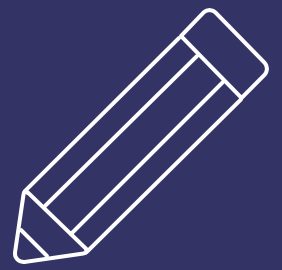
Angular Control Value Accessor

Pierre Roux

- A look at how to better handle user input on web frontends

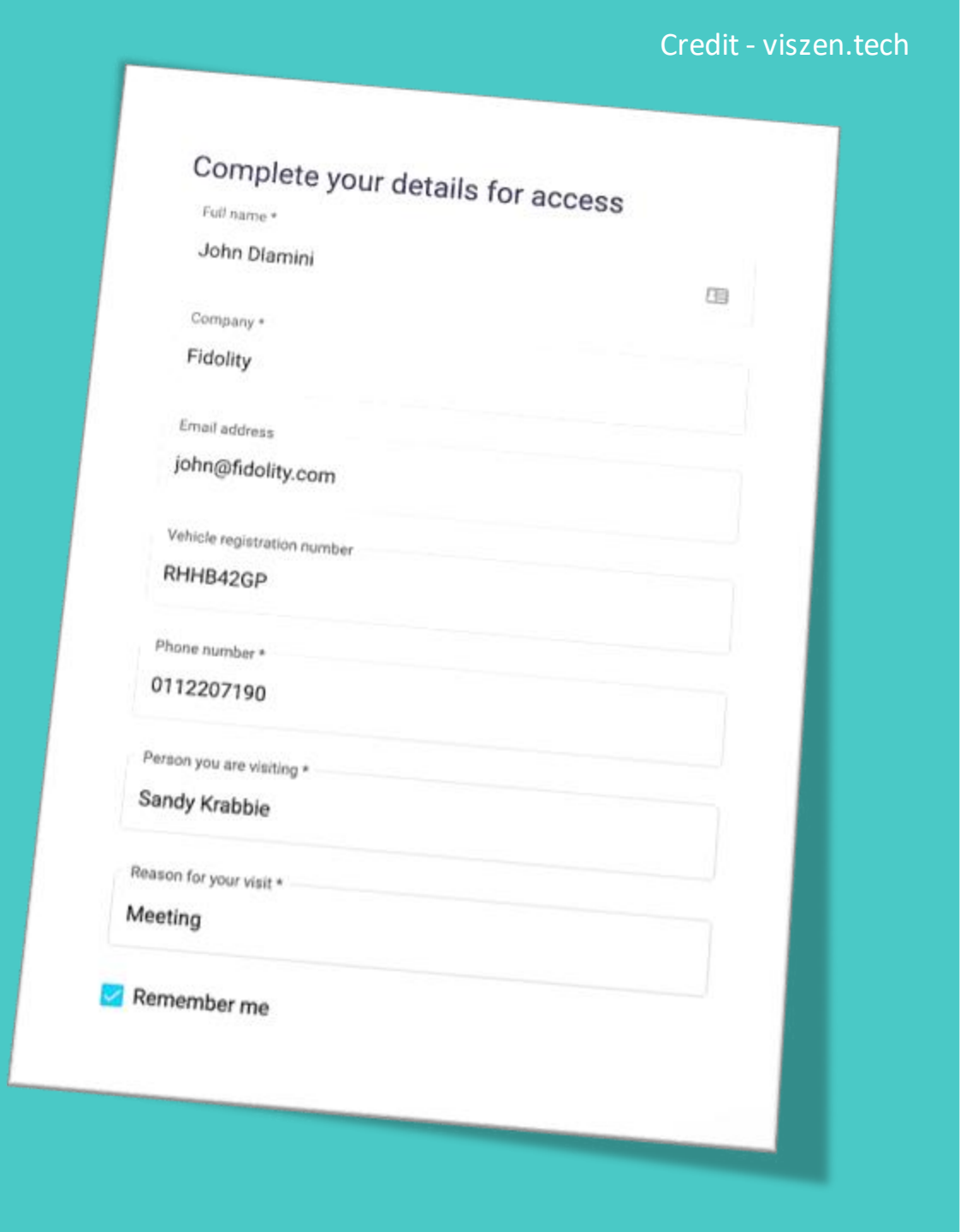


What's this about?

