers();
  }

  getUserById(id: number) {
    return this.userService.getUserById(id);
  }
}

```

In this `UserComponent`, the `UserService` is injected into the constructor using Dependency Injection. In the `ngOnInit()` method, the `getUsers()` method of the `UserService` is called to get all the users, which are then assigned to the `users` property of the component. The `getUserById()` method is also defined in the component, which calls the `getUserById()` method of the `UserService` to get a specific user by ID.

To summarize, the `UserService` is a reusable class that encapsulates functionality related to user management, and it can be used in multiple components by injecting it using Dependency Injection. In this example, the `UserComponent` uses the `getUsers()` and `getUserById()` methods of the `UserService` to get all users and a specific user by ID.

In Angular, the Router is a built-in service that allows for navigation between different views of the application based on the URL. The `ActivatedRoute` provides information about the currently activated route, while the `router.navigate()` method is used to navigate to a different route programmatically. Here's an example of how to use these concepts in Angular:

Routing Configuration: `app-routing.module.ts`

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

```

```

import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';
import { ContactComponent } from './contact/contact.component';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

In this routing configuration, we define three routes: the home route (''), the about route ('about'), and the contact route ('contact'). Each route corresponds to a component: HomeComponent, AboutComponent, and ContactComponent, respectively.

Component: home.component.ts

```

import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {

  constructor(private router: Router) { }

  ngOnInit(): void {
  }

  goToAboutPage() {
    this.router.navigate(['/about']);
  }
}

```

```

    goToContactPage() {
      this.router.navigate(['/contact']);
    }
  }
}

```

In this HomeComponent, we inject the Router service into the constructor using Dependency Injection. We define two methods, `goToAboutPage()` and `goToContactPage()`, which use the `router.navigate()` method to navigate to the about and contact pages, respectively.

Component: `home.component.html`

```

<div>
  <h1>Welcome to the Home Page!</h1>
  <button (click)="goToAboutPage()">Go to About Page</button>
  <button (click)="goToContactPage()">Go to Contact Page</button>
</div>

```

In this HomeComponent HTML template, we use two buttons that are bound to the `goToAboutPage()` and `goToContactPage()` methods defined in the TypeScript file. When a user clicks one of these buttons, the corresponding method is called, which uses the `router.navigate()` method to navigate to the specified route.

Here ContentComponent contain the same thing as HomeComponent except the description of it.

Overall, the Router, ActivatedRoute, and `router.navigate()` method are essential components of Angular development, allowing for dynamic routing, state management, and navigation guards. In this example, we demonstrated how to use these concepts to navigate between different views of the application based on the URL.

H. Resources/Equipment Required

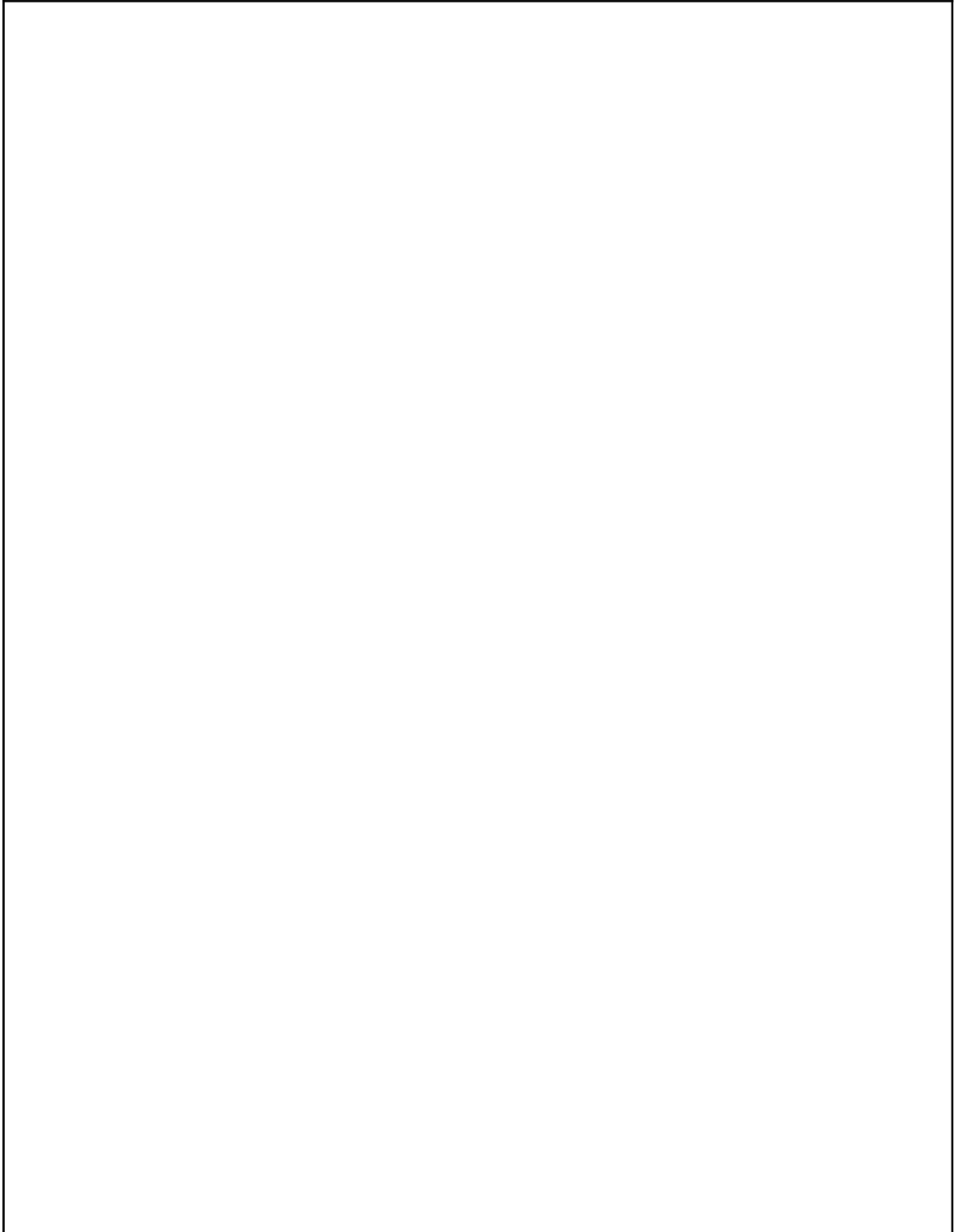
S r. N o.	Equipment/ Software Resources	Specification

1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

I. Program Source code Output

SOURCE CODE

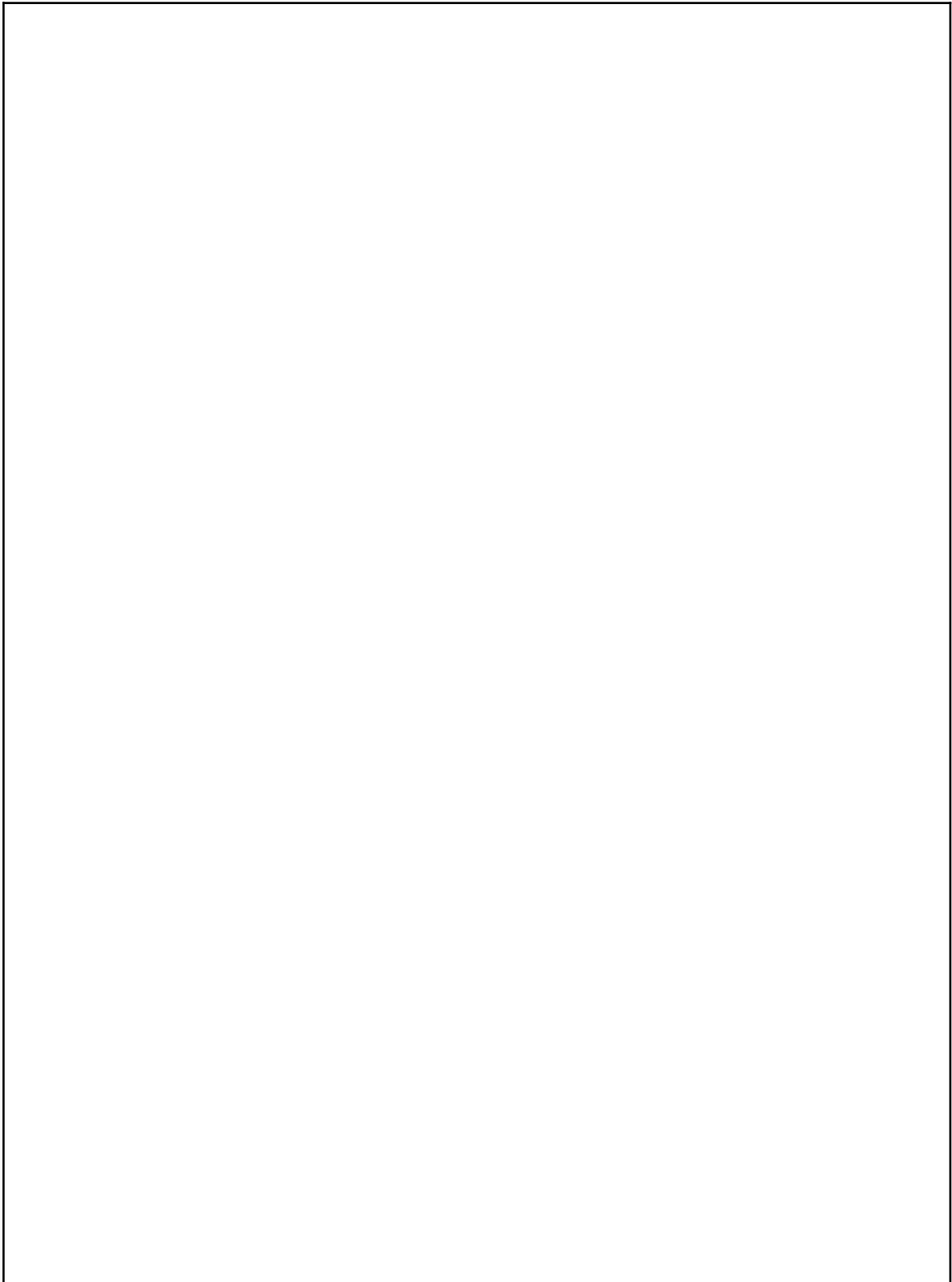
OUTPUT

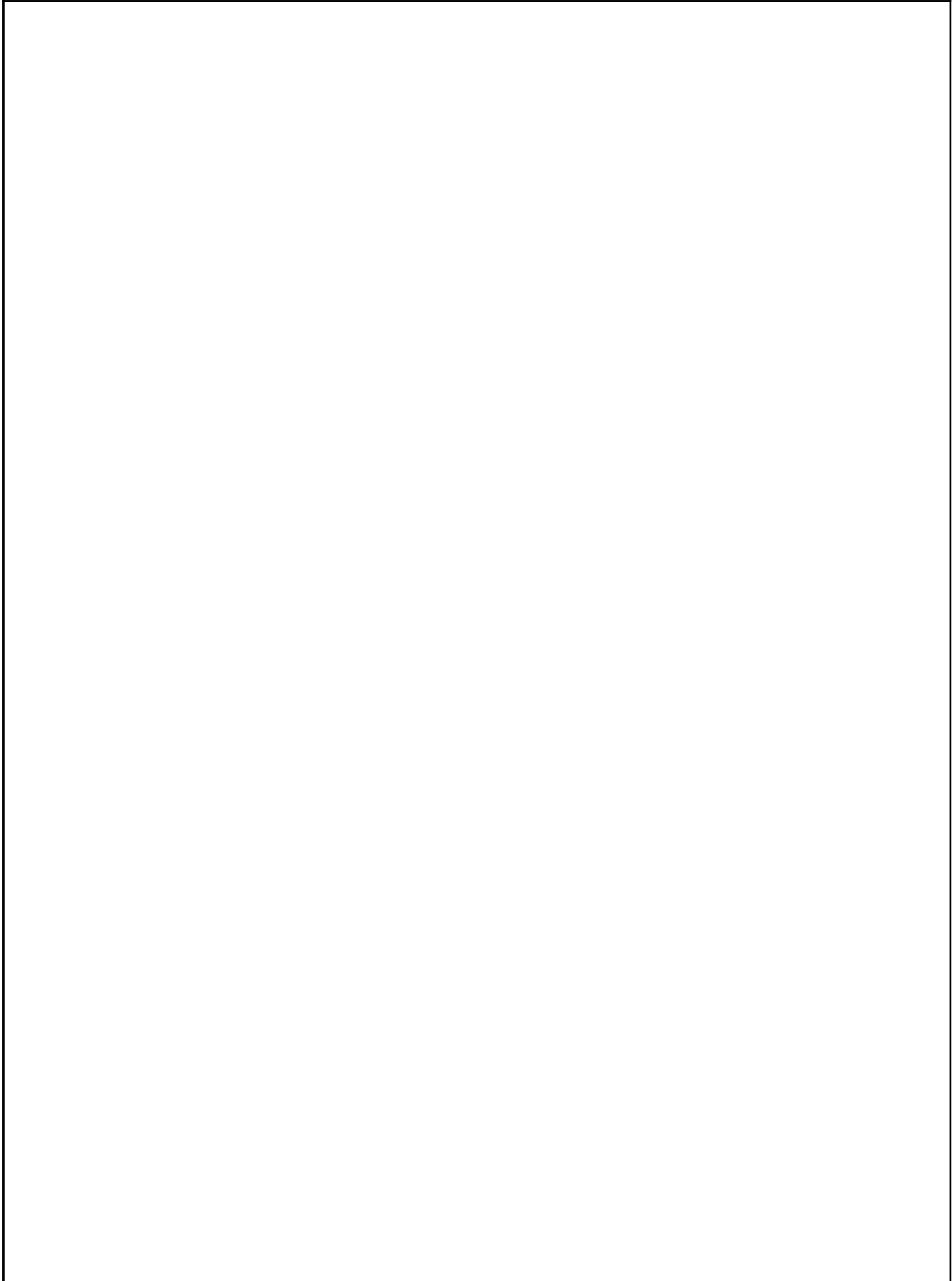


J. Practical related Exercises.

1. Design Component to find name of student from property object created in service using textbox and button control in html.

2. Design a Book Gallery and Book Detail page using service and routing concepts.

A large, empty rectangular box with a thin black border, intended for a student to draw or design a book gallery and book detail page. The box occupies the majority of the page area below the question.



K. References Links

1. https://www.tutorialspoint.com/angular7/angular7_services.htm

2. <https://www.simplilearn.com/tutorials/angular-tutorial/angular-service>
3. <https://angular.io/guide/router-tutorial>
4. https://www.tutorialspoint.com/angular8/angular8_routing_and_navigation.htm
5. <https://www.youtube.com/watch?v=H-1Pb9zXmkY>
6. <https://www.youtube.com/watch?v=lsWZh9ohDgM>

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of Input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No.13: Design a page to display student information using dependency Injection.

- A. Objective:** GET and POST web APIs is important task to develop any dynamic single page application so here students details will be displayed on page using web APIs by students.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.
2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.
3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.
4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.
5. **Life-long learning:** Ability to analyze individual needs and engage in updating in the context of technological changes in field of engineering.

C. Expected Skills to be developed based on competency:

1. Develop service to use dependency injection.
2. Design a page to display student details using angular component and service.

D. Expected Course Outcomes(Cos)

Design pages to make HTTP GET/POST calls to perform CRUD operations using different server-side APIs.

E. Practical Outcome(Pro)

Design a page to display student information using dependency Injection.

F. Expected Affective domain Outcome (ADos)

1. Maintain tools and equipment.
2. Follow Coding standards and practices.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory:Dependency Injection (DI) in Angular is a design pattern that allows us to create loosely coupled components by providing dependencies to a component at the time of creation. In other words, DI is a way of providing objects that a class needs (its dependencies) from an external source.

In Angular, we use DI to inject the required services or dependencies into a component, directive, or any other class that needs them. This is done by adding the dependencies as parameters to the class constructor. When the component is instantiated, Angular creates and injects an instance of the required dependencies into the component.

Using DI in Angular has several benefits, including:

Loosely coupled components: DI helps to create components that are independent of each other and can be easily reused.

Testability: Since the dependencies are passed in as parameters, it's easy to provide mock or stub implementations of the dependencies for unit testing.

Code maintainability: DI makes it easy to manage the dependencies of a component and update them if needed.

Scalability: DI allows for easy scaling of an application by making it possible to easily add or remove dependencies as needed.

To use DI in Angular, we first need to define the dependencies as services and then inject them into the components that need them. This is done using the `@Injectable()` decorator for services and adding the required services as parameters to the constructor of the component that needs them. Below is example of student info display with dependency injection.

StudentService.ts

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class StudentService {

  public students = [
    { id: 1, name: 'Alpesh Thaker', major: 'Computer Engineering' },
    { id: 2, name: 'Umang Shah', major: 'Biology' },
    { id: 3, name: 'Yagnik Tank', major: 'Mathematics' }
  ];
}
```

```
getStudents() {  
    return this.students;  
}
```

```
getStudentById(id: number) {  
    return this.students.find(student => student.id === id);  
}  
}
```

