ANGULAR-17-CODES

LAB-1

To create a simple Angular application with 4 components arranged in 2 rows and 2 columns, each labeled as Component1, Component2, Component3, and Component4, you can follow these steps:

1. **Create Angular Application**:
   - Use Angular CLI to create a new Angular application by running `ng new my-app-name`.

2. **Generate Components**:
   - Generate 4 components using Angular CLI commands:
     - `ng generate component component1`
     - `ng generate component component2`
     - `ng generate component component3`
     - `ng generate component component4`

3. **Arrange Components**:
   - To arrange the components in 2 rows and 2 columns, you can use Angular's Flex Layout.
   - In the main app component HTML file (`app.component.html`), you can structure the layout using Flex Layout directives like `fxLayout`, `fxLayoutAlign`, and `fxFlex`.
   - Here is an example of how you can arrange the components in a 2x2 grid:

```html
<div fxLayout="row">
  <div fxLayout="column" fxFlex>
    <app-component1></app-component1>
    <app-component3></app-component3>
  </div>
  <div fxLayout="column" fxFlex>
    <app-component2></app-component2>
    <app-component4></app-component4>
  </div>
</div>
```

4. **Add Titles to Components**:
   - In each component's HTML file (e.g., `component1.component.html`), add a title to the component using HTML tags like `<h1>` or `<h2>`.
   - For example, in `component1.component.html`:

```html
<h2>Component 1</h2>
```

<!-- Add your component content here -->
```

5. **Run the Application**:
   - Start the Angular development server by running `ng serve --open` to see your application in the browser.

By following these steps, you can create a simple Angular application with 4 components arranged in 2 rows and 2 columns, each labeled as Component1, Component2, Component3, and Component4.
```

LAB-2

To create a parent and child component in Angular and demonstrate passing values between them, you can follow these steps:

1. **Create Parent Component**:
   - Generate a new component using Angular CLI: `ng generate component parent-component`.
   - In the `parent-component.component.ts` file, define a variable to hold the value that will be passed to the child component:

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

@Component({
  selector: 'app-parent-component',
  template: `
    <h2>Parent Component</h2>
    <app-child-component [value]="value"></app-child-component>
  `
})
export class ParentComponent {
  value = 'Hello from Parent';
}
```

2. **Create Child Component**:
   - Generate a new component using Angular CLI: `ng generate component child-component`.
   - In the `child-component.component.ts` file, define a property to receive the value from the parent component:

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

@Component({
  selector: 'app-child-component',
  template: `
    <h2>Child Component</h2>
    <p>Received value: {{ value }}</p>
  `
})
export class ChildComponent {
```

```
  @Input() value: string;
}
```

3. **Pass Value from Parent to Child**:
   - In the `parent-component.component.html` file, use the `[value]` binding to pass the value from the parent component to the child component:

```html
<app-child-component [value]="value"></app-child-component>
```

4. **Pass Value from Child to Parent**:
   - In the `child-component.component.html` file, use the `(valueChange)` binding to emit an event when the value changes in the child component.
   - In the `parent-component.component.ts` file, define a method to handle the event and update the value:

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

@Component({
  selector: 'app-parent-component',
  template: `
    <h2>Parent Component</h2>
    <app-child-component [value]="value" (valueChange)="onValueChange($event)"></
app-child-component>
  `
})
export class ParentComponent {
  value = 'Hello from Parent';

  onValueChange(newValue: string) {
    this.value = newValue;
  }
}
```

5. **Run the Application**:
   - Start the Angular development server by running `ng serve --open` to see your application in the browser.

By following these steps, you can create a parent and child component in Angular and demonstrate passing values between them. The value will be passed from the parent

component to the child component using the `[value]` binding, and from the child component to the parent component using the `(valueChange)` binding.

LAB-3

To create a text input in the parent component, pass a number to the child component, and create a custom directive to change the font size based on the number passed from the parent component, you can follow these steps:

1. **Create Parent Component**:
   - Generate a new component using Angular CLI: `ng generate component parent-component`.
   - In the `parent-component.component.ts` file, define a variable to hold the number that will be passed to the child component:

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

@Component({
 selector: 'app-parent-component',
 template: `
   <h2>Parent Component</h2>
   <input type="number" [(ngModel)]="number" />
   <app-child-component [number]="number"></app-child-component>
 `
})
export class ParentComponent {
 number = 10;
}
```

