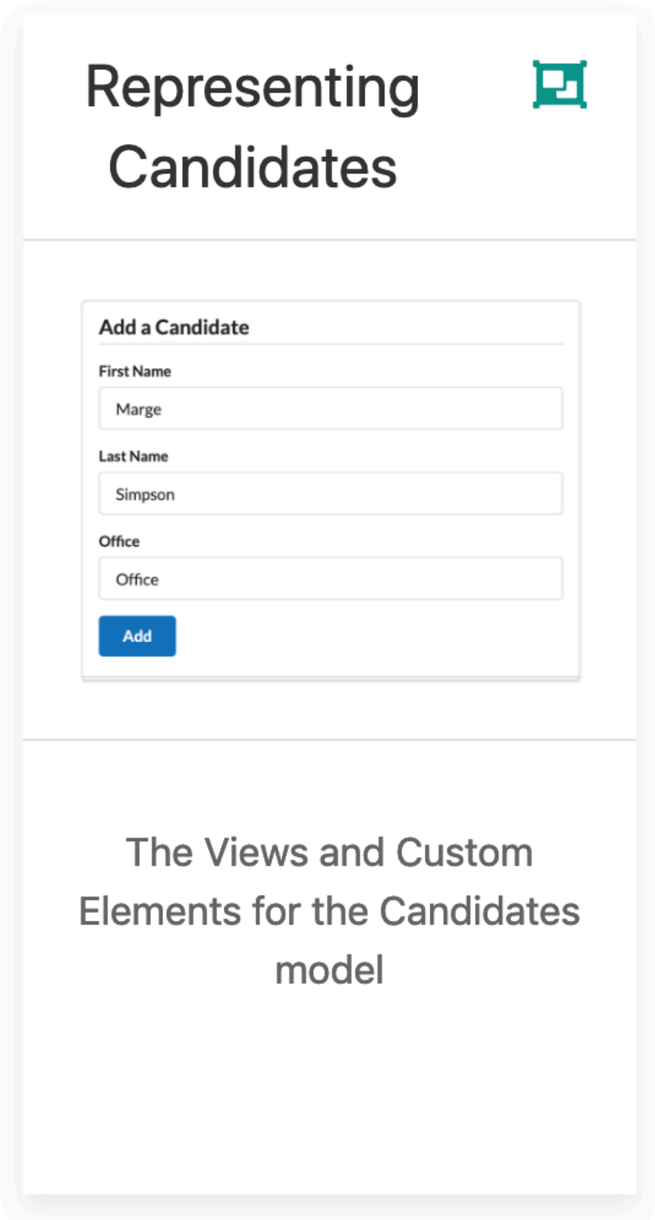
Representing Candidates



Typescript Interfaces

View Components

Custom Element Components

Typescript Interfaces

Add a Candidate

First Name

Marge

Last Name

Simpson

Office

Office

Add

Candidates

Last Name	First Name	Office
Simpson	Homer	Office
Simpson	Marge	Office

Interfaces in Typescript

```
export interface Candidate {
  firstName: string;
  lastName: string;
  office: string;
}
```

- Similar concept to Java equivalent
- Specifies ‘shape’ of valid candidate objects
- Typescript compiler will type check candidates against this specification

“One of TypeScript’s core principles is that type-checking focuses on the shape that values have. This is sometimes called “duck typing” or “structural subtyping”. In TypeScript, interfaces fill the role of naming these types, and are a powerful way of defining contracts within your code as well as contracts with code outside of your project.”

https://www.typescriptlang.org/docs/handbook/interfaces.html

Types in Typescript

```
export class App {
  firstName: string;
  lastName: string;
  office: string;
  candidates: any[] = [];

  addCandidate() {
    const candidate = {
      firstName: this.firstName,
      lastName: this.lastName,
      office: this.office
    };
    this.candidates.push(candidate);
    console.log(candidate);
  }
}
```

```
export interface Candidate {
  firstName: string;
  lastName: string;
  office: string;
}
```

```
export class App {
  firstName: string;
  lastName: string;
  office: string;
  candidates: Candidate[] = [];

  addCandidate() {
    const candidate = {
      firstName: this.firstName,
      lastName: this.lastName,
      office: this.office
    };
    this.candidates.push(candidate);
    console.log(candidate);
  }
}
```