student-information.component.ts

```
import { Component, OnInit } from '@angular/core';  
import { StudentService } from '../student.service';
```

```
@Component({  
    selector: 'app-student-information',  
    templateUrl: './student-information.component.html',  
    styleUrls: ['./student-information.component.css']  
})  
export class StudentInformationComponent implements OnInit {  
  
    public students:any;  
  
    constructor(private studentService: StudentService) {}  
  
    ngOnInit() {  
        this.students = this.studentService.getStudents();  
    }  
}
```


student-information.component.html

```
<p>student-information works!</p>
<div *ngFor="let student of students">
  <h3>{{ student.name }}</h3>
  <p>Major: {{ student.major }}</p>
</div>
```

H. Resources/Equipment Required

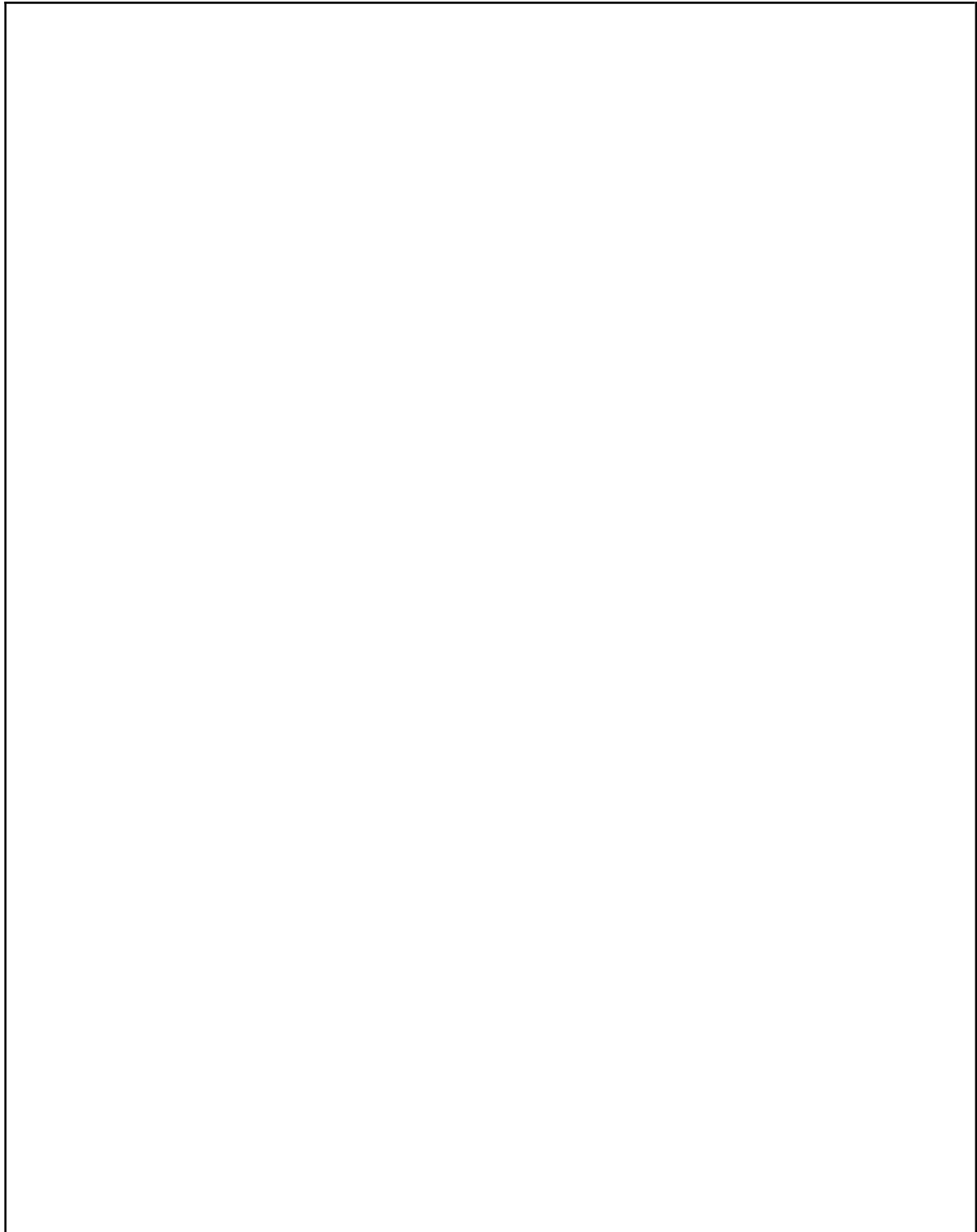
Sr. No	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

I. Source code:

SOURCE CODE

A large, empty rectangular box with a thin black border, occupying the central portion of the page. It is intended for the main content of the document.

OUTPUT



J. Practical related Questions.

Design web page to display product data in table using HTTP GET/POST Calls from web APIs.

Design web page to display Employee data in table using HTTP GET/POST Calls from web APIs.

K. References Links

1. [Angular - Understanding dependency injection](#)
2. [Angular Dependency Injection Explained with Examples \(freecodecamp.org\)](#)
3. [A Complete Guide To Angular Dependency Injection || Simplilearn](#)
4. [Angular 2 - Dependency Injection \(tutorialspoint.com\)](#)
5. [How dependency injection works in Angular - LogRocket Blog](#)
6. [A Complete Guide To Angular Dependency Injection || Simplilearn](#)
7. [\(492\) Angular Tutorial - 18 - Dependency Injection - YouTube](#)

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No.14: Develop a page for product listing and search-as-you type using observables and web APIs from database.

- A. Objective:** The objective of implementing a search-as-you-type feature using observables and web APIs from a database in Angular is to enhance the user experience by providing real-time search results as the user types into the search bar. The use of observables in Angular allows for asynchronous data streams to be

handled in a reactive manner, ensuring that the search results are always up-to-date and responsive to user input.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.
2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.
3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.
4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.
5. **Life-long learning:** Ability to analyze individual needs and engage in updating in the context of technological changes in field of engineering.

C. Expected Skills to be developed based on competency:

1. Knowledge of components, modules, services, and observables.
2. Developing search-as-you-type functionality requires a strong understanding of RxJS operators and how to use them effectively.

D. Expected Course Outcomes (Cos)

Design pages to make HTTP GET/POST calls to perform CRUD operations using different server-side APIs.

E. Practical Outcome (PRo)

Develop a page for product listing and search-as-youtype using observables and web APIs from database.

F. Expected Affective domain Outcome (ADos)

1. Maintain tools and equipment.
2. Follow Coding standards and practices.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory:

Observables: In Angular, observables are a powerful feature that allow developers to work with asynchronous data streams. An observable is a representation of a stream of events that can be subscribed to by one or more observers. It provides a way to handle asynchronous operations and to propagate changes across an application.

Observables are used extensively in Angular for handling events such as HTTP requests, user interactions, and data updates. They can be used to manage the flow of data between components, services, and other parts of an application.

Some key features of observables in Angular include:

1. **Asynchronous data handling:** Observables allow developers to work with asynchronous data streams, such as HTTP requests, in a more efficient and reliable manner.
2. **Easy to compose:** Observables can be composed and combined to create more complex data streams, which makes it easier to handle complex scenarios and data flows.
3. **Cancellation support:** Observables can be cancelled, which helps to prevent memory leaks and reduce unnecessary processing.
4. **Error handling:** Observables have built-in error handling support, which makes it easier to handle errors and exceptions that may occur during asynchronous operations.
5. **Support for multiple values:** Observables can emit multiple values over time, which makes them well-suited for handling real-time data streams.

In summary, observables in Angular are a powerful feature that enable developers to handle asynchronous data streams in a more efficient and reliable manner. They provide a flexible and composable way to manage the flow of data in an application, and are used extensively in Angular for handling events and data updates.

RXJS Operators: `DebounceTime()`, `distinctUntilChanged()`, and `switchMap()` are three RxJS operators commonly used together in Angular for implementing search-as-you-type functionality using Observables.

1. **`debounceTime()`:** This operator delays the emission of items from an Observable for a specified period of time after the last emission. It is commonly used to prevent unnecessary network requests when a user is typing quickly in a search input field. For example, if you specify a debounce time of 300 milliseconds, the operator will wait for 300 milliseconds after the last keystroke before emitting the value.
2. **`distinctUntilChanged()`:** This operator filters out duplicate values emitted by an Observable. It compares the current value emitted with the previous value emitted, and if they are the same, it suppresses the emission of the current value.

This is useful in search-as-you-type scenarios to avoid making unnecessary network requests when the user types the same value repeatedly.

3. `switchMap()`: This operator maps each value emitted by an Observable to another Observable, and then flattens the resulting Observable into a single stream of values. This is useful in search-as-you-type scenarios because it allows you to cancel previous network requests and only return the results for the most recent search term entered by the user.

Now search-as-you-type functionality can be implemented using observables. The basic idea is to listen for changes to the search input field and use an observable to update the search results in real-time.

Here are the steps to implement search-as-you-type functionality using observables in Angular:

1. Set up the search input field: Create a text input field in your template and bind it to a property in your component using `[(ngModel)]`.
2. Create an Observable: Use the RxJS library to create an Observable that listens to changes in the search input field.
3. Handle the Observable: In the Observable's `subscribe()` method, use the value of the search input field to make an HTTP request to your server-side API and retrieve the search results.
4. Update the search results: Use Angular's change detection mechanism to update the search results in the template as soon as new results are retrieved from the server.

Here's some sample code to help illustrate how this works In your component:

```
import { Component } from '@angular/core';
import { Observable, Subject } from 'rxjs';
import { debounceTime, distinctUntilChanged, switchMap } from 'rxjs/operators';
import { ApiService } from './api.service';

@Component({
  selector: 'app-search',
  template: `
    <input type="text" [(ngModel)]="searchTerm"
    placeholder="Search">
```

```
<ul>
  <li *ngFor="let result of searchResults">{{result}}</li>
</ul>
`,
})
export class SearchComponent {
  searchTerm: string;
  searchResults: string[];
  private searchTerms = new Subject<string>();

  constructor(private apiService: ApiService) {}

  ngOnInit(): void {
    this.searchTerms.pipe(
      debounceTime(300),
      distinctUntilChanged(),
      switchMap((term: string) => this.apiService.search(term))
    ).subscribe((results: string[]) => {
      this.searchResults = results;
    });
  }

  search(term: string): void {
    this.searchTerms.next(term);
  }
}
```

In your API service:

```
import { Injectable } from '@angular/core';
```

```

import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root',
})
export class ApiService {
  constructor(private http: HttpClient) {}

  search(term: string): Observable<string[]> {
    return this.http.get<string[]>(`/api/search?term=${term}`);
  }
}

```

In the example above, the searchTerms property is a Subject that emits new search terms whenever the search input field changes. These search terms are then piped through a series of operators, including debounceTime(), distinctUntilChanged(), and switchMap(). The switchMap() operator makes an HTTP request to the server-side API to retrieve the search results, and then the results are updated in the searchResults property of the component.

H. Resources/Equipment Required

Sr. No.	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

I. Source code:

SOURCE CODE

OUTPUT

J. Practical related Questions.

Design web page to implement searching facility for library book searching assume books table with suitable fields available in database.

Design web page to

K. References Links

1. [HTTP Example with Observables • Angular \(codecraft.tv\)](#)
2. [\(540\) RxJS Type Ahead search | RxJS Autocomplete | Angular RxJS search | Debounce search 🐱 - YouTube](#)
3. [Consuming events as Observables in Angular 2 | egghead.io](#)
4. [How To Build a Search Bar with RxJS | DigitalOcean](#)
5. [typescript - Angular v4: Search by name with observable - Stack Overflow](#)

L. Assessment-Rubrics

S.No .	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No.15: Design web page to display student data in table using HTTP GET/POST Calls from web APIs.

- A. **Objective:** GET and POST web APIs is important task to develop any dynamic single page application so here students details will be displayed on page using web APIs by students.
- B. **Expected Program Outcomes (POs)**

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the engineering problems.
2. **Problem analysis:** Identify and analyse well-defined engineering problems using codified standard methods.
3. **Design/ development of solutions:** Design solutions for engineering well-defined technical problems and assist with the design of systems components or processes to meet specified needs.
4. **Engineering Tools, Experimentation and Testing:** Apply modern engineering tools and appropriate technique to conduct standard tests and measurements.
5. **Life-long learning:** Ability to analyze individual needs and engage in updating in the context of technological changes in field of engineering.

C. Expected Skills to be developed based on competency:

1. Develop GET and POST web APIs in PHP and MYSQL.
2. Design a page to display student details using angular component and service.

D. Expected Course Outcomes(Cos)

Design pages to make HTTP GET/POST calls to perform CRUD operations using different server-side APIs.