2. **Create Child Component**:
   - Generate a new component using Angular CLI: `ng generate component child-component`.
   - In the `child-component.component.ts` file, define a property to receive the number from the parent component:

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

@Component({
 selector: 'app-child-component',
 template: `
   <h2>Child Component</h2>
   <p>Received number: {{ number }}</p>
```

```
  `
})
export class ChildComponent {
  @Input() number: number;
}
```

3. **Create Custom Directive**:
   - Generate a new directive using Angular CLI: `ng generate directive font-size-directive`.
   - In the `font-size-directive.directive.ts` file, define the directive to change the font size based on the number passed from the parent component:

```typescript
import { Directive, ElementRef, Input } from '@angular/core';

@Directive({
  selector: '[fontSize]'
})
export class FontSizeDirective {
  constructor(private el: ElementRef) {}

  @Input('fontSize') set fontSize(size: number) {
    this.el.nativeElement.style.fontSize = `${size}px`;
  }
}
```

4. **Apply Custom Directive to Child Component**:
   - In the `child-component.component.html` file, apply the custom directive to the `<p>` element:

```html
<p fontSize="number">Received number: {{ number }}</p>
```

5. **Run the Application**:
   - Start the Angular development server by running `ng serve --open` to see your application in the browser.

By following these steps, you can create a text input in the parent component, pass a number to the child component, and create a custom directive to change the font size based on the number passed from the parent component. The number will be passed from the parent component to the child component using the `[number]` binding, and the font size will be changed in the child component using the custom directive.

LAB-4

To create a custom pipe in Angular to display the first two characters of a string, you can follow these steps:

1. **Generate a new pipe using Angular CLI**: `ng generate pipe first-two-characters-pipe`.

2. **Define the custom pipe in the `first-two-characters-pipe.ts` file**:

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

@Pipe({
  name: 'firstTwoCharacters'
})
export class FirstTwoCharactersPipe implements PipeTransform {
  transform(value: string): string {
    return value.substring(0, 2);
  }
}
```

3. **Use the custom pipe in the component's HTML file**:

```html
<p>{{ 'Hello World' | firstTwoCharacters }}</p>
```

4. **Run the Application**:
   - Start the Angular development server by running `ng serve --open` to see your application in the browser.

By following these steps, you can create a custom pipe in Angular to display the first two characters of a string. The pipe will be applied to the string using the `|` pipe operator in the component's HTML file.

LAB-5

To create a simple Angular form with 4 text fields and perform a mandatory check using ReactForm validation, you can follow these steps:

1. **Generate a new component using Angular CLI**: `ng generate component form-component`.
2. **In the `form-component.component.ts` file, import the necessary modules and define the form model and validation rules**:

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

@Component({
 selector: 'app-form-component',
 template: `
  <form [formGroup]="form" (ngSubmit)="onSubmit()">
   <div>
    <label>Name</label>
    <input type="text" formControlName="name" />
    <div *ngIf="form.get('name').errors && form.get('name').touched">
     <div *ngIf="form.get('name').errors.required">Name is required</div>
    </div>
   </div>
   <div>
    <label>Email</label>
    <input type="email" formControlName="email" />
    <div *ngIf="form.get('email').errors && form.get('email').touched">
     <div *ngIf="form.get('email').errors.required">Email is required</div>
     <div *ngIf="form.get('email').errors.email">Invalid email</div>
    </div>
   </div>
   <div>
    <label>Age</label>
    <input type="number" formControlName="age" />
    <div *ngIf="form.get('age').errors && form.get('age').touched">
     <div *ngIf="form.get('age').errors.required">Age is required</div>
     <div *ngIf="form.get('age').errors.min">Age must be greater than or equal to 18</div>
    </div>
   </div>
   <div>
```

