Experiment No.: 08

Title: Create a form and fetch data of form using angular.

Objectives:

1. To study form layouts in angular during building the web app.

Theory:

Forms in Angular take the standard capabilities of the HTML based forms and add an orchestration layer to help with creating custom form controls, and to supply great validation experiences. Applications use forms to enable users to log in, to update a profile, to enter sensitive information, and to perform many other data-entry tasks.

Angular provides two different approaches to handling user input through forms: reactive and template-driven. Both capture user input events from the view, validate the user input, create a form model and data model to update, and provide a way to track changes. Reactive forms are more robust: they're more scalable, reusable, and testable. Template-driven forms are useful for adding a simple form to an app, such as an email list signup form. A very basic form requirements and logic that can be managed solely in the template.

Reactive Forms-

Reactive forms use an explicit and immutable approach to managing the state of a form at a given point in time. Each change to the form state returns a new state, which maintains the integrity of the model between changes. Reactive forms are built around observable streams, where form inputs and values are provided as streams of input values, which can be accessed synchronously.

Steps to add single form control-

1) Registering the reactive forms module

To use reactive forms, import ReactiveFormsModule from the @angular/forms package and add it to NgModule's imports array.

2) Generating & Importing forms module

Generate a component for the control using command- **ng generate component ComponentName**

The FormControl class is the basic building block when using reactive forms. To register a single form control, import the FormControl class into newly created component and create a new instance of the form control to save as a class property.

Use the constructor of FormControl to set its initial value. By creating these controls in component class, get immediate access to listen for, update, and validate the state of the form input.

3) Registering the control in the template

After creating the control in the component class, it must be associate with a form control element in the template. Update the template with the form control using the formControl binding provided by FormControlDirective included in ReactiveFormsModule.

Displaying the component-

The form control assigned to name is displayed when the component is added to a template.

Template-driven forms-

In developing a form, it's important to create a data-entry experience that guides the user efficiently and effectively through the workflow.

Steps to build a simple form-

1) Build an Angular form with a component and template.
2) Use **ngModel** to create two-way data bindings for reading and writing input-control values.
3) Track state changes and the validity of form controls.
4) Provide visual feedback using special CSS classes that track the state of the controls.
5) Display validation errors to users and enable/disable form controls.
6) Share information across HTML elements using template reference variables.

Form Validation-

Template-driven Validation - Validators are added through attributes in the template.

Reactive form validation - Add validator functions directly to the form control model in the component class.

Validator Functions-

There are two types of validator functions: sync validators and async validators. **Sync validators:** functions that take a control instance and immediately return either a set of validation errors or null.

Async validators: functions that take a control instance and return a Promise or Observable that later emits a set of validation errors or null.

Key Concept: Forms, Reactive Forms, Template-driven forms, form validation