E. Practical Outcome(PRo)

Design web page to display student data in table using HTTP GET/POST Calls from web APIs.

F. Expected Affective domain Outcome(ADos)

1. Maintain tools and equipment.
2. Follow Coding standards and practices.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory: Web APIs are also called RESTFUL APIs. REST (Representational State Transfer) is a popular architectural style for creating web services, and it uses HTTP methods to perform various CRUD (Create, Read, Update, Delete) operations on resources. In this answer, I'll provide a basic theory for creating POST and GET RESTful APIs using PHP and MySQL.

To start with, we need to create a MySQL database and a table to store the data that we want to expose through our APIs. For example, let's assume that we want to

create an API to manage a list of users, so we can create a MySQL database with a table named "users" having columns like "id", "name", "email", "phone", etc.

Now let's see how we can create a POST API to add a new user to our database using PHP.

POST API:

The HTTP POST method is used to create a new resource. To create a new user, we need to send a POST request to our API endpoint with the user data in the request body. Here's an example of how we can implement this in **students.php**:

```
<?php

$dbhost = "localhost";
$dbname = "angular";
$username = "root";
$password = "";

$conn = mysqli_connect($dbhost, $username, $password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// Get all students
function getstudents($conn) {
    $sql = "SELECT * FROM students";
    $result = mysqli_query($conn, $sql);

    $students = array();

    if (mysqli_num_rows($result) > 0) {
        while($row = mysqli_fetch_assoc($result)) {
            $students[] = $row;
        }
    }
}
```

```
    }  
    }  
  
    return $students;  
}  
  
// Get a single student by ID  
function getStudent($conn, $id) {  
    $sql = "SELECT * FROM students WHERE id = $id";  
    $result = mysqli_query($conn, $sql);  
  
    $students = mysqli_fetch_assoc($result);  
  
    return $students;  
}  
  
$method = $_SERVER['REQUEST_METHOD'];  
echo $_SERVER['PATH_INFO'];  
$request = explode('/', trim($_SERVER['PATH_INFO'], '/'));  
  
switch ($method) {  
    case 'GET':  
        if ($request[0] == 'Students') {  
            if (isset($request[1])) {  
                $response = getStudent($conn, $request[1]);  
            } else {  
                $response = getstudents($conn);  
            }  
        } else {  
            $response = getstudents($conn);  
        }  
    }  
}
```

```
        // Handle other endpoints
    }
    break;
case 'POST':
    // Handle POST requests
    break;
case 'PUT':
    // Handle PUT requests
    break;
case 'DELETE':
    // Handle DELETE requests
    break;
}
```

```
echo json_encode($response);
```

In the above example, GET method demonstrated to generate response of students in json format and below details describe angular example.

Sample Angular code for designing a web page to display student data in a table using HTTP GET/POST calls from web APIs:

1. First, create a new Angular component using the Angular CLI:

Css code

ng generate component student-table

2. In the **student-table.component.html** file, create a table to display the student data:

```
<table>
<thead>
<tr>
<th>ID</th>
```

```
<th>Name</th>
<th>Email</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr *ngFor="let student of students">
<td>{{ student.id }}</td>
<td>{{ student.name }}</td>
<td>{{ student.email }}</td>
<td>{{ student.phone }}</td>
</tr>
</tbody>
</table>
```

3. In the **student-table.component.ts** file, import the **HttpClient** module from **@angular/common/http**, inject it into the constructor, and create a method to fetch the student data using a GET call to a web API:

Typescript code

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-student-table',
  templateUrl: './student-table.component.html',
  styleUrls: ['./student-table.component.css']
})
export class StudentTableComponent implements OnInit {
  students: any[];
```

```
    constructor(private http: HttpClient) {}

    ngOnInit() {

        this.http.get<any[]>('http://localhost/api/students.php/students').subscribe
        (
            data => {
                this.students = data;
            },
            error => {
                console.log(error);
            }
        );
    }
}
```

In the **app.module.ts** file, import the **HttpClientModule** module from **@angular/common/http** and add it to the **imports** array:

typescript code

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';
```

```
@Component({
    selector: 'app-student-table',
    templateUrl: './student-table.component.html',
    styleUrls: ['./student-table.component.css']
})
export class StudentTableComponent implements OnInit {
    students: any[];
```

```

    constructor(private http: HttpClient) {}

    ngOnInit() {

        this.http.get<any[]>('http://localhost/api/students.php/student').subscribe(
            data => {
                this.students = data;
            },
            error => {
                console.log(error);
            }
        );
    }
}

```

In the **app.component.html** file, add the **app-student-table** selector to display the **StudentTableComponent**:

htmlCopy code

```
<app-student-table></app-student-table>
```

- Finally, start the Angular development server using **ng serve** and navigate to **http://localhost:4200** to view the student data in the table.

Note: You can also use a POST call to a web API to add new student data to the table. To do this, create a form in the **student-table.component.html** file to capture the student data and use the **HttpClient** module to make a POST call to the web API when the form is submitted.

H. Resources/Equipment Required

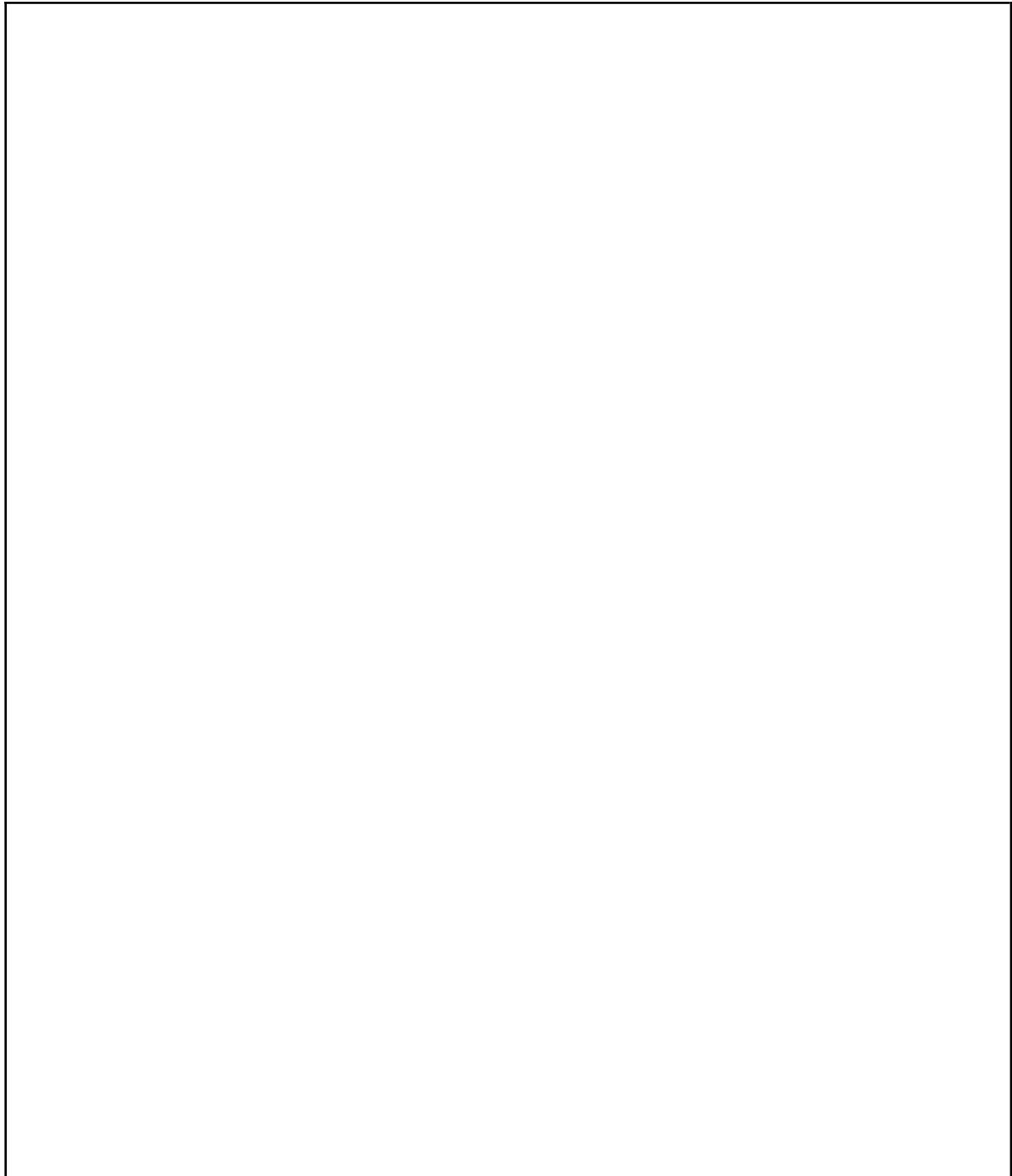
Sr. No	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.

2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

I. Source code:

SOURCE CODE

OUTPUT



J. Practical related Questions.

Design web page to display product data in table using HTTP GET/POST Calls from web APIs.

Design web page to display Employee data in table using HTTP GET/POST Calls from web APIs.

K. References Links

1. [Angular CRUD using PHP and MySQL – FahmidasClassroom](#)
2. [\(492\) Part-1 Angular 13 CRUD PHP MySql | Angular 13 PHP MySql Insert Update Delete | Angular 13 Tutorial - YouTube](#)
3. [\(492\) Part - 1 \(2022\). Angular PHP MySql CRUD operation | Angular 12 / 13 PHP MySql Insert Update Delete - YouTube](#)
4. [PHP + Angular + MySQL CRUD Example \(knowledgefactory.net\)](#)
5. [Angular 13 CRUD Using PHP and MySQL - Sathish kumar Ramalingam - Medium](#)

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No.16: Design web page to insert product data in table using web APIs.

A. Objective: POST web APIs is important task to develop any dynamic single page application so here products details will be inserted on database table on page using web APIs by students.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.
2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.
3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.
4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.
5. **Life-long learning:** Ability to analyze individual needs and engage in updating in the context of technological changes in field of engineering.

C. Expected Skills to be developed based on competency:

1. Develop POST web APIs for insert product in PHP and MYSQL.
2. Design a page to insert product details in angular component and service.

D. Expected Course Outcomes(Cos)

Design pages to make HTTP GET/POST calls to perform CRUD operations using different server-side APIs..

E. Practical Outcome(PRo)

Design web page to insert product data in table using web APIs.

F. Expected Affective domain Outcome(ADos)

1. Maintain tools and equipment.
2. Follow Coding standards and practices.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory: WebAPI for insert product details required to create in php and mysql as shown below.

1. Define the endpoint URL:

Let's assume that the endpoint URL for inserting product data will be: <http://localhost/products/insert.php>

2. Define the HTTP method:

In this case, we will be using the HTTP POST method to submit the product data.

3. Define the request parameters:

The product data to be inserted will be passed as POST parameters. The required parameters for inserting a new product might include: name: the name of the product (required), description: a short description of the product (optional), price: the price of the product (required), quantity: the quantity of the product available (optional)

4. Define the response format:

The API should return a JSON response indicating whether the insertion was successful or not, along with any error messages if applicable.

Here's an example of how you can implement the API in PHP:

```
<?php
```

```
// Connect to the database
```

```
$db = new mysqli('localhost', 'username', 'password', 'database_name');
```

```
// Check for errors
```

```
if ($db->connect_error) {
```

```
    die("Connection failed: " . $db->connect_error);
```

```
}
```

```
// Get the product data from POST parameters
```

```
$name = $_POST['name'];
```

```
$description = isset($_POST['description']) ? $_POST['description'] : '';
```

```
$price = $_POST['price'];
```

```
$quantity = isset($_POST['quantity']) ? $_POST['quantity'] : '';
```

```
// Insert the product data into the database
```



```
$sql = "INSERT INTO products (name, description, price, quantity) VALUES ('$name', '$description', $price, $quantity)";
```

```
if ($db->query($sql) === TRUE) {
```

```
    // Return a success message
```

```
    $response = array('status' => 'success', 'message' => 'Product inserted successfully.');
```

```
} else {
```

```
    // Return an error message
```

```
    $response = array('status' => 'error', 'message' => 'Error inserting product: ' . $db->error);
```

```
}
```

```
// Close the database connection
```

```
$db->close();
```

```
// Return the JSON response
```

```
header('Content-Type: application/json');
```

```
echo json_encode($response);
```

In the above example, POST method demonstrated to insert product with input injson format and below details describe angular example.

Sample Angular code for designing a web page to insert products data in a database table using HTTP POST calls from web APIs. We need to create an Angular component that will handle the insertion of product component for insert data using InsertProductComponent:

```
import { Component } from '@angular/core';
```

```
import { HttpClient } from '@angular/common/http';
```

```
@Component({
```

```
    selector: 'app-insert-product',
```

```
templateUrl: './insert-product.component.html',
styleUrls: ['./insert-product.component.css']
})
export class InsertProductComponent {

  product = {
    name: "",
    description: "",
    price: ""
  };

  constructor(private http: HttpClient) {}

  onSubmit() {
    this.http.post('http://localhost/products/insert.php', this.product)
      .subscribe(response => {
        console.log(response);
        // Insert logic to update table with new product data
      });
  }
}
```

Next, we need to create the HTML template for the InsertProductComponent component:

```
<h2>Insert Product</h2>
<form (ngSubmit)="onSubmit()">
  <div>
    <label for="name">Name</label>
    <input type="text" id="name" name="name" [(ngModel)]="product.name">
```

```

</div>

<div>

<label for="description">Description</label>

<input      type="text"      id="description"      name="description"
[(ngModel)]="product.description">

</div>

<div>

<label for="price">Price</label>

<input      type="number"      id="price"      name="price"
[(ngModel)]="product.price">

</div>

<button type="submit">Submit</button>

</form>

```

Finally, we need to update the table with the new product data after it has been successfully inserted:

```

import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-product-table',
  templateUrl: './product-table.component.html',
  styleUrls: ['./product-table.component.css']
})
export class ProductTableComponent {

  products = [];

  constructor(private http: HttpClient) {
    this.getProducts();
  }

```

```
    }

    getProducts() {
      this.http.get(http://localhost/products/insert.php/products)
        .subscribe((response: any) => {
          this.products = response;
        });
    }

    insertProduct(product) {
      this.products.push(product);
    }
  }
}
```

We also need to update the HTML template for the **ProductTableComponent** component to display the table of products:

```
<h2>Product Table</h2>

<table>

  <thead>

    <tr>

      <th>Name</th>

      <th>Description</th>

      <th>Price</th>

    </tr>

  </thead>

  <tbody>

    <tr *ngFor="let product of products">

      <td>{{ product.name }}</td>

      <td>{{ product.description }}</td>

      <td>{{ product.price }}</td>
```

</tr>

</tbody>

</table>

<app-insert-product

(insertProduct)="insertProduct(\$event)"></app-insert-product>

In the **ProductTableComponent** HTML template, we also include the **InsertProductComponent** and pass in a function to handle the insertion of new product data.