```
      <label>Address</label>
      <input type="text" formControlName="address" />
      <div *ngIf="form.get('address').errors && form.get('address').touched">
        <div *ngIf="form.get('address').errors.required">Address is required</div>
      </div>
    </div>
    <button type="submit">Submit</button>
  </form>
  `
})
export class FormComponent implements OnInit {
  form: FormGroup;

  constructor(private fb: FormBuilder) {}

  ngOnInit() {
    this.form = this.fb.group({
      name: ['', Validators.required],
      email: ['', [Validators.required, Validators.email]],
      age: ['', [Validators.required, Validators.min(18)]],
      address: ['', Validators.required]
    });
  }

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

3. **Run the Application**:
   - Start the Angular development server by running `ng serve --open` to see your application in the browser.

By following these steps, you can create a simple Angular form with 4 text fields and perform a mandatory check using ReactForm validation. The form will be validated using the `Validators` module from Angular, and any validation errors will be displayed using Angular's reactive form validation API.

LAB-6

To create an Angular application with 3 pages (Home, About, and Contact) and demonstrate routing from one page to another on the click of a button in each page using Angular 17, you can follow these steps:

1. **Generate Components**:
   - Generate components for Home, About, and Contact pages using Angular CLI:
     - `ng generate component home`
     - `ng generate component about`
     - `ng generate component contact`

2. **Set up Routing**:
   - Define the routes for the Home, About, and Contact pages in the `app-routing.module.ts` file:

```typescript
import { NgModule } from '@angular/core';
import { RouterModule, Routes } 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: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent },
  { path: '', redirectTo: '/home', pathMatch: 'full' },
];

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

3. **Update App Component**:
   - Update the `app.component.html` file to include buttons that navigate to different pages:

```html
<button routerLink="/home">Home</button>
```

```
<button routerLink="/about">About</button>
<button routerLink="/contact">Contact</button>
<router-outlet></router-outlet>
```

4. **Add Navigation Buttons**:
   - In each component's HTML file (e.g., `home.component.html`, `about.component.html`, `contact.component.html`), add buttons to navigate to other pages:

```html
<button routerLink="/about">Go to About</button>
<button routerLink="/contact">Go to Contact</button>
```

5. **Run the Application**:
   - Start the Angular development server by running `ng serve --open` to see your application in the browser.

By following these steps, you can create an Angular application with 3 pages (Home, About, and Contact) and demonstrate routing from one page to another on the click of a button in each page using Angular 17. The navigation between pages will be handled by Angular's built-in routing functionality.

LAB-7

To create an application demonstrating the working of observables and subscribers using the RxJS library with a simple array in Angular, you can follow these steps:

1. **Install RxJS**:
   - Ensure that RxJS is already included in your Angular project. If not, you can install it using npm:
   ```bash
   npm install rxjs
   ```

2. **Create Service**:
   - Generate a service using Angular CLI to handle the observable logic:
   ```bash
   ng generate service data
   ```

3. **Implement Observable Logic**:
   - In the service file (`data.service.ts`), create an observable that emits values from a simple array:

```typescript
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';

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

  private data: string[] = ['Apple', 'Banana', 'Cherry'];

  getData(): Observable<string[]> {
    return of(this.data);
  }
}
```

4. **Subscribe to the Observable**:
   - In a component where you want to subscribe to the observable (e.g., `app.component.ts`), inject the service and subscribe to the observable:

```typescript
import { Component, OnInit } from '@angular/core';
import { DataService } from './data.service';

