Purpose of Web Frameworks:

Web Frameworks provide a way to make web development easier (especially for dynamic web applications) by proving data layout structures (eg. MVC), libraries (eg. for UI, database or session management), and mainly help reuse the defined components.

Most frameworks provide a way to organise data for web apps to keep them maintainable and easily debuggable. The data is organised in components which can be any part of an application (usually grouped together markup, style and script), linked to each other to create dynamic web apps. The defined change in any components can trigger the events in other components using state management.

The report below compares the widely used Angular web framework and the new Svelte framework, and provides reasons for why one would be better than the other for creating the web application (a charity App which connects beneficiaries and donors) for my project.

Comparison

The comparison will be as follows:

1. Proof of concept code examples in each framework.
2. Complexity of coding in both frameworks
3. Justification for chosen framework

Proof of concept code examples:

Components:

Angular	Svelte
Components: In Angular each component has a markup, CSS and Scripts, and separate files for each of these allows the code to be cleaner and more maintainable, however they can be put into just the component.ts file, which exports the complete component as a class. Creating components: ‘ng g n <component>’ can be used in CLI to generate a new component which has the general structure and is wrapped in the class/	In Svelt the same is true, but the code for style, template and script is put into a single file and String and function interpolation is easier, as shown in next example. The components are created by adding the file extension .svlte to file.

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


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


 constructor() { }


 ngOnInit(): void {
 }
}
```

```
<script>
 export let itemName = "",
   itemPrice = 0,
   addItemToInventory;
 $: disabled = itemName == "";
 let donate = true;
</script>
<div></div>
<style>
</style>
```

Hooks and Data Binding: In both Angular and Svelte the data usually flows from parent to child components. But to pass the data from child to parent the @Input() and @Output() directives can be used. @Input to receive the data from parent and @Output to send the data.

```
import { Component, Input } from '@angular/core'; // First, import Input
export class ItemDetailComponent {
  @Input() itemName = ''; // decorate the property with @Input()
}
```

In Svelte the data can be passed from parent to child by either using the export keyword or using 'store' to store the data that will be shared between components.

```
<script>
 export let itemName = "",
   itemPrice = 0,
   addItemToInventory;
 $: disabled = itemName == "";
 let donate = true;
</script>
```

Services and Store: In Angular 'services' are used to include the data that is shared between components.

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

```
export class CartService {
  items: Product[] = [];
}
```

Injectable service needs to be imported into each component where the shared data will be used.

In Svelte 'stores' are used

```
import { writable} from 'svelte/store'

export const user = writable(null)
export const items = writable([])
```

Here the writable library allows the data on store to be writable and not just read only.

The global data is referenced using $ sign before the variable and can be modified from anywhere in the app.

Directives: Angular provide many directives the most used being, ngIf, ngFor, ngStyle, ngModel
 	
`<h2>Items</h2>`
 	
`<div *ngFor="let item of items">`
`{{item}}`
`</div>`
	Svelte provides the same but in an easy format that can be understood by any new developer.
	
	`{#if items.length}>0}`
	`Here are the items:`
	`{#each items as item}`
	`{item}`
	`{/each}`
	`{/if}`

Fetching Data from Server:
To make http requests in angular HTTP can be imported.
 	
`export class ConfigService {`
`constructor(private http: HttpClient) { }`
`}`
`configUrl = 'assets/config.json';`
`getConfig() {`
`return this.http.get<Config>(this.configUrl);`
`}`
	In Svelte simple async functions linked with general libraries like axios or fetch to fetch data from server:
	
	`Async getUserData(async () => {`
	`const { data } = await`
	`axios.get("http://abc.com/api/auth/user");`
	`$user = data.user;`
	`});`

Data Binding: Angular used [] to bind the data. `<input [value]="name">`	Single curly brackets are used to bind data. `<input value={name}>`

State: Changing the data in variables when a variable that they depend upon is changed requires @input and @output directives.	In Svelt the same thing can be achieved using '$:' before the variable, the change in whose value would trigger the change in all other values that depend on it, without redeclaring them.

Analysis:

After practicing with Angular and Svelte, I've found Angular requires getting used to the Typescript format, its ng directives, and the way of moving data from the child components (whereas in Svelte just the bind keyword can be used to bind the required data within components, and stores can be used to keep global data)

It's just the easy format and quick data binding that makes me decide to use Svelte for this project. This project would not require much maintenance and addition of many more components in the long run, therefore Svelte will get the job done with more efficiency and tidiness in the code of this small project.