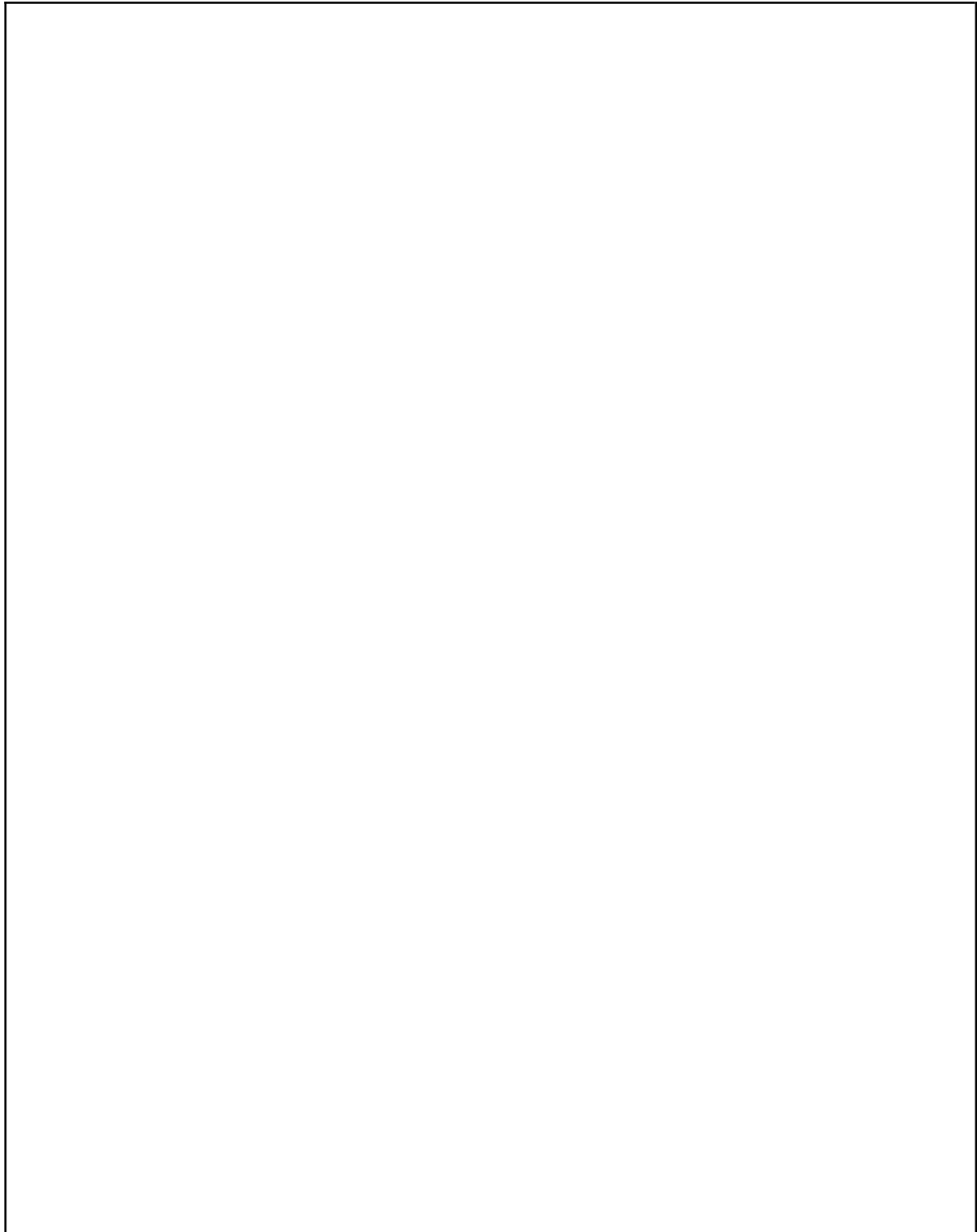
H. Resources/Equipment Required

Sr. No	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

I. Source code:

SOURCE CODE

OUTPUT



J. Practical related Questions.

1. Design web page to insert furniture data in table using web APIs.
2. Design web page to insert employee data in table using web APIs.
3. Design web page to insert classrooms data in table using web APIs.

K. References Links

1. [Angular CRUD using PHP and MySQL – FahmidasClassroom](#)
2. [\(492\) Part-1 Angular 13 CRUD PHP MySql | Angular 13 PHP MySql Insert Update Delete | Angular 13 Tutorial - YouTube](#)
3. [\(492\) Part - 1 \(2022\). Angular PHP MySql CRUD operation | Angular 12 / 13 PHP MySql Insert Update Delete - YouTube](#)
4. [PHP + Angular + MySQL CRUD Example \(knowledgefactory.net\)](#)
5. [Angular 13 CRUD Using PHP and MySQL - Sathish kumar Ramalingam - Medium](#)

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No.17: Design a page to implement Multiview component with login, logout functionalities using different routing options.

A. Objective: Routing is important option of angular and students will implement it with multiview using Login, Home Screen and Logout screens.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.
2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.
3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.
4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.
5. **Engineering practices for society, sustainability and environment:** Apply appropriate technology in context of society, sustainability, environment and ethical practices.
6. **Project Management:** Use engineering management principles individually, as a team member or a leader to manage projects and effectively communicate about well-defined engineering activities.
7. **Life-long learning:** Ability to analyze individual needs and engage in updating in the context of technological changes in field of engineering.

C. Expected Skills to be developed based on competency:

1. Design a page to implement Multiview components for login, home screen and logout functionality using routing option.

D. Expected Course Outcomes(Cos)

Develop single page dynamic applications using Angular framework and APIs.

E. Practical Outcome(Pro)

Design a page to implement Multiview component with login, logout functionalities using different routing options.

F. Expected Affective domain Outcome(ADos)

1. Maintain tools and equipment.
2. Follow Coding standards and practices.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory:

To implement the Multiview component with login and logout functionalities using different routing options in Angular, we need to create the following components:

- HomeComponent
- LoginComponent
- DashboardComponent

The HomeComponent will contain a basic welcome message and a login button, which will redirect the user to the LoginComponent. The LoginComponent will have a form to enter the username and password, and once the user clicks on the login button, the user will be redirected to the DashboardComponent.

The DashboardComponent will contain the main content of the application and will be accessible only after the user logs in. Once the user logs out, the user will be redirected back to the HomeComponent.

Here are the steps to implement this in Angular:

Step 1: Create a new Angular project

```
ng new multiview-component-demo
```

Step 2: Create the necessary components using the Angular CLI

```
ng generate component home
```

```
ng generate component login
```

```
ng generate component dashboard
```

Step 3: Add the necessary routes to the app-routing.module.ts file

```
import { NgModule } from '@angular/core';  
import { Routes, RouterModule } from '@angular/router';  
import { HomeComponent } from './home/home.component';  
import { LoginComponent } from './login/login.component';  
import { DashboardComponent } from './dashboard/dashboard.component';  
  
const routes: Routes = [  
  { path: '', component: HomeComponent },  
  { path: 'login', component: LoginComponent },  
  { path: 'dashboard', component: DashboardComponent },
```

```
];
```

```
@NgModule({  
  imports: [RouterModule.forRoot(routes)],  
  exports: [RouterModule]  
})  
  
export class AppRoutingModule { }
```

Step 4: Update the HomeComponent template to include a login button.

```
<h1>Welcome to the Multiview Component Demo</h1>  
  
<button routerLink="/login">Login</button>
```

Step 5: Update the LoginComponent template to include a login form

```
<h1>Login</h1>  
  
<form>  
  <label>  
    Username:  
    <input type="text">  
  </label>  
  
  <br>  
  
  <label>  
    Password:  
    <input type="password">  
  </label>  
  
  <br>  
  
  <button routerLink="/dashboard">Login</button>  
  
</form>
```

Step 6: Update the DashboardComponent template to include a logout button

```
<h1>Welcome to the Dashboard</h1>  
  
<button routerLink="/">Logout</button>
```

Step 7: Update the AppComponent template to include the router outlet

```
<router-outlet></router-outlet>
```

Step 8: Run the application using the following command

```
ng serve
```

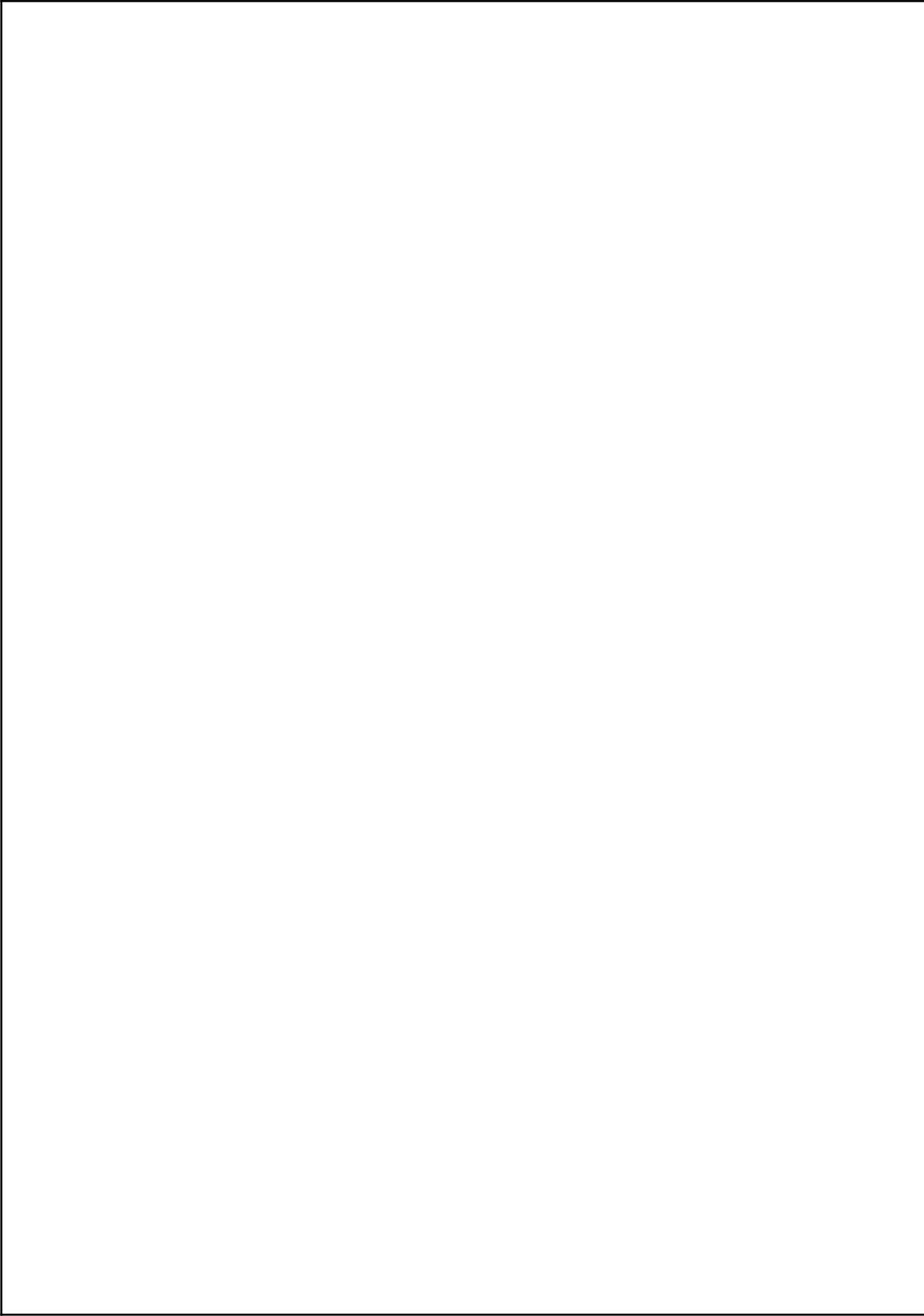
Now you should be able to see the HomeComponent with a login button. Clicking on the login button should redirect you to the LoginComponent with a login form. Once you enter the username and password and click on the login button, you should be redirected to the DashboardComponent. Clicking on the logout button in the DashboardComponent should redirect you back to the HomeComponent.