- Getting user details from keyboard inputs to the API
  - How is this done in plain Javascript
  - How is this done in Angular
- What is the clear, concise, and maintainable way of solving user input
- Re-using fundamental pieces of components
  - What are the benefits and drawbacks

Input forms

- It is like a paper form to fill in, but on your browser
- Ask the user for some info, like name, email address, contact number, etc.
- Package this info into a neat object, and send that to the API for processing



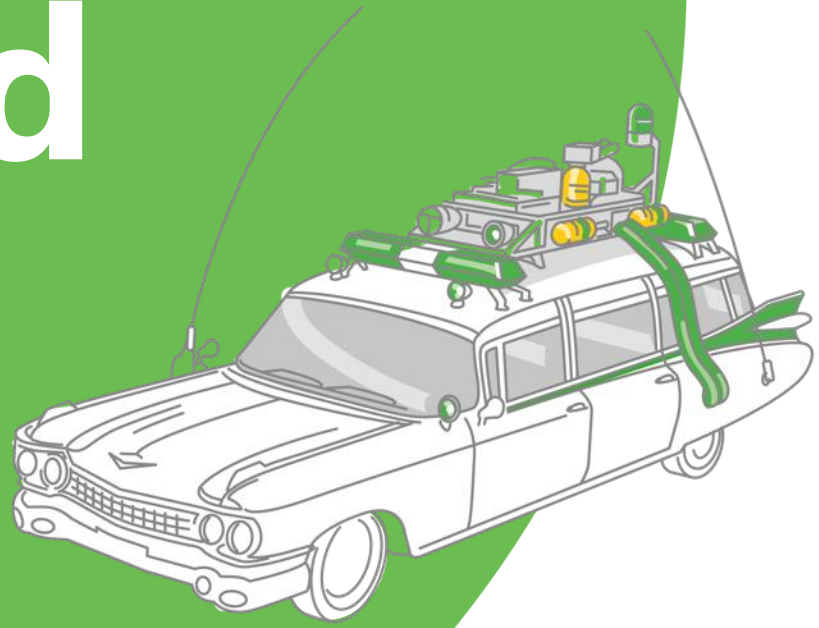
Encapsulates all the logic around this specific element

- Stores the value that the user provided, like a number/string
- Validates the value against a preset rule
  - Is it optional or required?
  - Must it be a number/string?
- Disables the element when it is not ready for input
  - The user must first fill in another form control
  - Or this form control is mutually exclusive with another

Form Control

- An entity that controls form input
  - For a single detail like
    - "Phone Number"

Time for a road trip



- Demo
  - Go to https://control-value-accessor.blaarkies.com/
  - Checkout how it feels, but don't get lost, we need you in this presentation!

Native/HTML Input

- Every HTML element is just a big Javascript object
  - This object contains everything that makes the element "alive"
  - Every value, title, on-click function, etc. is contained in this object
- It is possible to get the input value from this object

- See what this looks like on any website
  - In Chrome, right-click the element, select "*Inspect*"
  - DevTools will open up showing the HTML code. Right-click the element, select "*Store as global variable*"
  - In the console below, it will show "*temp1*" as a new variable
  - Type in "*{temp1}*", press *enter*, click the arrows to see what's inside

