User Registration Form in Angular with Node.js API Integration

Objective

In this hands-on lab, you will build a fully functional user registration form using Angular and connect it to an existing Node.js REST API for data handling. You will learn how to implement Angular reactive form controls with validation, send HTTP POST requests using Angular's `HttpClient`*, and ensure your backend API correctly receives and stores the submitted user data.*

Instructions:

Open the Integrated Terminal

1. Launch VS Code

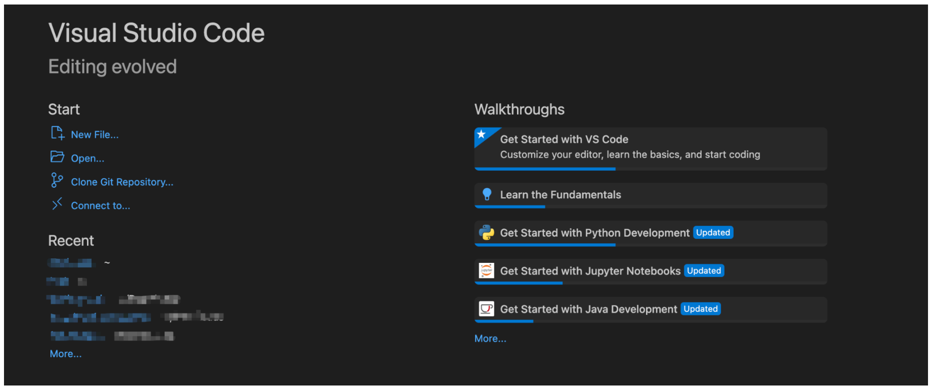

Visual Studio Code
Editing evolved
Start
New File...
Open...
Clone Git Repository...
Connect to...
Recent
More...
Walkthroughs
Get Started with VS Code
Customize your editor, learn the basics, and start coding
Learn the Fundamentals
Get Started with Python Development Updated
Get Started with Jupyter Notebooks Updated
Get Started with Java Development Updated
More...

***Figure 1**: VS Code Default Window*

2. Open your project folder or any

   a. Click on “Open”.

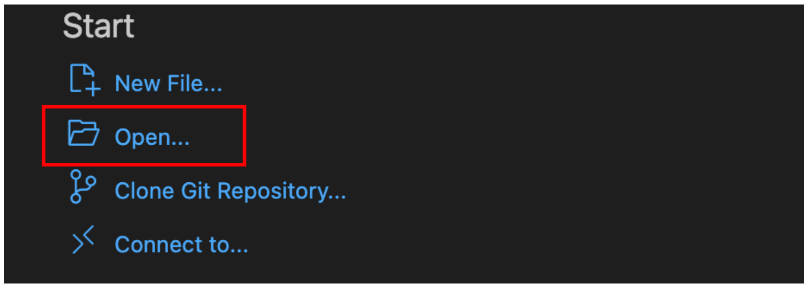


***Figure 2**: Opening the project directory in VS Code*

b. Select the folder and click on “Open” or create a new folder.