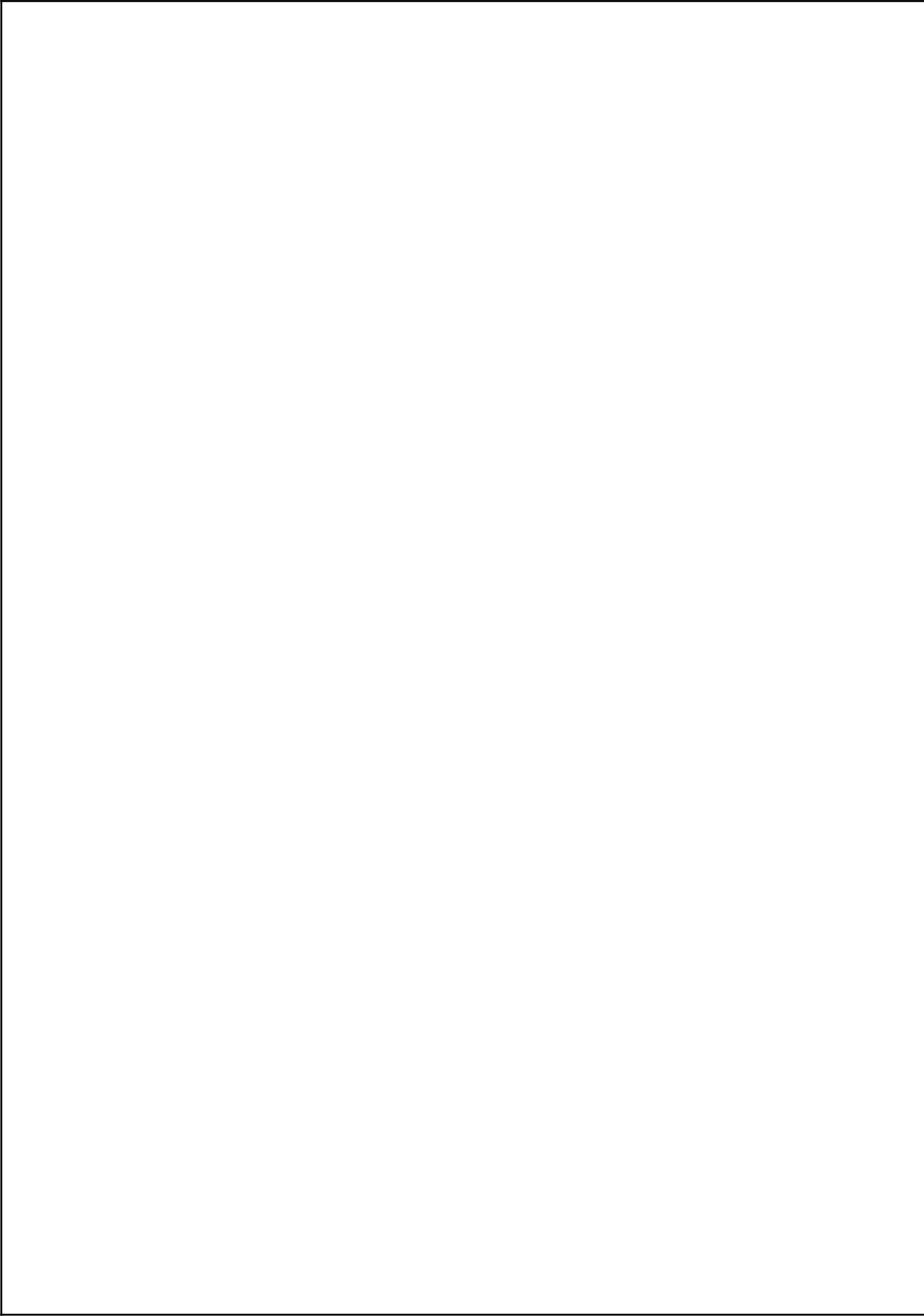
H. Resources/Equipment Required

S r. N o.	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

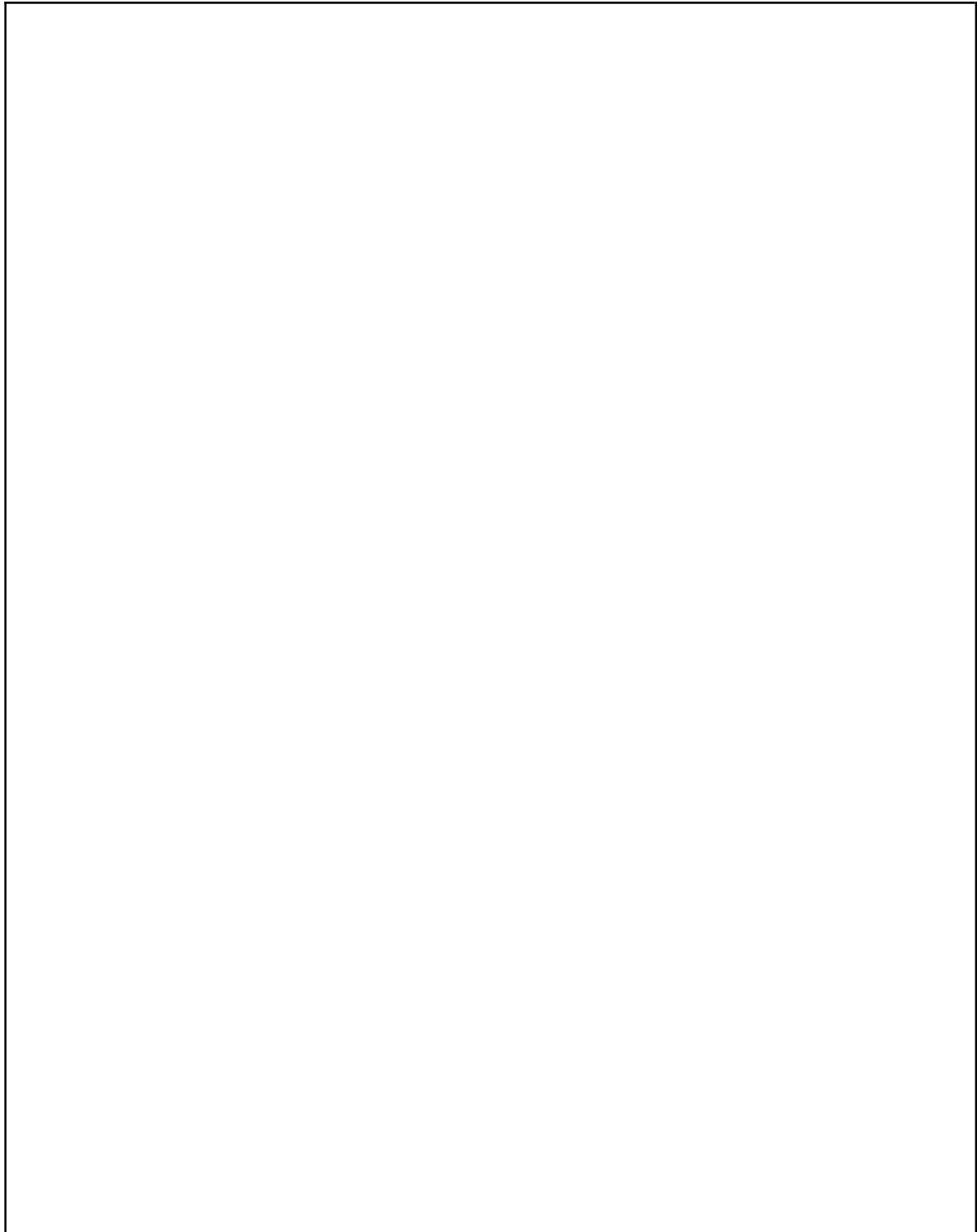
I. Source Code:

SOURCE CODE





OUTPUT



J. Practical related Questions.

1. Design multiview component for product listing and product detail view using routing concept.
2. Design web for hotel listing and hotel room booking functionality using routing concept.

K. References Links

1. [How To Add Login Authentication to React Applications | DigitalOcean](#)
2. [\(492\) Add Login Logout Functionality - YouTube](#)
3. [The Complete Guide to React User Authentication with Auth0](#)
4. [\(492\) User Login and Logout - Authenticating Your Angular App Part 1 - YouTube](#)
5. [javascript - AngularJS- Login and Authentication in each route and controller - Stack Overflow](#)

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No.18: Develop a page to demonstrate page navigation of product list using routing concepts.

- A. **Objective:** Routing is important option of angular and students will implement it for page navigation of products listing.
- B. **Expected Program Outcomes (POs)**

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.
2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.
3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.
4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.
5. **Engineering practices for society, sustainability and environment:** Apply appropriate technology in context of society, sustainability, environment and ethical practices.
6. **Project Management:** Use engineering management principles individually, as a team member or a leader to manage projects and effectively communicate about well-defined engineering activities.
7. **Life-long learning:** Ability to analyze individual needs and engage in updating in the context of technological changes in field of engineering.

C. Expected Skills to be developed based on competency:

1. Design Effective page navigation.
2. Apply page routing concepts in angular.

D. Expected Course Outcomes (Cos)

Develop single page dynamic applications using Angular framework and APIs.

E. Practical Outcome (PRo)

Develop a page to demonstrate page navigation of product list using routing concepts.

F. Expected Affective domain Outcome (ADos)

1. Maintain tools and equipment.
2. Follow Coding standards and practices.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory: to design a product list with pagination navigation using routing concepts in Angular. First, create a new Angular project by running the following command in your terminal:

ng new product-list-app

Next, create a new component to display the product list. Run the following command in your terminal:

ng generate component product-list

Now, open the `product-list.component.ts` file and add the following code to define the product list:

```
import { Component, OnInit } from '@angular/core';

interface Product {
  id: number;
  name: string;
}

@Component({
  selector: 'app-product-list',
  templateUrl: './product-list.component.html',
  styleUrls: ['./product-list.component.css']
})
export class ProductListComponent implements OnInit {

  products: Product[] = [];

  constructor() { }

  ngOnInit(): void {
    // Populate the product list with some dummy data
    for (let i = 1; i <= 100; i++) {
      this.products.push({
        id: i,
        name: `Product ${i}`
      })
    }
  }
```

```

    });
  }
}
}

```

This code defines an interface for the product object, and initializes an empty array for the product list. In the `ngOnInit` method, we populate the product list with some dummy data.

Next, open the `product-list.component.html` file and add the following code to display the product list:

```

<h2>Product List</h2><ul>
  <li *ngFor="let product of products">{{ product.name }}</li>
</ul>

```

This code simply uses the `*ngFor` directive to loop through the `products` array and display each product's name in a list item.

Now, we'll create a pagination component to navigate between pages of the product list. Run the following command in your terminal:

```
ng generate component pagination
```

Open the `pagination.component.ts` file and add the following code to define the pagination component:

```

import { Component, OnInit, Input } from '@angular/core';

@Component({
  selector: 'app-pagination',
  templateUrl: './pagination.component.html',
  styleUrls: ['./pagination.component.css']
})
export class PaginationComponent implements OnInit {

  @Input() totalItems: number = 0;
  @Input() itemsPerPage: number = 10;

```



```
@Input() currentPage: number = 1;
```

```
@Input() totalPages: number = 0;
```

```
constructor() {}
```

```
ngOnInit(): void {
```

```
    this.setTotalPages();
```

```
}
```

```
setTotalPages() {
```

```
    this.totalPages = Math.ceil(this.totalItems / this.itemsPerPage);
```

```
}
```

```
}
```

This code defines a pagination component that takes four input properties: `totalItems`, `itemsPerPage`, `currentPage`, and `totalPages`. In the `ngOnInit` method, we call the `setTotalPages` method to calculate the total number of pages based on the total number of items and the number of items per page.

Next, open the `pagination.component.html` file and add the following code to display the pagination links:

```
import { Injectable } from '@angular/core';
```

```
import { HttpClient } from '@angular/common/http';
```

```
@Injectable({
```

```
    providedIn: 'root'
```

```
})
```

```
export class ProductService {
```

```
    private apiUrl = 'https://example.com/api/products';
```

```
constructor(private http: HttpClient) { }
```

```
getProducts(page: number, pageSize: number) {  
  const url = `${this.apiUrl}?page=${page}&pageSize=${pageSize}`;  
  return this.http.get<any[]>(url);  
}
```

```
}
```

In this service, we're using HttpClient to make a GET request to the backend API with the current page and page size parameters.

Finally, let's create a template for the ProductListComponent that will display the list of products and paging navigation. Here's an example implementation:

```
<div *ngFor="let product of products">
```

```
  {{ product.name }}
```

```
</div>
```

```
<div>
```

```
  <button *ngIf="page > 1" [routerLink]="['/products']" [queryParams]="{  
page: page - 1, pageSize: pageSize }">Previous</button>
```

```
    <button *ngIf="products?.length === pageSize  
[routerLink]="['/products']" [queryParams]="{ page: page + 1, pageSize:  
pageSize }">Next</button>
```

```
</div>
```

In this template, we're using *ngFor to display each product in the products array. We're also using *ngIf to conditionally display the "Previous" and "Next" buttons based on whether we're on the first page or whether there are more products to display.