@Component({
 selector: 'app-root',
 template: `
  <h2>Observable Data:</h2>
  <ul>
   <li *ngFor="let item of items">{{ item }}</li>
  </ul>
 `
})
export class AppComponent implements OnInit {
 items: string[];

 constructor(private dataService: DataService) {}

 ngOnInit() {
  this.dataService.getData().subscribe(data => {
   this.items = data;
  });
 }
}
```

5. **Display Data in Template**:
  - In the component's HTML file (e.g., `app.component.html`), display the data fetched from the observable:

```html
<h2>Observable Data:</h2>
<ul>
 <li *ngFor="let item of items">{{ item }}</li>
</ul>
```

6. **Run the Application**:
  - Start the Angular development server by running `ng serve --open` to see your application in the browser.

By following these steps, you can create an Angular application that demonstrates the working of observables and subscribers using the RxJS library with a simple array. The observable will emit values from the array, and the subscriber will receive and display these

values in the application.

LAB-8

To create an Angular application with a button that fetches data from an external service (e.g., Random User Generator API) and displays the details upon clicking the button, you can follow these steps:

1. **Install HttpClient Module**:
   - Ensure that the `HttpClientModule` is imported in your Angular application. If not, you can import it in the `app.module.ts` file:

```typescript
import { HttpClientModule } from '@angular/common/http';

@NgModule({
 imports: [
  HttpClientModule
 ]
})
```

2. **Create Service to Fetch Data**:
   - Create a service to handle the HTTP request to fetch data from the Random User Generator API. You can use Angular CLI to generate a service:
    ```bash
    ng generate service user
    ```

3. **Implement Service Logic**:
   - In the service file (`user.service.ts`), make an HTTP GET request to fetch user data from the Random User Generator API:

```typescript
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
 providedIn: 'root'
})
export class UserService {
 private apiUrl = 'https://randomuser.me/api/';

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

```
  getUserData() {
    return this.http.get(this.apiUrl);
  }
}
```

4. **Create Component**:
   - Generate a component using Angular CLI where you will implement the button click functionality and display the fetched user data:
     ```bash
     ng generate component user-details
     ```

5. **Implement Component Logic**:
   - In the component file (`user-details.component.ts`), inject the `UserService`, call the `getUserData()` method on button click, and display the fetched user details:

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

@Component({
  selector: 'app-user-details',
  template: `
    <button (click)="fetchUserData()">Fetch User Data</button>
    <div *ngIf="userData">
      <img [src]="userData.picture.large" alt="User Image">
      <p>Name: {{ userData.name.first }} {{ userData.name.last }}</p>
      <p>Email: {{ userData.email }}</p>
    </div>
  `
})
export class UserDetailsComponent {
  userData: any;

  constructor(private userService: UserService) {}

  fetchUserData() {
    this.userService.getUserData().subscribe((data: any) => {
      this.userData = data.results[0];
    });
  }
}
```

6. **Update App Module**:
   - Import and declare the `UserDetailsComponent` in the `app.module.ts` file.

7. **Run the Application**:
   - Start the Angular development server by running `ng serve --open` to see your application in the browser.

By following these steps, you can create an Angular application with a button that fetches data from an external service (Random User Generator API) and displays the user details upon clicking the button.

LAB-9

To build a Node.js server with Express to provide API endpoints for GET requests and invoke this service from an Angular component, you can follow these steps:

1. **Create a Node.js Server with Express**:
   - Create a new Node.js project and install Express:
     ```bash
     npm init -y
     npm install express
     ```

2. **Set up Express Server**:
   - Create a `server.js` file and set up a basic Express server with a GET endpoint:

```javascript
const express = require('express');
const app = express();
const PORT = 3000;

app.get('/api/data', (req, res) => {
  const data = { message: 'Hello from the server!' };
  res.json(data);
});

app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

3. **Start the Node.js Server**:
   - Run the Node.js server by executing `node server.js`.