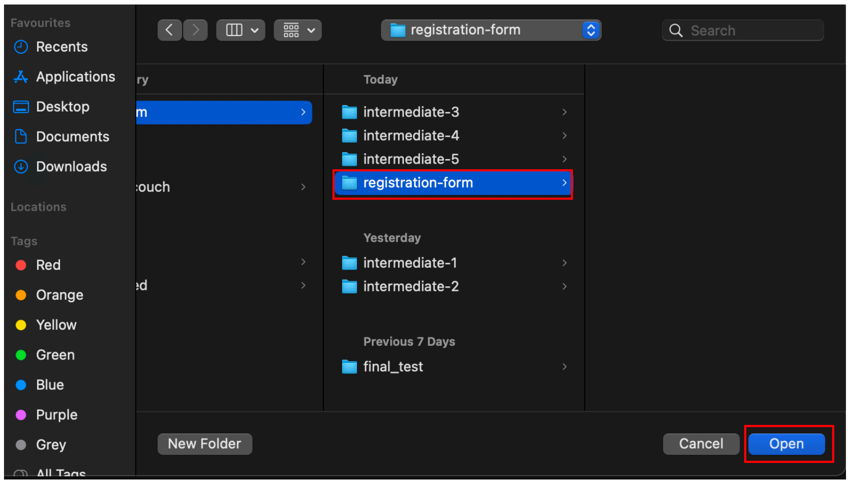
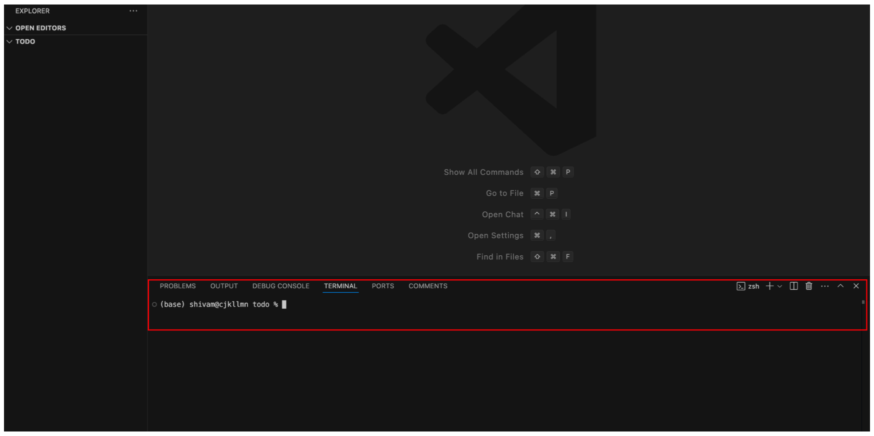


***Figure 3**: Selecting the project directory*

3. Open the terminal with Ctrl+` (backtick)



```
(base) shivam@cjkllmn todo %
```

***Figure 4**: VS Code’s Terminal*

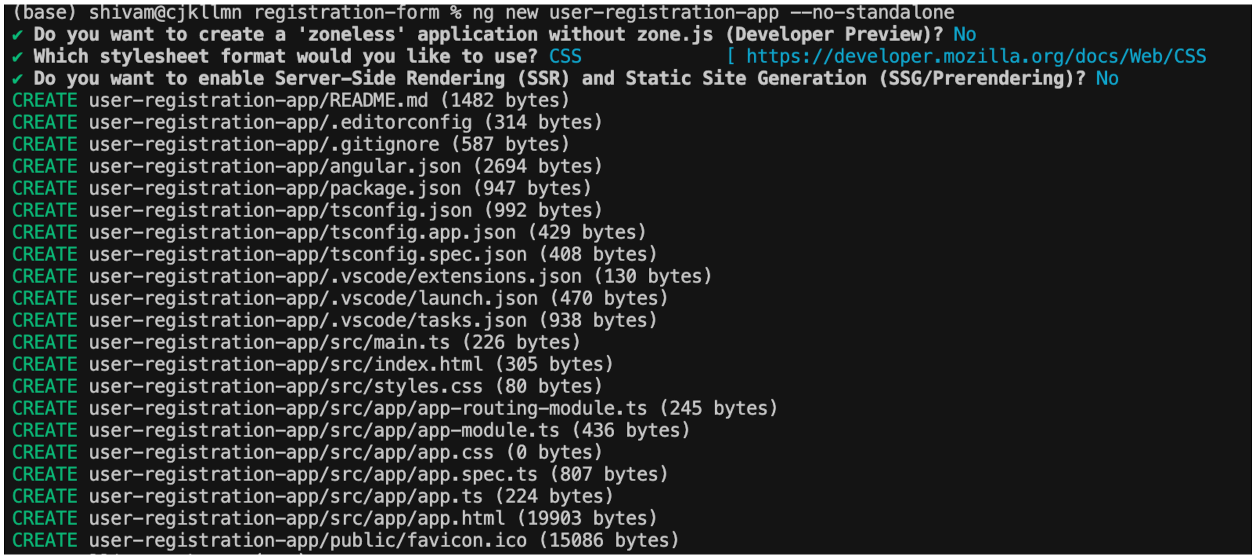
1. Setup the Angular Project

Start by creating a new Angular project or use an existing one where you want to add the registration form.