- Types in Typescript

Type Checking

```
export interface Candidate {
  firstName: string;
  lastName: string;
  office: string;
}

export class App {
  firstName: string;
  lastName: string;
  office: string;
  candidates: Candidate[] = [];

  addCandidate() {
    const candidate = {
      firstName: this.firstName,
      lastName: this.lastName,
      office: this.office
    };
    this.candidates.push(candidate);
    console.log(candidate);
  }
}
```

Define Candidate Type

Declare candidates as an array of this type

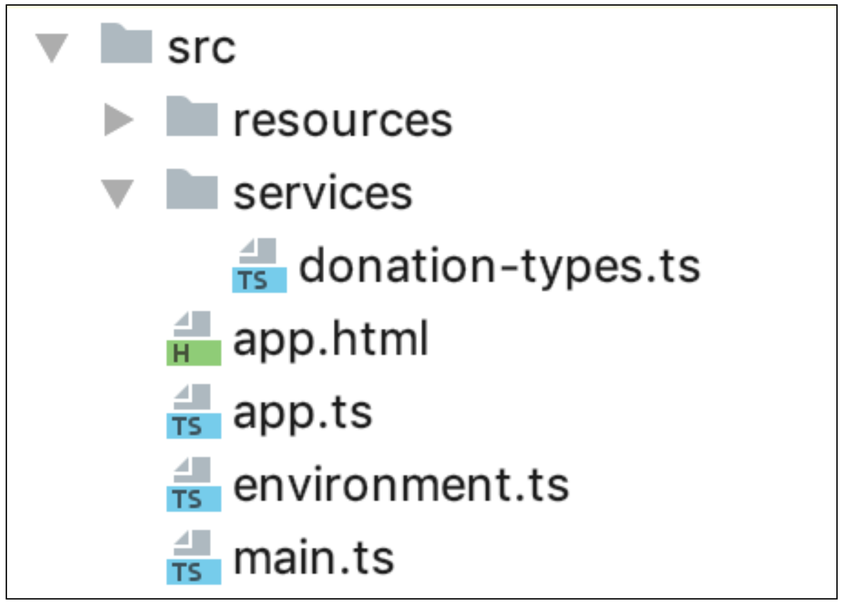
Create an object… and insert into array

The ‘push’ will be type checked, to verify that the object you are inserting matches the Candidate type

import / export

src/services/donation-types

```
export interface Candidate {
  firstName: string;
  lastName: string;
  office: string;
}
```



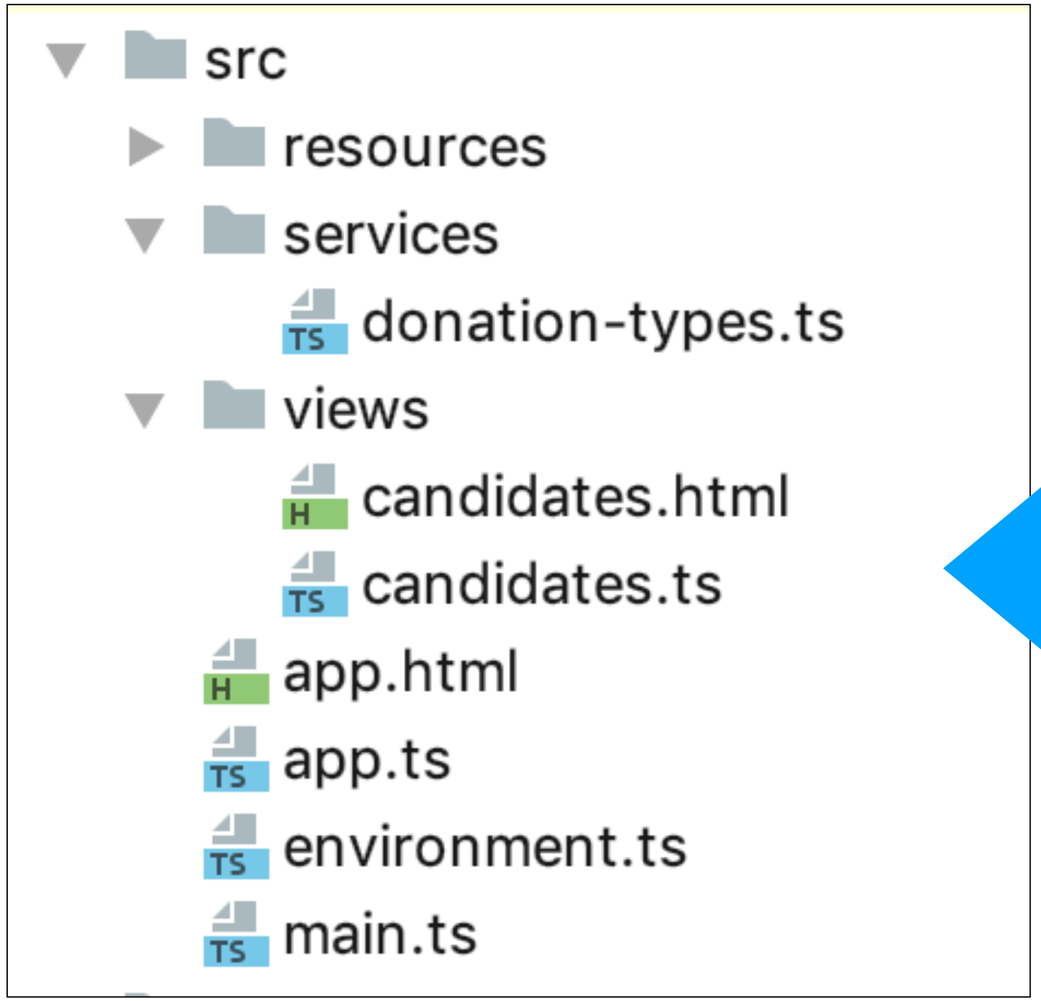
src/app.ts

```
import {Candidate} from "./services/donation-types";

export class App {
  firstName: string;
  lastName: string;
  office: string;
  candidates: Candidate[] = [];

  addCandidate() {
    const candidate = {
      firstName: this.firstName,
      lastName: this.lastName,
      office: this.office
    };
    this.candidates.push(candidate);
    console.log(candidate);
  }
}
```