4. **Invoke the API from Angular Component**:
   - In your Angular project, create a service to make HTTP requests to the Node.js server. Use Angular's `HttpClientModule` for this purpose.
   - Generate a service using Angular CLI:
     ```bash
     ng generate service api
     ```

5. **Implement Service Logic**:
   - In the service file (`api.service.ts`), make an HTTP GET request to fetch data from the

Node.js server:

```typescript
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class ApiService {
  private apiUrl = 'http://localhost:3000/api/data';

  constructor(private http: HttpClient) {}

  getData() {
    return this.http.get(this.apiUrl);
  }
}
```

6. **Invoke Service in Angular Component**:
   - In the component where you want to fetch data (e.g., `app.component.ts`), inject the `ApiService` and call the `getData()` method:

```typescript
import { Component } from '@angular/core';
import { ApiService } from './api.service';

@Component({
  selector: 'app-root',
  template: `
    <button (click)="fetchData()">Fetch Data</button>
    <div *ngIf="data">
      <p>{{ data.message }}</p>
    </div>
  `
})
export class AppComponent {
  data: any;

  constructor(private apiService: ApiService) {}

  fetchData() {
    this.apiService.getData().subscribe((response: any) => {
```

```
    this.data = response;
   });
  }
}
```

7. **Update App Module**:
   - Ensure that the `HttpClientModule` is imported in your Angular application.

8. **Run the Application**:
   - Start the Angular development server by running `ng serve --open` to see your application in the browser.

By following these steps, you can build a Node.js server with Express to provide API endpoints for GET requests and invoke this service from an Angular component to fetch and display data from the server.

LAB-10

To build a Node.js server with Express to provide API PUT endpoints for adding new objects to an array and invoke this service from an Angular component, you can follow these steps:

1. **Create a Node.js Server with Express**:
   - Create a new Node.js project and install Express:
     ```bash
     npm init -y
     npm install express body-parser
     ```

2. **Set up Express Server**:
   - Create a `server.js` file and set up an Express server with a PUT endpoint to push new objects to an array:

```javascript
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const PORT = 3000;

app.use(bodyParser.json());

let data = [];

app.put('/api/data', (req, res) => {
  const newData = req.body;
  data.push(newData);
  res.json({ message: 'Data added successfully' });
});

app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

3. **Start the Node.js Server**:
   - Run the Node.js server by executing `node server.js`.

4. **Invoke the API from Angular Component**:
   - In your Angular project, create a service to make HTTP requests to the Node.js server for adding new objects.

- Generate a service using Angular CLI:
  ```bash
  ng generate service api
  ```

5. **Implement Service Logic**:
   - In the service file (`api.service.ts`), make an HTTP PUT request to add new data objects to the server:

```typescript
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
 providedIn: 'root'
})
export class ApiService {
 private apiUrl = 'http://localhost:3000/api/data';

 constructor(private http: HttpClient) {}

 addData(newData: any) {
  return this.http.put(this.apiUrl, newData);
 }
}
```

6. **Invoke Service in Angular Component**:
   - In the component where you want to add new data (e.g., `app.component.ts`), inject the `ApiService` and call the `addData()` method:

```typescript
import { Component } from '@angular/core';
import { ApiService } from './api.service';

@Component({
 selector: 'app-root',
 template: `
  <button (click)="addNewData()">Add New Data</button>
 `
})
export class AppComponent {

 constructor(private apiService: ApiService) {}
```

```
  addNewData() {
    const newData = { name: 'New Data', value: '123' };
    this.apiService.addData(newData).subscribe((response: any) => {
      console.log(response);
    });
  }
}
```

7. **Update App Module**:
   - Ensure that the `HttpClientModule` is imported in your Angular application.

8. **Run the Application**:
   - Start the Angular development server by running `ng serve --open` to see your application in the browser.

By following these steps, you can build a Node.js server with Express to provide API PUT endpoints for adding new objects to an array and invoke this service from an Angular component to push new objects to the server's array.