1. Open your terminal and create a new Angular project (if needed):

```
ng new user-registration-app --no-standalone
```

Continue pressing “Enter” to continue and select default settings till the installation starts.



```
(base) shivam@cjkllmn registration-form % ng new user-registration-app --no-standalone
✔ Do you want to create a 'zoneless' application without zone.js (Developer Preview)? No
✔ Which stylesheet format would you like to use? CSS             [ https://developer.mozilla.org/docs/Web/CSS
✔ Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? No
CREATE user-registration-app/README.md (1482 bytes)
CREATE user-registration-app/.editorconfig (314 bytes)
CREATE user-registration-app/.gitignore (587 bytes)
CREATE user-registration-app/angular.json (2694 bytes)
CREATE user-registration-app/package.json (947 bytes)
CREATE user-registration-app/tsconfig.json (992 bytes)
CREATE user-registration-app/tsconfig.app.json (429 bytes)
CREATE user-registration-app/tsconfig.spec.json (408 bytes)
CREATE user-registration-app/.vscode/extensions.json (130 bytes)
CREATE user-registration-app/.vscode/launch.json (470 bytes)
CREATE user-registration-app/.vscode/tasks.json (938 bytes)
CREATE user-registration-app/src/main.ts (226 bytes)
CREATE user-registration-app/src/index.html (305 bytes)
CREATE user-registration-app/src/styles.css (80 bytes)
CREATE user-registration-app/src/app/app-routing-module.ts (245 bytes)
CREATE user-registration-app/src/app/app-module.ts (436 bytes)
CREATE user-registration-app/src/app/app.css (0 bytes)
CREATE user-registration-app/src/app/app.spec.ts (807 bytes)
CREATE user-registration-app/src/app/app.ts (224 bytes)
CREATE user-registration-app/src/app/app.html (19903 bytes)
CREATE user-registration-app/public/favicon.ico (15086 bytes)
```

Figure 5: *New Angular project created*

2. Navigate into your project folder:

```
cd user-registration-app
```

3. Install required Angular modules by importing them into your `AppModule`:
   - `ReactiveFormsModule` - for reactive forms
   - `HttpClientModule` - for making HTTP requests
4. Open `src/app/app.ts` and edit as follows:

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

import { Routes } from '@angular/router';
import { RegisterComponent } from './register/register';

export const routes: Routes = [
{ path: '', component: RegisterComponent },
];

@Component({
 selector: 'app-root',
 standalone: true,
 imports: [RouterOutlet],
 template: `<router-outlet></router-outlet>`,
 styleUrls: ['./app.css']
})
```

```
export class AppComponent {}
```

2. Create the Registration Form Component

Generate a new Angular component named `register` to hold the registration form.

1. Create the component using Angular CLI:

```
ng generate component register --standalone
```

2. In the component TypeScript file `src/app/register/register.ts`, set up the reactive form:

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from
'@angular/forms';
import { HttpClient, HttpClientModule } from '@angular/common/http';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-register',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModule, HttpClientModule],
  templateUrl: './register.html',
  styleUrls: ['./register.css']
})
export class RegisterComponent implements OnInit {
  registerForm!: FormGroup;
  submitted = false;
  successMessage = '';
  errorMessage = '';

  constructor(private fb: FormBuilder, private http: HttpClient) {}

  ngOnInit(): void {
    this.registerForm = this.fb.group({
      name: ['', Validators.required],
      email: ['', [Validators.required, Validators.email]],
      password: ['', [Validators.required, Validators.minLength(6)]]
    });
  }
  get f() {
    return this.registerForm.controls;
  }

  submitForm(): void {
    this.submitted = true;
    this.successMessage = '';
    this.errorMessage = '';

    if (this.registerForm.invalid) return;

    const user = {
      name: this.f['name'].value,
      email: this.f['email'].value,
      password: this.f['password'].value
    };

    this.http.post('http://localhost:3000/users', user).subscribe({
      next: () => {
        this.successMessage = 'Registration successful!';
        this.registerForm.reset();
        this.submitted = false;
      },
      error: (err) => {
        this.errorMessage = err?.error?.message || 'Registration failed.
Please try again.';
      }
    });
  }
}
```

3. Design the Form UI

Build a user-friendly form in the component HTML file `src/app/register/register.html` leveraging Angular form directives and display validation messages.

```
<form [formGroup]="registerForm" (ngSubmit)="submitForm()" novalidate>
    <div>
      <label>Name:</label>
      <input type="text" formControlName="name" />
      <div *ngIf="submitted && f['name'].errors">
        <small *ngIf="f['name'].errors['required']">Name is
```

```
required.</small>
      </div>
    </div>

    <div>
      <label>Email:</label>
      <input type="email" formControlName="email" />
      <div *ngIf="submitted && f['email'].errors">
        <small *ngIf="f['email'].errors['required']">Email is
required.</small>
        <small *ngIf="f['email'].errors['email']">Enter a valid
email.</small>
      </div>
    </div>

    <div>
      <label>Password:</label>
      <input type="password" formControlName="password" />
      <div *ngIf="submitted && f['password'].errors">
        <small *ngIf="f['password'].errors['required']">Password is
required.</small>
        <small *ngIf="f['password'].errors['minlength']">Minimum 6
characters.</small>
      </div>
    </div>

    <button type="submit">Register</button>

    <p class="success" *ngIf="successMessage">{{ successMessage }}</p>
    <p class="error" *ngIf="errorMessage">{{ errorMessage }}</p>
  </form>
```

register.css

src/app/register/register.css

```
/* Container for the form */
:host {
   display: flex;
   justify-content: center;
   align-items: center;
   padding: 2rem;
   background-color: #f9f9f9;
   min-height: 100vh;
```

```css
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}
 /* Form styling */
form {
  background: #fff;
  padding: 2rem 3rem;
  border-radius: 8px;
  box-shadow: 0 8px 16px rgba(0, 0, 0, 0.1);
  width: 100%;
  max-width: 400px;
  box-sizing: border-box;
}
 /* Form title */
h2 {
  margin-bottom: 1.5rem;
  color: #333;
  text-align: center;
}
 /* Input field container */
.form-group {
  margin-bottom: 1.25rem;
}
 /* Label styling */
label {
  display: block;
  margin-bottom: 0.5rem;
  font-weight: 600;
  color: #555;
}
 /* Inputs styling */
input[type="text"],
input[type="email"],
input[type="password"] {
  width: 100%;
  padding: 0.5rem 0.75rem;
  font-size: 1rem;
  border: 1.5px solid #ccc;
  border-radius: 4px;
  transition: border-color 0.3s ease;
  box-sizing: border-box;
}
 input[type="text"]:focus,
input[type="email"]:focus,
input[type="password"]:focus {
  border-color: #007bff;
  outline: none;
}
 /* Error state */
input.ng-invalid.ng-touched {
  border-color: #dc3545;
}
 /* Validation error messages */
.error-message {
  color: #dc3545;
  font-size: 0.875rem;
  margin-top: 0.25rem;
}
 /* Success and error messages */
.success-message {
  color: #28a745;
  margin-bottom: 1rem;
  text-align: center;
}
 .error-message-global {
  color: #dc3545;
  margin-bottom: 1rem;
  text-align: center;
}
 /* Submit button */
button[type="submit"] {
  width: 100%;
  padding: 0.6rem 1rem;
  font-size: 1.1rem;
  font-weight: 700;
  color: #fff;
  background-color: #007bff;
  border: none;
  border-radius: 6px;
  cursor: pointer;
  transition: background-color 0.3s ease;
  margin-top: 0.5rem;
}
 button[type="submit"]:hover {
  background-color: #0056b3;
}
 /* Disabled button */
```

```
button[type="submit"]:disabled {
  background-color: #a0a5aa;
  cursor: not-allowed;
}
```