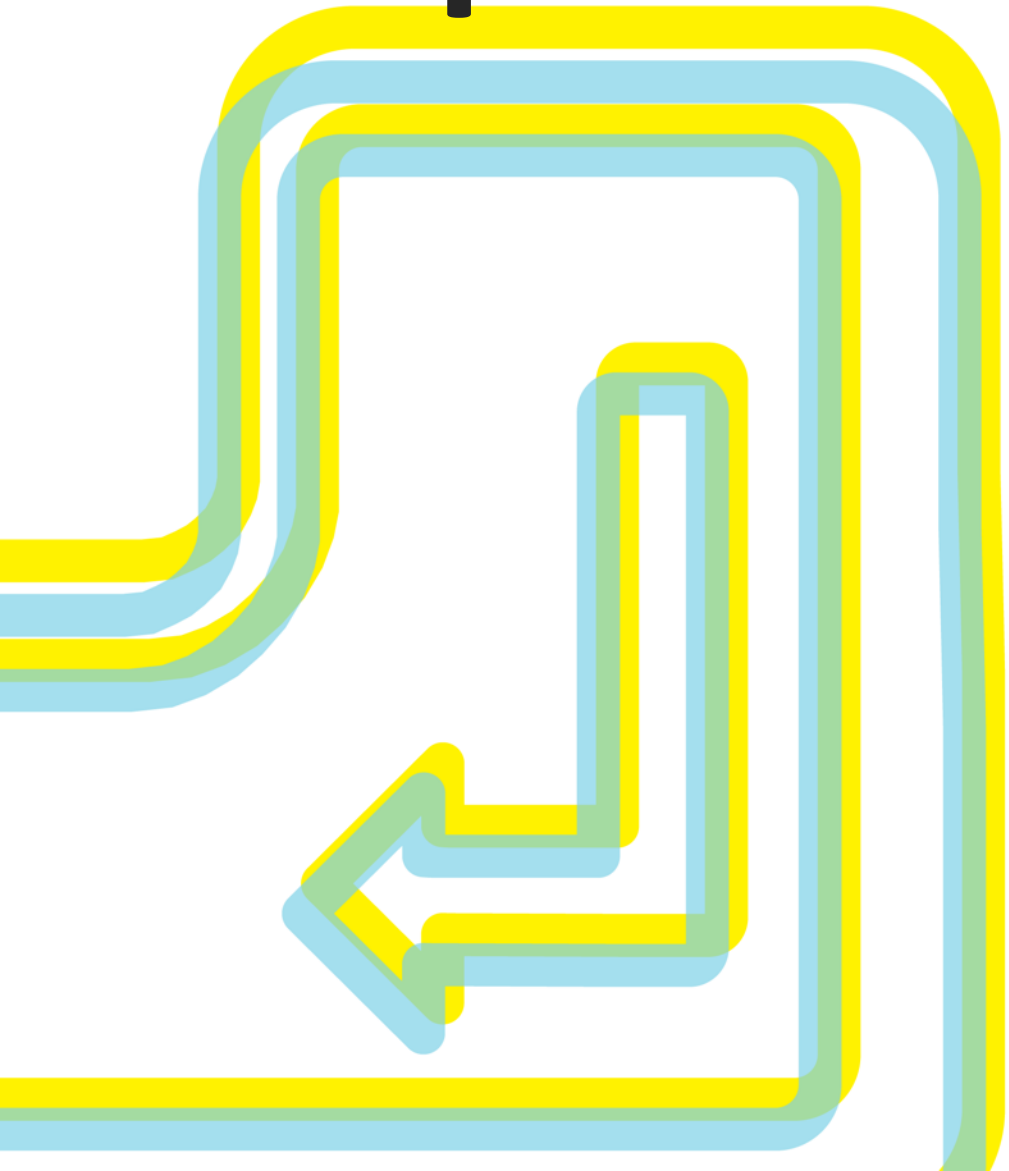
Angular Inputs

How does Angular simplify user input?

Template-driven forms

Reactive forms

Template-driven forms

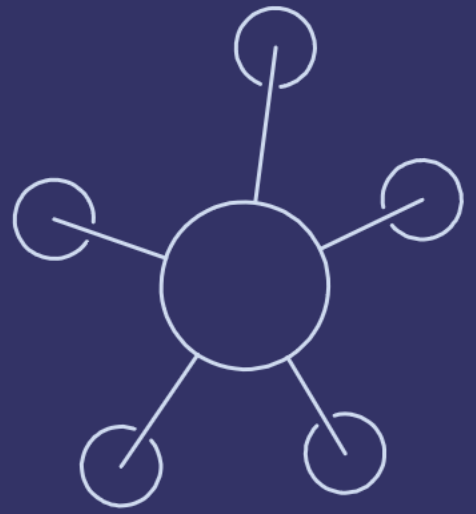


- Define element with

```
<input type="email" [(ngModel)]="myVarEmail"/>
```

- *myVarEmail* will now always be the same as the *input element's value*
- The *input element's value is A*
- *myVarEmail is B*
  - Change A, then B will also change
  - Change B, then A will also change

Reactive forms



- Define element with
  <input type="email" [formControl]="myEmailControl"/>
- *myEmailControl* is a "new FormControl()"
  - An object that controls this input form
  - Angular tells the input element how to use *myEmailControl*

Changes to the input element will cause changes in *myEmailControl,* and vice versa

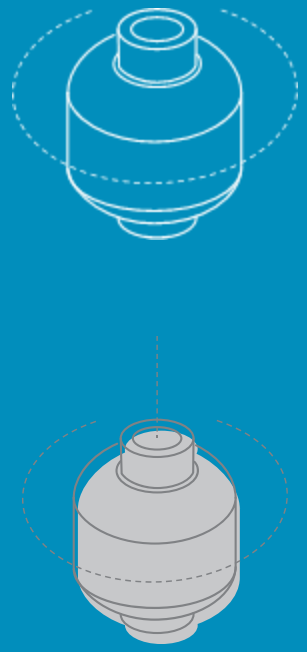
Form controls are great at abstracting away mundane input logic

Let's make an Autocomplete!