To use this component, you can add it to your routing configuration like this:

```
import { NgModule } from '@angular/core';
```

```
import { RouterModule, Routes } from '@angular/router';
```

```

import { ProductListComponent } from
'./product-list/product-list.component';

const routes: Routes = [
  { path: 'products', component: ProductListComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

Now you can navigate to the /products URL in your app and see the product list with paging navigation.

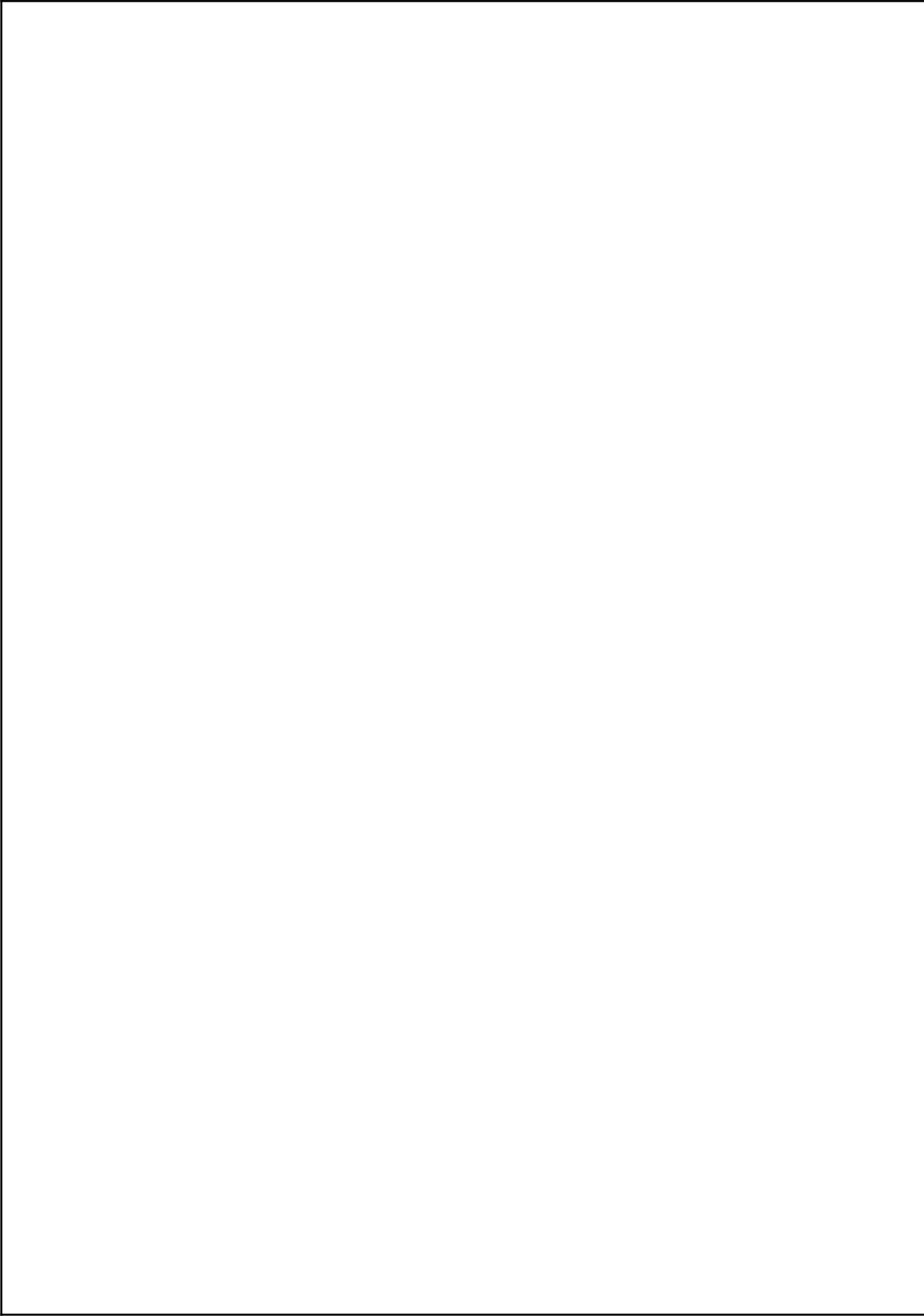
H. Resources/Equipment Required

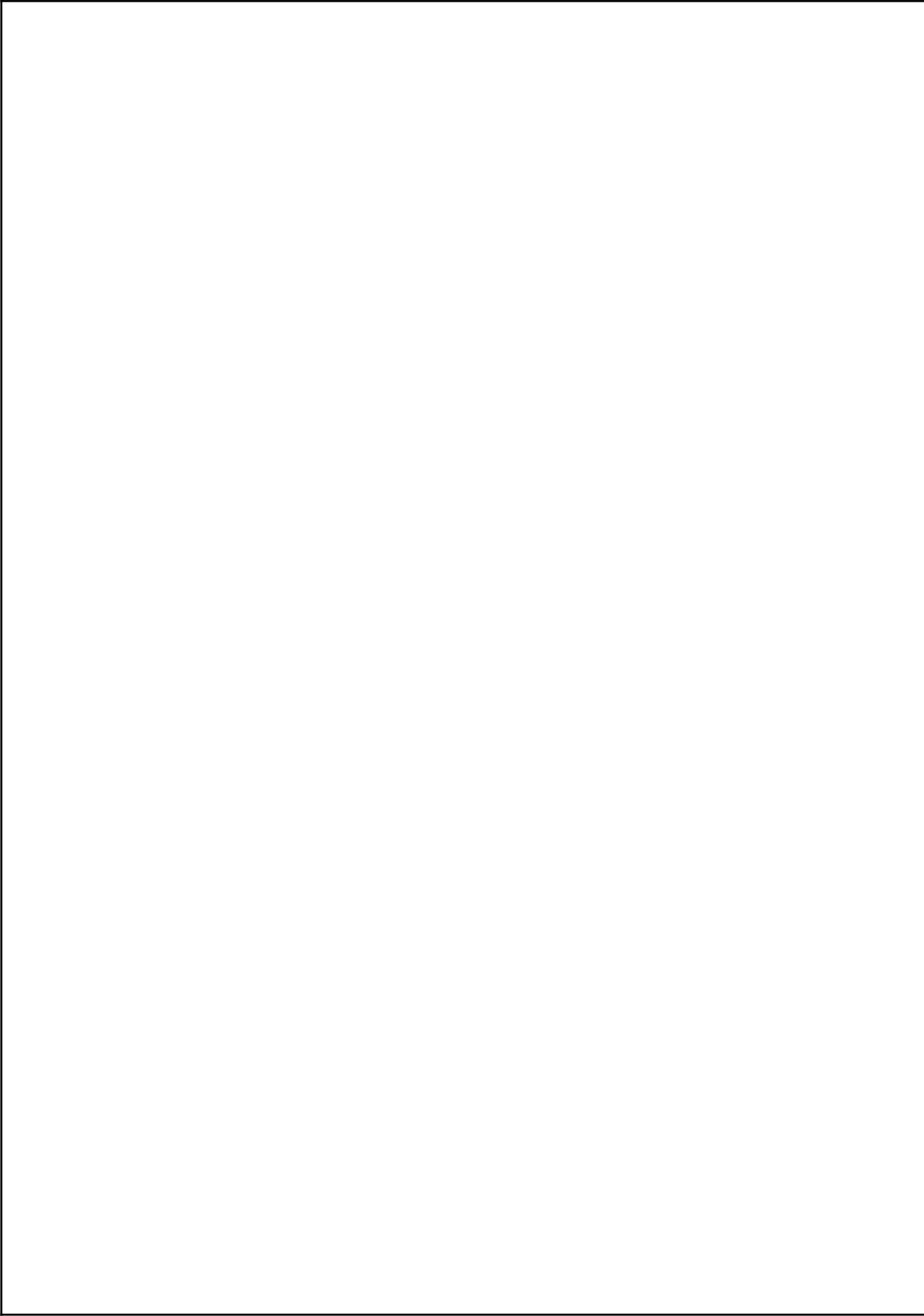
S r. N o.	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

I. Source code:

SOURCE CODE







OUTPUT

J. Practical related Questions.

1. Design web page to implement page navigation for students listing using routing concepts.
2. Design web page to implement page navigation for books listing using routing concepts.
3. Design web page to implement page navigation for subjects listing using routing concepts.

K. References Links

1. [Create a Simple Pagination Component in Angular | JavaScript in Plain English](#)
2. [Angular Routing - javatpoint](#)
3. [Angular Pagination Example - Java Code Geeks - 2023](#)
4. [Angular Tutorial - 23 - Routing and Navigation - YouTube](#)
5. [\(492\) Angular Material Tutorial - 31 - Data table Pagination - YouTube](#)

L. Assessment-Rubrics

S.No .	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No.19: Design a page to load customer and Sales order data using lazy loading technique in angular.

A. Objective: Lazy loading is a technique used in Angular to optimize the loading time of large applications by loading only the necessary components and modules when they are required, rather than loading everything on page loading time.

B. Expected Program Outcomes (POs)

1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.
2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.
3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.
4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.
5. **Engineering practices for society, sustainability and environment:** Apply appropriate technology in context of society, sustainability, environment and ethical practices.
6. **Project Management:** Use engineering management principles individually, as a team member or a leader to manage projects and effectively communicate about well-defined engineering activities.
7. **Life-long learning:** Ability to analyze individual needs and engage in updating in the context of technological changes in field of engineering.

C. Expected Skills to be developed based on competency:

1. Design a page to load data using lazy loading technique in angular.

D. Expected Course Outcomes(Cos)

Develop single page dynamic applications using Angular framework and APIs.

E. Practical Outcome(PRo)

Design a page to load customer and Sales order data using lazy loading technique in angular.

F. Expected Affective domain Outcome(ADos)

1. Maintain tools and equipment.
2. Follow Coding standards and practices.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory:

Lazy loading is a technique used in Angular to optimize the loading time of large applications by loading only the necessary components and modules when they are required, rather than loading everything upfront.

In a typical Angular application, all the components and modules are loaded when the application is first launched. This can slow down the initial loading time of the application, especially if the application is large and contains many components and modules. With lazy loading, the application can be divided into smaller feature modules, each of which is loaded only when the user navigates to the corresponding route.

Here's an example of how to implement lazy loading in Angular:

1. Create a feature module that contains the components and services required for a specific feature. For example, a module for managing user profiles.

ng generate module user-profile --route user-profile --module app.module

2. Define a route in the user-profile.module.ts file that loads the components and services required for the user profile feature.

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { UserProfileComponent } from './user-profile.component';
```

```
const routes: Routes = [
  { path: '', component: UserProfileComponent }
];
```

```
@NgModule({
  imports: [RouterModule.forChild(routes)],
  declarations: [UserProfileComponent]
})
export class UserProfileModule { }
```

3. In the app-routing.module.ts file, define the lazy-loaded route for the user profile feature.

```
import { NgModule } from '@angular/core';
```

```

import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'home' },
  { path: 'home', loadChildren: () =>
import('./home/home.module').then(m =>m.HomeModule) },
  { path: 'user-profile', loadChildren: () =>
import('./user-profile/user-profile.module').then(m
=>m.UserProfileModule) }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})

export class AppRoutingModule { }

```

4. In this example, the UserProfileModule is lazy-loaded when the user navigates to the /user-profile route. The loadChildren property in the route definition is used to specify the module that should be loaded lazily.
5. With lazy loading, the user profile feature is not loaded when the application first launches, but is loaded only when the user navigates to the /user-profile route. This can significantly improve the loading time of the application and provide a better user experience.

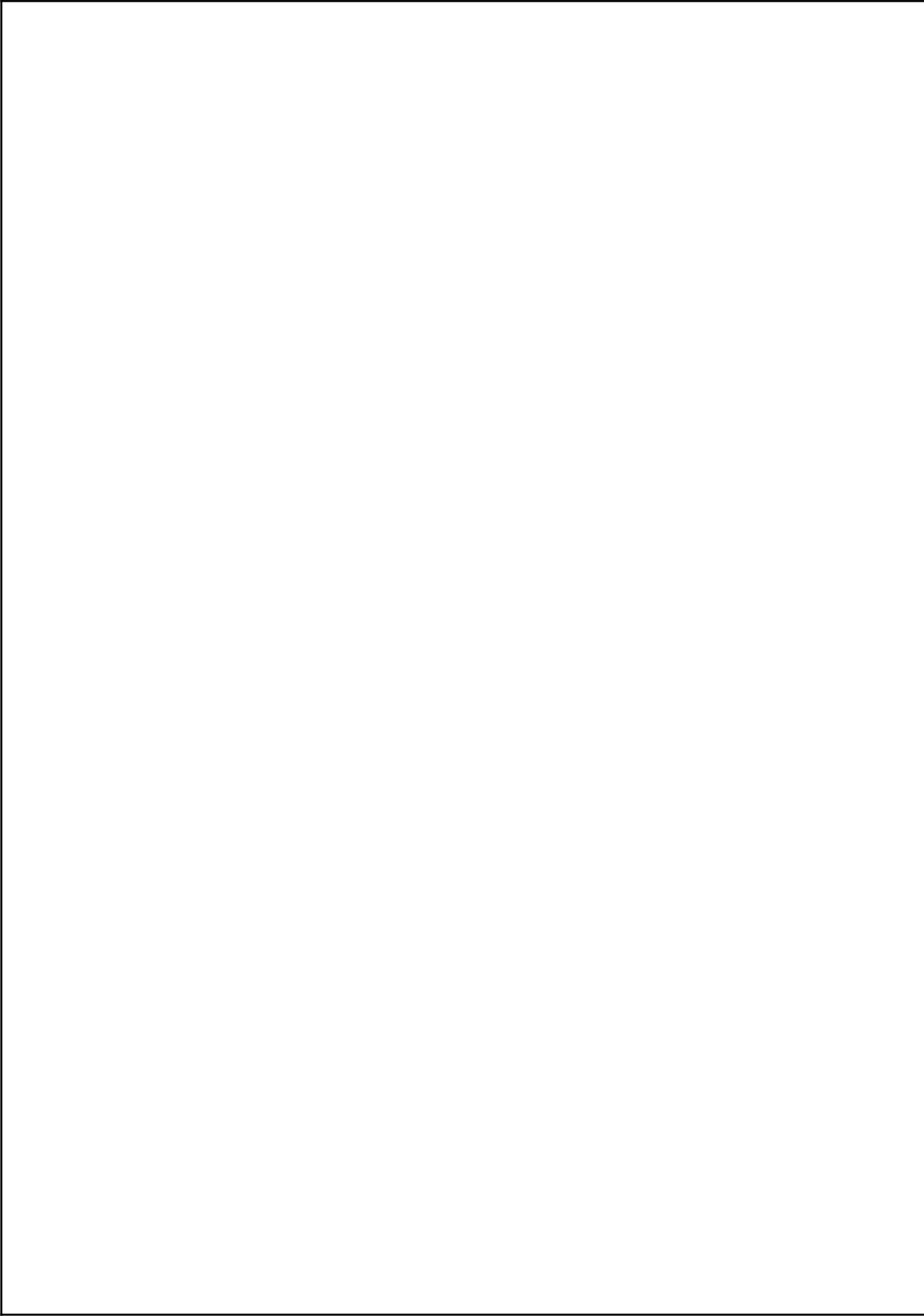
H. Resources/Equipment Required

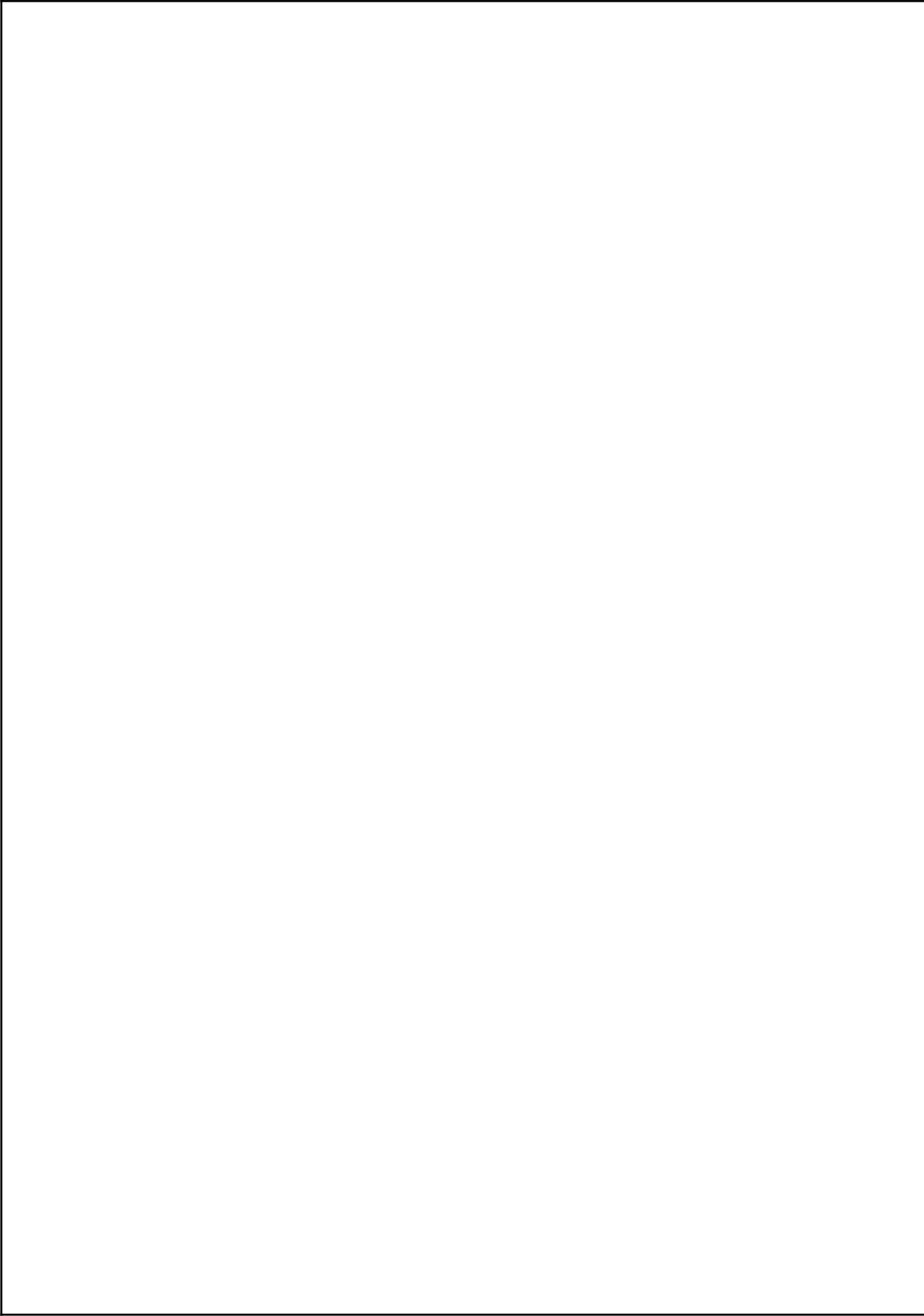
Sr. No	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open-source software from Microsoft