Handle Form Submission

In the `submitForm()` method inside the component TypeScript file, you implemented:

- Form validity check — the form submits only if inputs are valid.
- Sending a POST request to `http://localhost:3000/users` using Angular's `HttpClient`.
- Displaying success or error messages based on the backend response.

4. Replace main.ts

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideRouter } from '@angular/router';
import { AppComponent } from './app/app';

import { routes } from './app/app';

bootstrapApplication(AppComponent, {
 providers: [provideRouter(routes)]
});
```

Node.js API Setup (Backend)

1. Create a directory named “backend” outside the **user-registration-app** directory

```
cd..
```

```
mkdir backend
cd backend
```

2. Initialize Node Project

```
npm init -y
```

3. Install required dependencies

```
npm install express cors
```

4. Create server.js

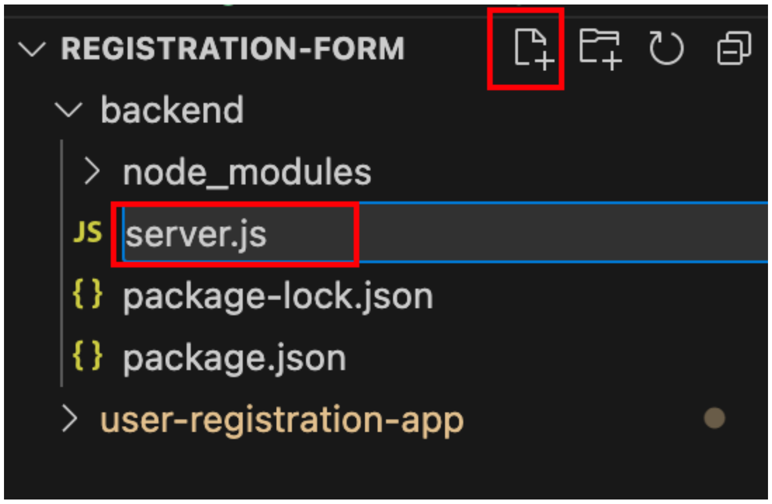


***Figure 6**: server.js created*

server.js

```
const express = require('express');
const cors = require('cors');
const app = express();

app.use(cors());
app.use(express.json());

const users = [];

app.post('/users', (req, res) => {
 const { name, email, password } = req.body;

 if (!name || !email || !password) {
   return res.status(400).json({ message: 'Name, email, and password are
required.' });
 }

 if (users.find(user => user.email === email)) {
   return res.status(409).json({ message: 'Email already registered.' });
 }
```

```
  users.push({ name, email, password });
  return res.status(201).json({ message: 'User registered successfully.' });
});

app.listen(3000, () => console.log('Server running on
http://localhost:3000'));
```

Delete app-module.ts

Delete or comment out app-module.ts if it exists.

Test the Integration

1. Start the Node.js server in the **"backend"** directory:

```
node server.js
```

2. Start the Angular development server in the **"user-registration-app"** directory:

```
ng serve
```

3. Open your web browser and navigate to `http://localhost:4200/register` (adjust according to your routing configuration).

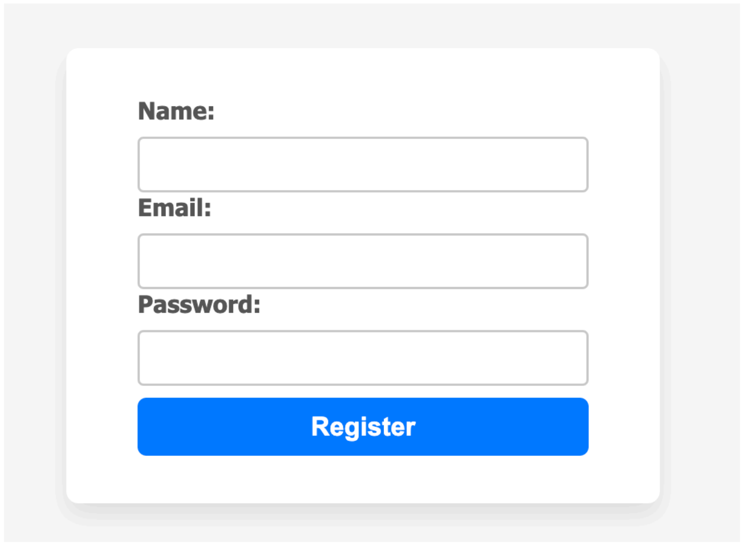
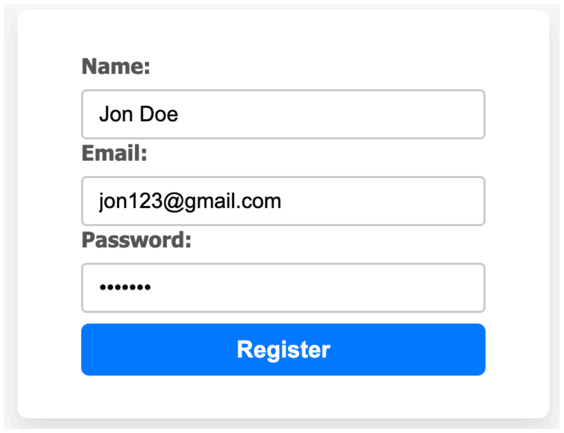


***Figure 6**: Registration Form UI*


Name:

Jon Doe

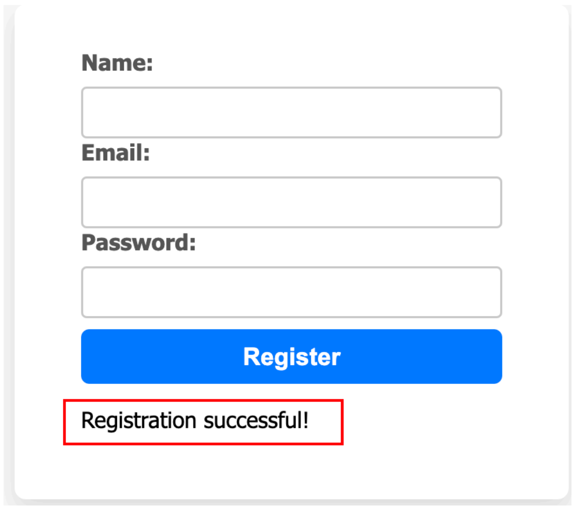
Email:

jon123@gmail.com

Password:

•••••••

***Figure 7**: Correct Data Format*

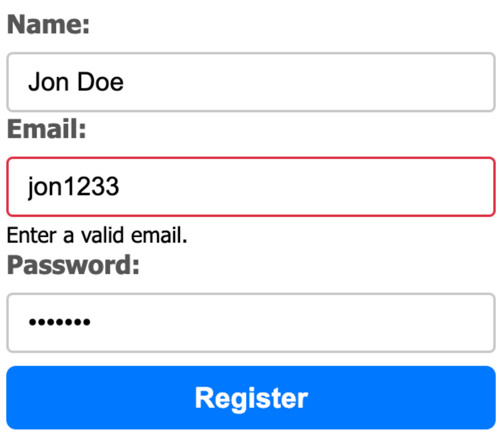

Name:

Email:

Password:

Registration successful!

***Figure 8**: Registration Successful*


Name:

Jon Doe

Email:

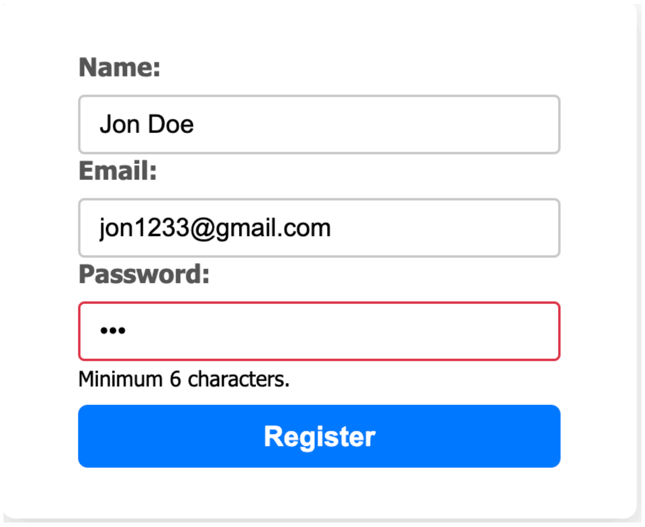
jon1233

Enter a valid email.

Password:

•••••••

***Figure 9**: Email Validation*


Name:

Jon Doe

Email:

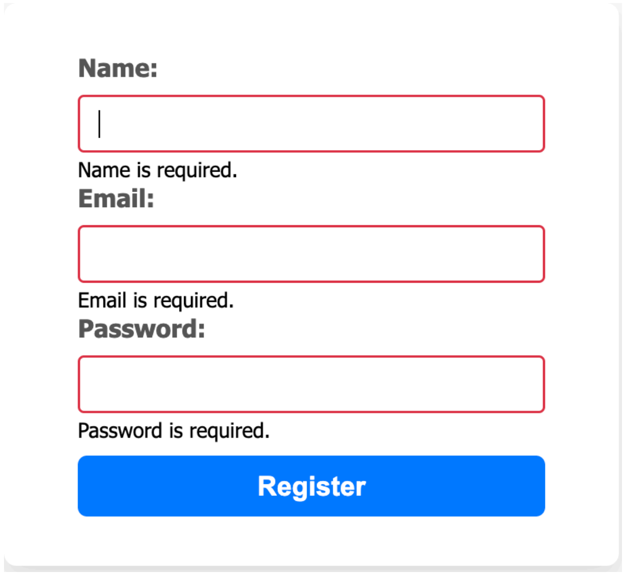
jon1233@gmail.com

Password:

•••

Minimum 6 characters.

***Figure 10**: Password Validation*


Name:

Name is required.

Email:

Email is required.

Password:

Password is required.

***Figure 11**: Compulsory fields to fill*

4. Fill the registration form with valid data and submit.
5. Verify that:
    - The form prevents submission if inputs are invalid (required fields, valid email, password length).
    - The backend successfully receives and stores the user data.
    - Success or error messages display appropriately based on API responses.

7. Enhancement (Optional)

For further learning and improvement, try the following enhancements:

- Add a `confirm password` field with validation to ensure the two password fields match.
- Replace the in-memory array in your Node.js backend with a real database such as MongoDB to persist data.
- Implement password hashing on the backend for secure storage.
- Improve UI/UX with better styling and feedback animations.
- Add authentication and login features to extend the user management system.

Conclusion

In this hands-on lab, you have developed a user registration form using Angular's reactive form system and integrated it with a Node.js REST API. You implemented form controls with validation, handled HTTP POST requests securely with Angular's `HttpClient`*, and ensured the backend API properly processed and stored the submitted user information.*

This exercise demonstrated full-stack integration, bridging frontend input handling and backend data persistence—an essential concept for real-world web applications. Moving forward, consider extending your project with authentication, enhanced form validations, and persistent database connections to create more comprehensive user management systems.