# TASK 6 DOCUMENTATION

**1️⃣ UserService — State Management & API Simulation**

ts

CopyEdit

// user.service.ts

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

import { BehaviorSubject } from 'rxjs';

import { User } from '../models/user.model';

@Injectable({

providedIn: 'root'

})

export class UserService {

private userSubject = new BehaviorSubject<User>({

firstName: '',

lastName: '',

userName: '',

email: '',

profilePicture: ''

});

user$ = this.userSubject.asObservable();

updateUser(user: User) {

this.userSubject.next(user);

}

}

**🔍 Explanation:**

* @Injectable({ providedIn: 'root' }): Registers service globally (singleton).
* BehaviorSubject: Stores current user state and emits updates to any subscribers.
* user$: Observable that other components subscribe to for updates.
* updateUser(): Pushes new user data to all listeners (like your profile component).

**2️⃣ ProfileComponent — Main Form + Live Preview**

**📄 profile.component.ts**

ts

CopyEdit

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

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { UserService } from '../../services/user.service';

import { User } from '../../models/user.model';

@Component({

selector: 'app-profile',

templateUrl: './profile.component.html',

styleUrls: ['./profile.component.css']

})

export class ProfileComponent implements OnInit {

userForm!: FormGroup;

previewData: User = {

firstName: '',

lastName: '',

userName: '',

email: '',

profilePicture: ''

};

constructor(private fb: FormBuilder, private userService: UserService) {}

ngOnInit(): void {

// Initialize form with validators

this.userForm = this.fb.group({

firstName: ['', [Validators.required, Validators.minLength(3)]],

lastName: ['', [Validators.required, Validators.minLength(3)]],

userName: ['', [Validators.required, Validators.minLength(3)]],

email: ['', [Validators.required, Validators.email]],

profilePicture: ['']

});

// Sync form and preview

this.userForm.valueChanges.subscribe(val => {

this.previewData = val;

});

// Load saved user data

this.userService.user$.subscribe(user => {

this.userForm.patchValue(user);

this.previewData = user;

});

}

onSubmit() {

if (this.userForm.valid) {

this.userService.updateUser(this.userForm.value);

alert('User profile updated!');

}

}

onProfilePictureChange(newUrl: string) {

this.userForm.get('profilePicture')?.setValue(newUrl);

}

}

**🔍 Explanation:**

| **Code Part** | **What & Why** |
| --- | --- |
| userForm | Reactive form holding user data. |
| previewData | Two-way preview of form values (for live update in HTML). |
| FormBuilder | Utility for building FormGroup with cleaner syntax. |
| Validators | Enforce form rules (like required, minLength, etc). |
| valueChanges.subscribe | Auto update preview UI on form change. |
| userService.user$.subscribe | Load saved data from service to form. |
| onSubmit() | Validates and updates service with new user data. |
| onProfilePictureChange() | Gets child component’s picture and updates form. |

**📄 profile.component.html**

html

CopyEdit

<form [formGroup]="userForm" (ngSubmit)="onSubmit()">

<input type="text" formControlName="firstName" placeholder="First Name">

<input type="text" formControlName="lastName" placeholder="Last Name">

<input type="text" formControlName="userName" placeholder="Username">

<input type="email" formControlName="email" placeholder="Email">

<app-profile-picture

[imageUrl]="previewData.profilePicture"

(imageChange)="onProfilePictureChange($event)">

</app-profile-picture>

<button type="submit" [disabled]="userForm.invalid">Update Profile</button>

</form>

<!-- Live Preview -->

<h3>Live Preview:</h3>

<ul>

<li>First Name: {{ previewData.firstName }}</li>

<li>Last Name: {{ previewData.lastName }}</li>

<li>Username: {{ previewData.userName }}</li>

<li>Email: {{ previewData.email }}</li>

<li>Image: <img [src]="previewData.profilePicture" width="100" /></li>

</ul>

**3️⃣ ProfilePictureComponent — Child with @Input/@Output**

**📄 profile-picture.component.ts**

ts

CopyEdit

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

@Component({

selector: 'app-profile-picture',

templateUrl: './profile-picture.component.html',

})

export class ProfilePictureComponent {

@Input() imageUrl: string = '';

@Output() imageChange = new EventEmitter<string>();

onImageSelect(event: any) {

const file = event.target.files[0];

if (file) {

const reader = new FileReader();

reader.onload = () => {

const result = reader.result as string;

this.imageChange.emit(result);

};

reader.readAsDataURL(file);

}

}

}

**🔍 Explanation:**

| **Line** | **What & Why** |
| --- | --- |
| @Input() imageUrl | Gets current picture from parent to display. |
| @Output() imageChange | Sends new image URL back to parent component. |
| onImageSelect() | Converts selected image to Base64 and emits it. |
| FileReader | Used to read uploaded image as data URL. |

**📄 profile-picture.component.html**

html

CopyEdit

<div>

<img [src]="imageUrl" width="100" \*ngIf="imageUrl">

<input type="file" (change)="onImageSelect($event)">

</div>

**✅ What You’ve Practiced**

| **Feature** | **Purpose** |
| --- | --- |
| Reactive Form | Structured, scalable form logic. |
| FormBuilder | Cleaner syntax to create forms. |
| valueChanges | Real-time preview of data. |
| @Input() / @Output() | Child-parent communication. |
| BehaviorSubject | Shared state across app via service. |