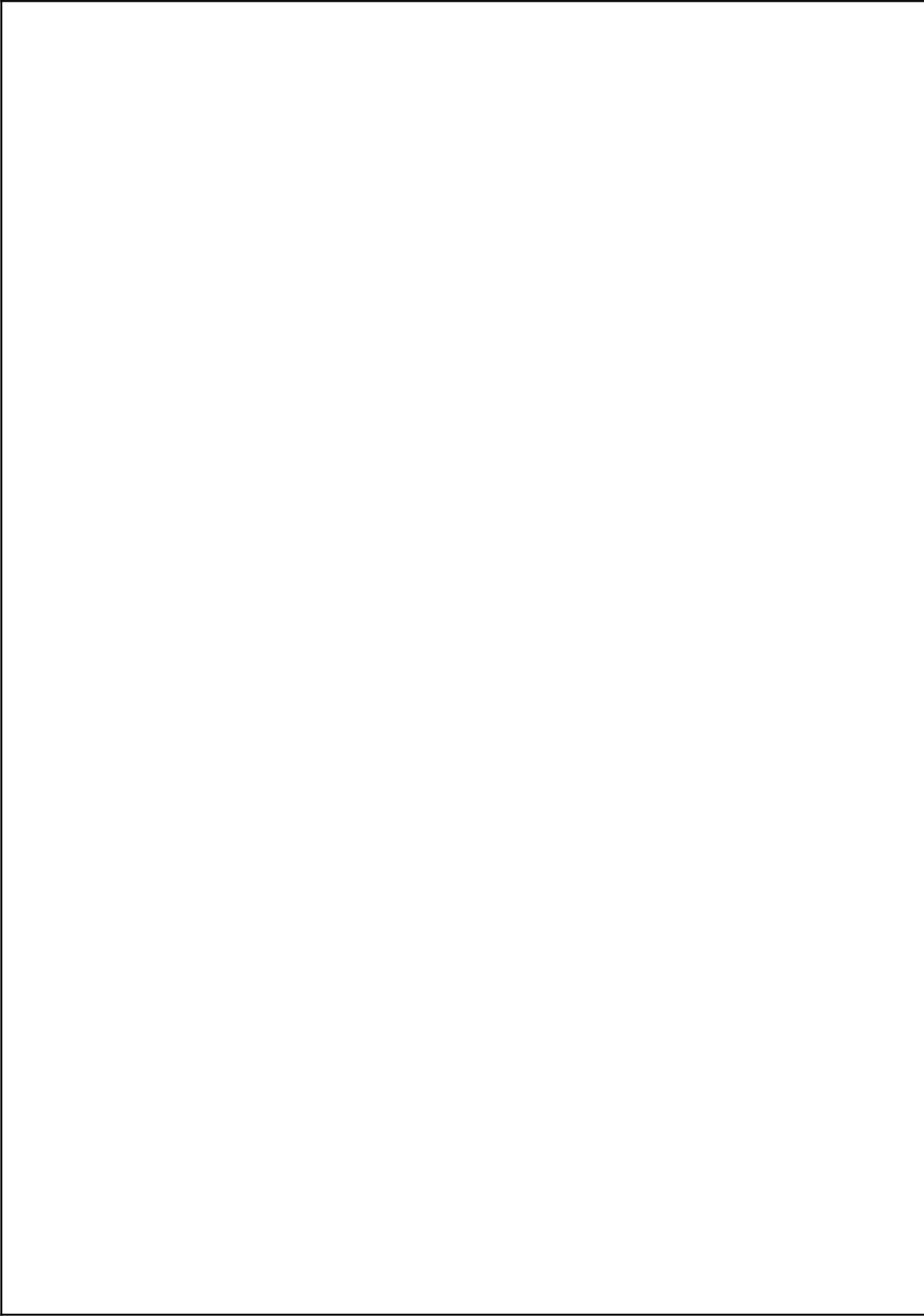
3	Node JS and NPM Package Manager	Open-source software
4	Browser	Microsoft Edge, Google Chrome etc

I. Source Code:

SOURCE CODE







OUTPUT

J. Practical related Additional Exercise.

1. Use the Angular Router to implement lazy loading for child routes within a feature module. Create a nested route structure that loads child modules lazily.
2. Use the Angular CLI to generate a new feature module and component. Implement lazy loading for the feature module and use the component to display data on the page.
3. Create an Angular application that includes multiple feature modules. Implement lazy loading for each feature module so that only the necessary components and services are loaded when the user navigates to a specific route.

K. References Link

1. [Angular - Lazy-loading feature modules](#)
2. [Lazy Loading in Angular – A Beginner's Guide to NgModules \(freecodecamp.org\)](#)
3. [Angular Lazy-loading - javatpoint](#)
4. [How To Implement Lazy Loading in Angular | by LakinduHewawasam | Enlear Academy](#)
5. [How To Use Lazy Loading Routes in Angular | DigitalOcean](#)
6. [Lazy loading in angular - YouTube](#)
7. [\(489\) Lazy Loading Components in Angular 4 - YouTube](#)

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of input and output displayed (messaging and formatting).	10
Total		100

Signature with Date

Practical No.20: Design a page to implement CORS concept.

- A. Objective:** CORS is an HTTP header- based mechanism that allows the server to indicate any other domain, scheme, or port origins and enables them to be loaded.
- B. Expected Program Outcomes (POs)**
1. **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the *engineering* problems.
 2. **Problem analysis:** Identify and analyse well-defined *engineering* problems using codified standard methods.
 3. **Design/ development of solutions:** Design solutions for *engineering* well-defined technical problems and assist with the design of systems components or processes to meet specified needs.
 4. **Engineering Tools, Experimentation and Testing:** Apply modern *engineering* tools and appropriate technique to conduct standard tests and measurements.
 5. **Engineering practices for society, sustainability and environment:** Apply appropriate technology in context of society, sustainability, environment and ethical practices.
 6. **Project Management:** Use engineering management principles individually, as a team member or a leader to manage projects and effectively communicate about well-defined engineering activities.
 7. **Life-long learning:** Ability to analyze individual needs and engage in updating in the context of technological changes in field of engineering.
- C. Expected Skills to be developed based on competency:**
1. Develop service for consuming web APIs.
 2. Use concept of CORS for multiple domain accesses.
- D. Expected Course Outcomes(Cos)**
- Develop single page dynamic applications using Angular framework and APIs.
- E. Practical Outcome(PRo)**
- Design a page to implement CORS concept.
- F. Expected Affective domain Outcome(ADos)**
1. Maintain tools and equipment.

2. Follow Coding standards and practices.
3. Follow safety practices.
4. Follow ethical practices

G. Prerequisite Theory:

CORS stands for Cross-Origin Resource Sharing. It is a security feature implemented in modern web browsers that restricts web pages from making requests to a different domain or origin than the one that served the web page.

In the context of Angular, CORS is an important concept to understand because when Angular applications communicate with APIs hosted on a different domain, CORS needs to be enabled on the API server-side to allow the Angular application to make requests to the API.

Angular applications typically use the **HttpClient** module to make HTTP requests to APIs. When a request is made to a different domain or origin, the browser will automatically send a preflight OPTIONS request to the API to check if the requested domain or origin is allowed to make requests to the API. If CORS is not enabled on the API server-side to allow requests from the Angular application, the API server will respond with an error and the request will fail.

To enable CORS on the API server-side, the API must respond with specific headers that allow the requesting domain or origin to access the API. These headers include **Access-Control-Allow-Origin**, **Access-Control-Allow-Methods**, and **Access-Control-Allow-Headers**, among others.

In summary, CORS is an important concept to understand when building Angular applications that communicate with APIs hosted on a different domain or origin. It is necessary to ensure that CORS is enabled on the API server-side to allow the Angular application to make requests to the API. Below example of how to implement CORS in an Angular application:

1. First, create an Angular service to handle the API request with CORS headers:

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class ApiService {

  constructor(private http: HttpClient) { }
```

```
getApiData() {  
  const headers = new HttpHeaders({  
    'Access-Control-Allow-Origin': '*',  
    'Access-Control-Allow-Methods': 'GET, POST, PUT, DELETE',  
    'Access-Control-Allow-Headers': 'Origin, Content-Type, X-Auth-Token'  
  });  
  
  return this.http.get('https://your-api-endpoint.com/data', { headers });  
}  
}
```

2. Next, create an Angular component that calls the API service and displays the data on the web page:

```
import { Component } from '@angular/core';  
import { ApiService } from '../api.service';  
  
@Component({  
  selector: 'app-cors-example',  
  templateUrl: './cors-example.component.html',  
  styleUrls: ['./cors-example.component.css']  
})  
export class CorsExampleComponent {  
  
  apiData = [];  
  
  constructor(private apiService: ApiService) {}  
  
  fetchData() {  
    this.apiService.getApiData().subscribe((data: any) => {
```

```

    this.apiData = data;
  });
}
}

```

3. Finally, create the HTML template for the CorsExampleComponent component:

```

<h2>CORS Example</h2>

<button (click)="fetchData()">Fetch Data</button>

<div *ngIf="apiData.length > 0">

  <ul>

    <li *ngFor="let data of apiData">

      {{ data }}

    </li>

  </ul>

</div>

```

In this HTML template, there is a button that calls the `fetchData()` function when clicked. This function calls the `getApiData()` function in the `ApiService` service to fetch data from the API. The fetched data is stored in the `apiData` variable and displayed on the web page using an `*ngFor` loop.

Note: CORS must also be enabled on the API server-side. These headers alone will not enable CORS if it is not already enabled on the server. Also, note that it is not recommended to use the `Access-Control-Allow-Origin` header with a wildcard (*) as it can be a security risk. It is better to specify specific origins that are allowed to access the API.

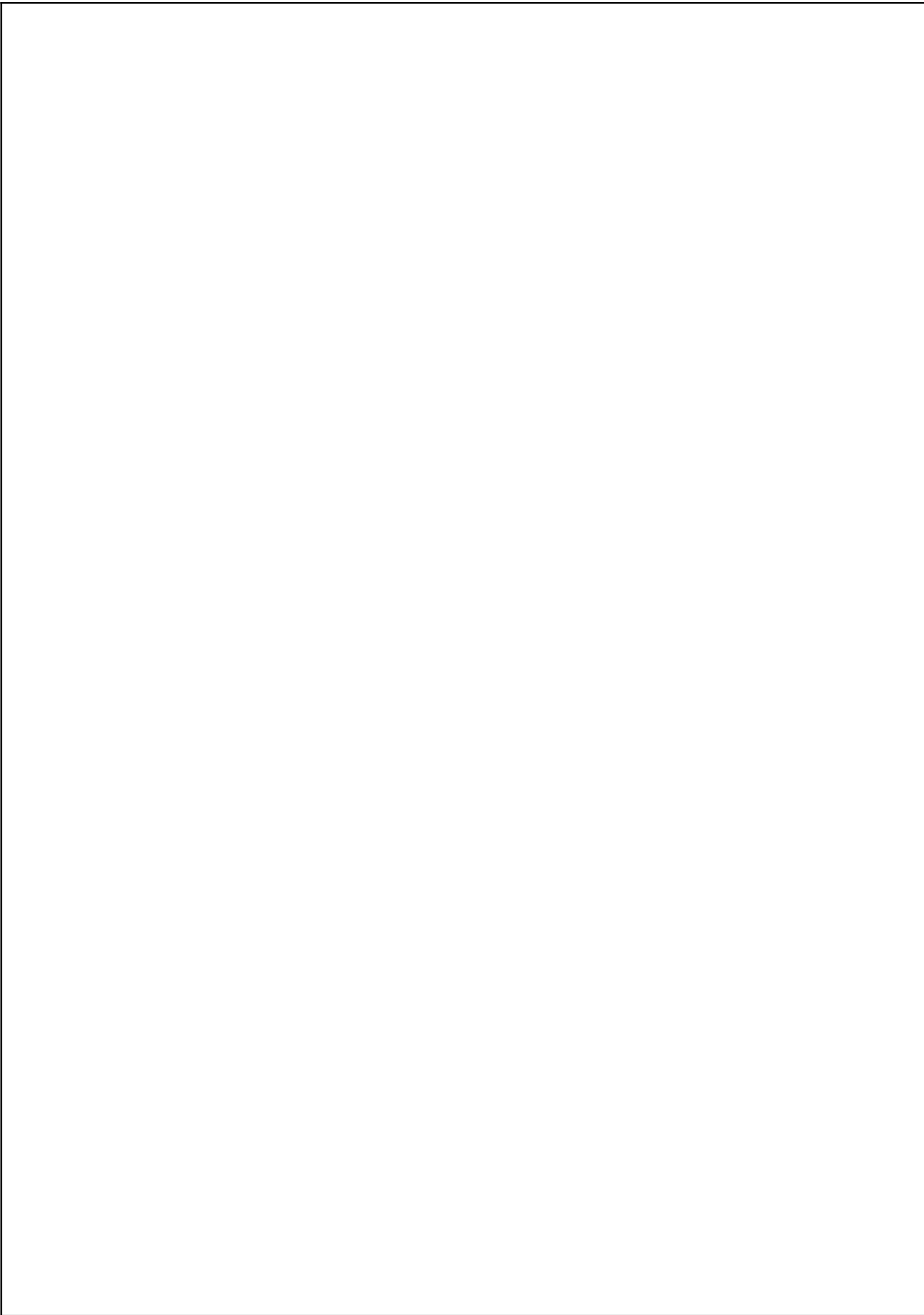
H. Resources/Equipment Required

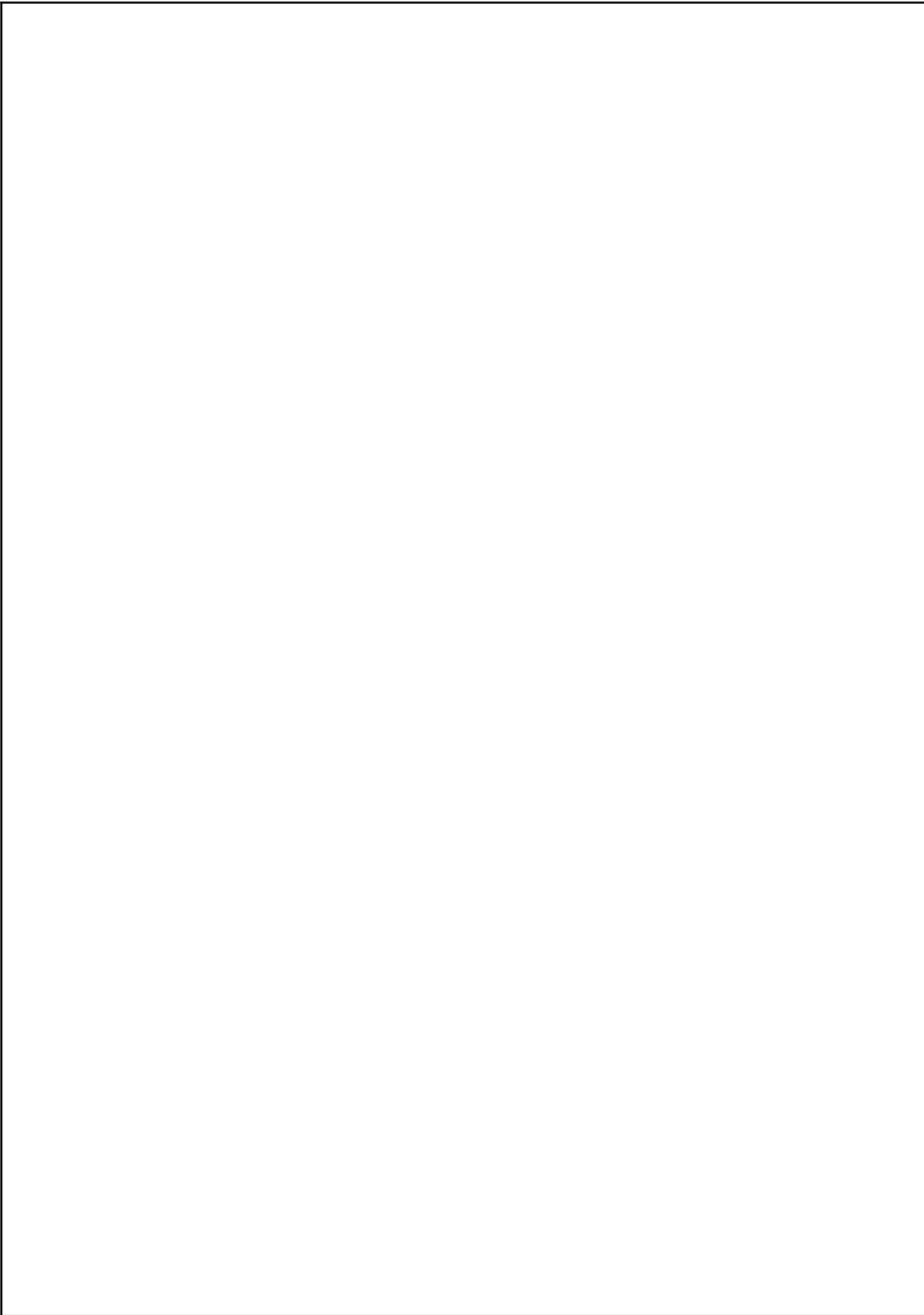
Sr. No	Equipment/ Software Resources	Specification
1	Computer System	Intel I3 processor with minimum 4 GB RAM, 40GB HDD, Windows 7 or above Operating system.
2	Visual Code	Open source software from Microsoft

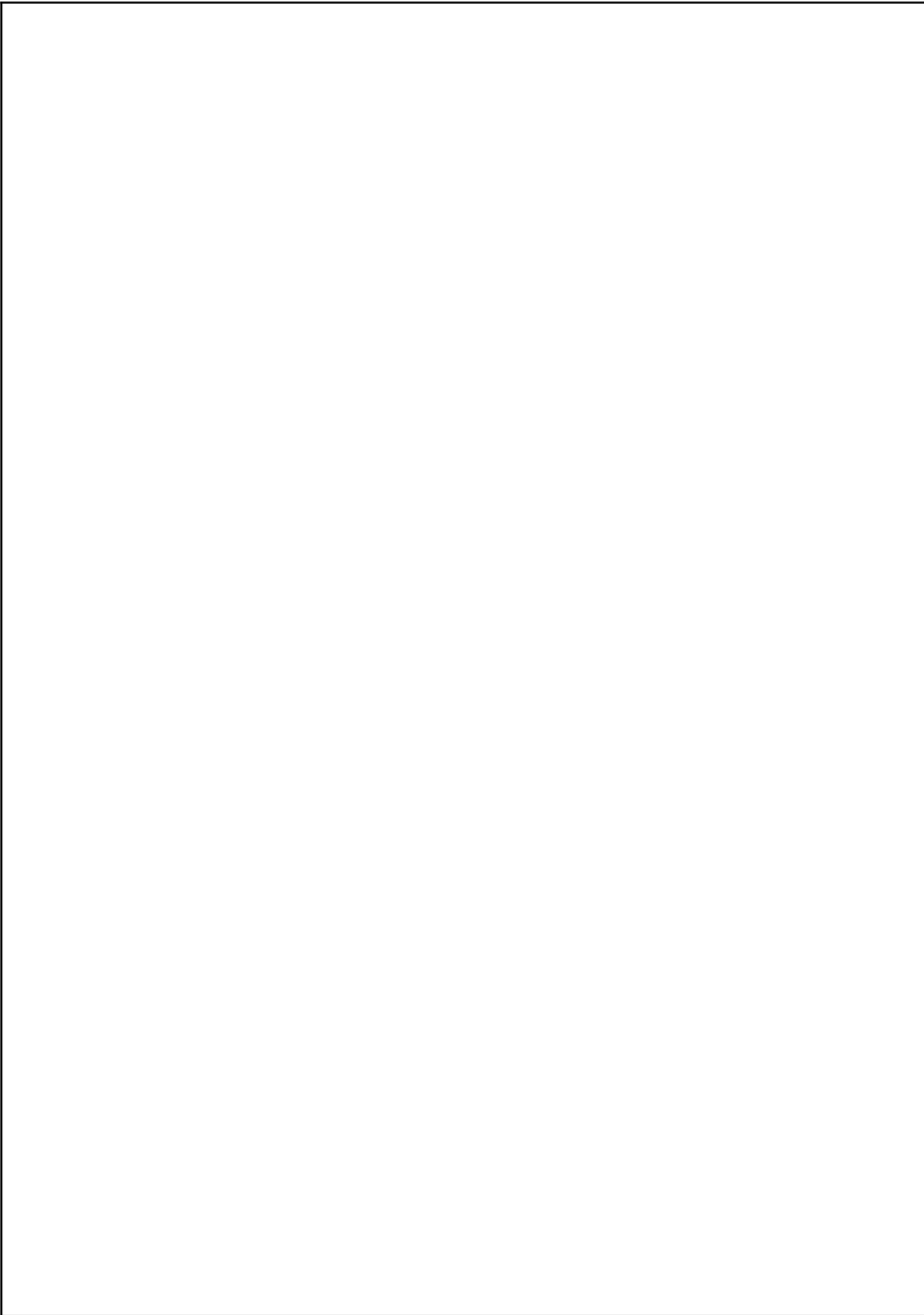
3	Node JS and NPM Package Manager	Open source software
4	Browser	Microsoft Edge, Google Chrome etc

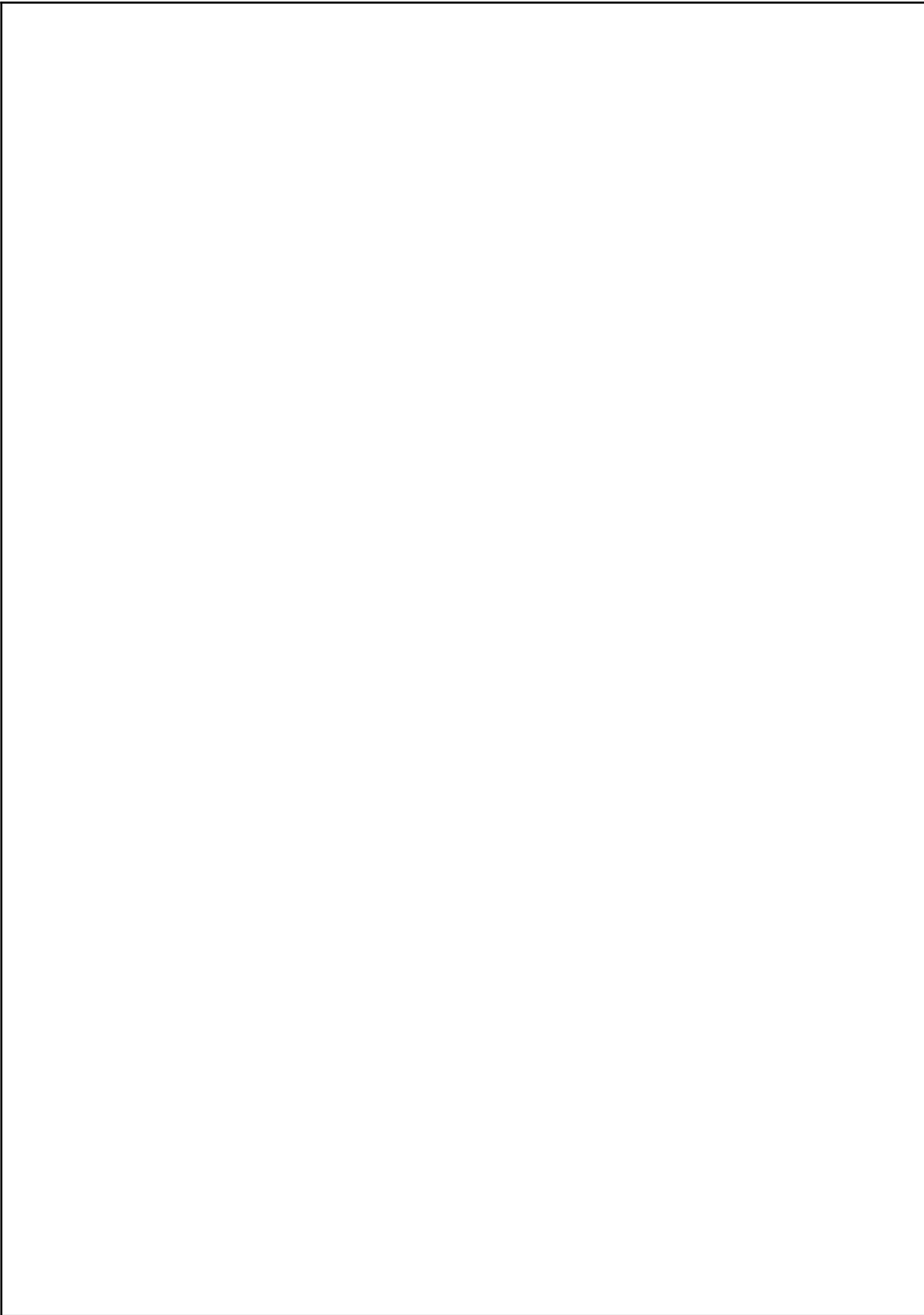
I. Source code:

SOURCE CODE

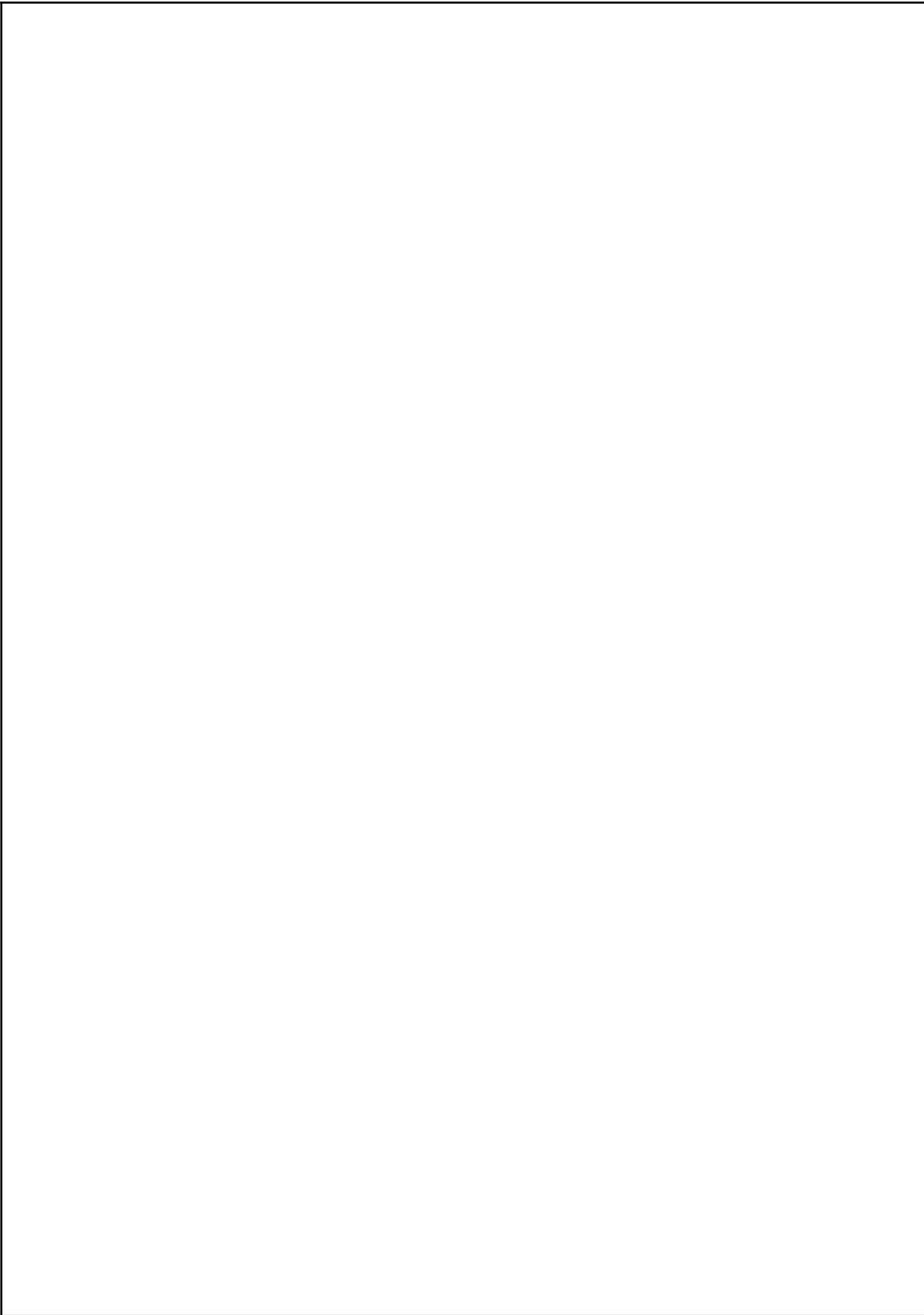








OUTPUT



J. Practical related Questions.

1. Create a simple Angular application that fetches data from an API on a different domain. Implement the necessary CORS headers on the API server-side to allow the Angular application to access the data.
2. Modify the Angular application from exercise 1 to include authentication. Implement CORS headers on the API server-side to allow the Angular application to send authentication tokens with requests.
3. Create an Angular application that sends data to an API on a different domain. Implement the necessary CORS headers on the API server-side to allow the Angular application to send data.

K. References Links

1. [What is CORS in Angular? \(educative.io\)](#)
2. [Do you know how to resolve CORS issues in Angular? | by Mousumi Hazarika | Weekly Webtips | Medium](#)
3. [Angular CORS Guide: Examples and How to Enable It \(stackhawk.com\)](#)
4. [Making Proxy Request In Angular/Fix CORS Issue In Angular Applications \(c-sharpcorner.com\)](#)

L. Assessment-Rubrics

S.No.	Sample Performance Indicators for the PrOs	Weightage in %
1	Use of creative and innovative approach.	20
2	Readability	15
3	Code Efficiency	30
4	Verify practical implementation for desired output.	25
5	Readability and documentation of the program/Quality of input and output displayed (messaging and formatting).	10
Total		100

Signature with Date