Important

› Focus on re-use
› Make it dev-friendly
› Must be expandable for future additions

› It will re-use a simple input field for the "search" query
› It will show a dropdown of options to select
  • The actual value comes from selecting an option



Re-usable Input Component

- Make a component that houses styling, labels, error messages, and the actual HTML input
- Use this component any time you ask the user for a specific detail, like “Contact Number”
- Implement the *ControlValueAccessor* interface on this component to support [formControl]

ControlValueAccessor

Implement the interface

- `value` is only for internal storage, it does nothing automatically
- Calling `onChange(value)` is what updates the control's value
- `disabled` and `setDisabledState()` can be used to style the component to visually appear inactive
- This component now directly uses Angular's form API

```
@Component({
 selector: 'app-input',
 providers: [{
  provide: NG_VALUE_ACCESSOR,
  useExisting: forwardRef(()=>InputComponent),
  multi: true,
 }],
})
export class InputComponent
        implements ControlValueAccessor {
```

```
 value: any;
 onChange: (value: any) => void;
 onTouched: (value: any) => void;
 disabled: boolean;

 writeValue(value: any) {}
 registerOnChange(fn: any) {}
 registerOnTouched(fn: any) {}
 setDisabledState(isDisabled: boolean) {}
```

Re-use

The Autocomplete uses the Input Component

- We use the app-input component as a search box
- Below the search box, we add dropdown logic and styling
- Remember to update the search box as well (when committing changes to the autocomplete)

```
<app-input #inputRef [label]="label"></app-input>

<div class="dropdown-container"
     [class.open]="inputRef.isActive">
  <button class="dropdown-option"
          *ngFor="let option of filteredOptions"
          (click)="selectOption(option)">
    {{option.label}}
  </button>
</div>
```

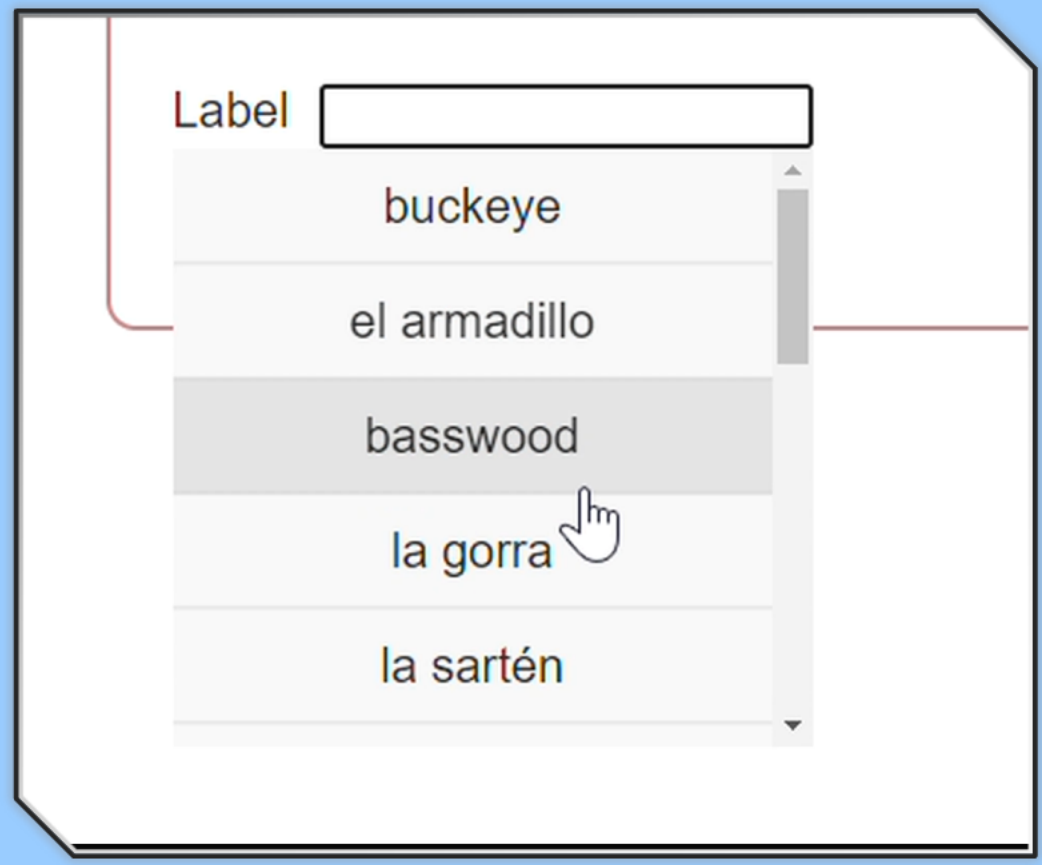
```
selectOption(option: LabeledOption<any>) {
  this.writeValue(option.value);
  this.inputComponent.writeValue(option.label);
}
```

Dev-friendly

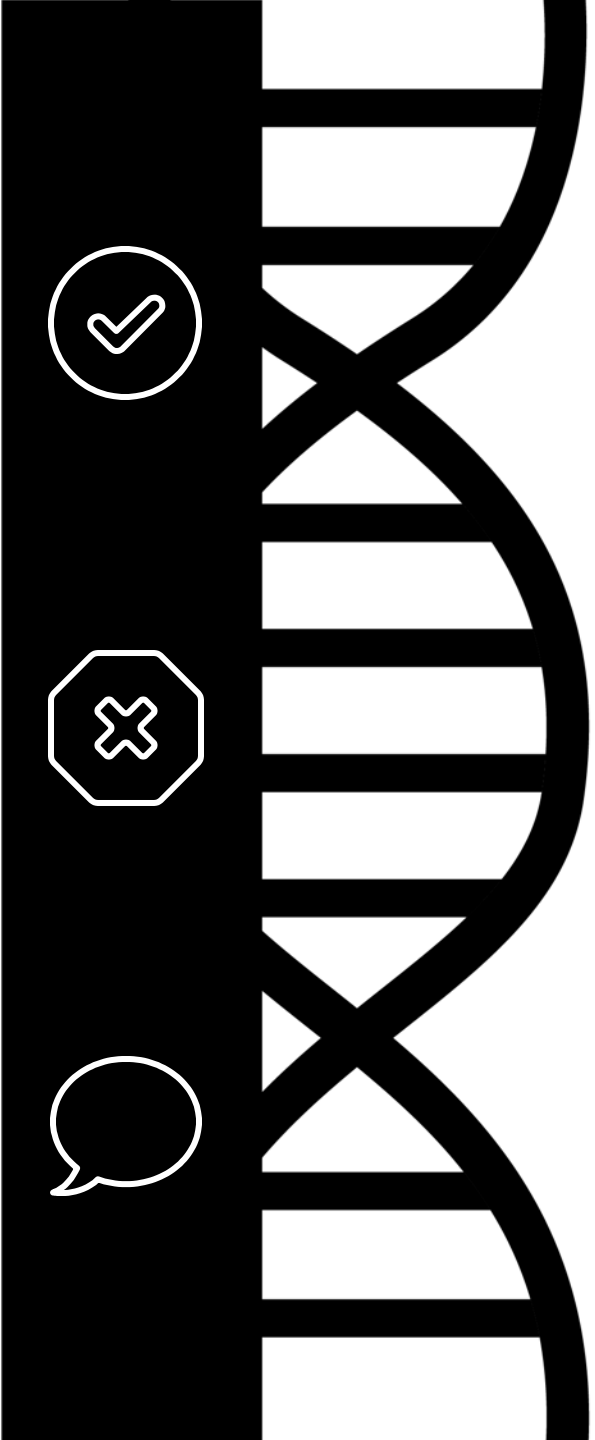
- An autocomplete component that is super easy to use

```
<app-autocomplete label="Tree Address"
                  [formControl]="treeControl"
                  [options]="trees">
</app-autocomplete>
```

- The *options* input can later be improved to handle promises/observables for better API performance



Note: Pretty styling still to be added



FAQ

- Can this be done without ControlValueAccessor?
  - Yes, but actually no
  - Monkey chained controls are bad at doing anything except simple form inputs
- Is my code correct? I can't find a StackOverflow to confirm it
  - Me too, that's why I made this talk
  - Look at angular material's implementations like: https://github.com/angular/components/blob/master/src/material/select/select.ts#L587
- *[(ngModel)]* is available on components implementing ControlValueAccessor

The End

+1 Angular skill

Thank you for growing

- https://medium.com/@majdasab/implementing-control-value-accessor-in-angular-1b89f2f84ebf
- https://angular.io/api/forms/ControlValueAccessor
- https://github.com/Blaarkies/angular-control-value-accessor