View Components



View Component

- view folder:
  - reusable components
  - ts/html pairs
  - can be included in other components

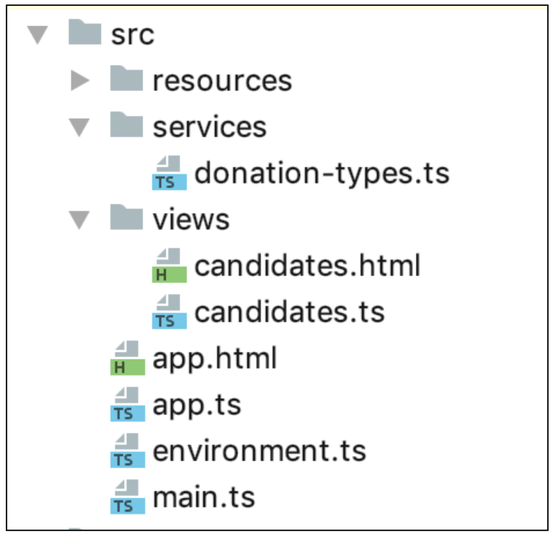
Candidate View Component

src/views/candidates.ts

```
import { Candidate } from '../services/donation-types';

export class Candidates {
  firstName: string;
  lastName: string;
  office: string;
  candidates: Candidate[] = [];

  addCandidate() {
    const candidate = {
      firstName: this.firstName,
      lastName: this.lastName,
      office: this.office
    };
    this.candidates.push(candidate);
    console.log(candidate);
  }
}
```



srs/views/candidates.html

```
<template>
  <div class="ui stackable two column grid">
    <div class="column">
      <form submit.trigger="addCandidate()" class="ui form stacked segment">
        <h3 class="ui dividing header"> Add a Candidate </h3>
        <div class="field">
          <label>First Name </label> <input value.bind="firstName">
        </div>
        <div class="field">
          <label>Last Name </label> <input value.bind="lastName">
        </div>
        <div class="field">
          <label>Office </label> <input value.bind="office">
        </div>
        <button class="ui blue submit button">Add</button>
      </form>
    </div>
    <div class="column">
      <h4 class="ui dividing header"> Candidates </h4>
      <table class="ui celled table segment">
        <thead>
          <tr>
            <th>Last Name</th>
            <th>First Name</th>
            <th>Office</th>
          </tr>
        </thead>
        <tbody>
          <tr repeat.for="candidate of candidates">
            <td>
              ${candidate.lastName}
            <td>
              ${candidate.firstName}
            </td>
            <td>
              ${candidate.office}
            </td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
</template>
```

src/app.ts

```
export class App {
}
```

src/app.html

```
<template>
  <require from="views/candidates"></require>

  <div class="ui container">
    <section class="ui raised segment">
      <h3 class="ui headder"> Donation </h3>
    </section>
    <div class="ui basic segment">
      <candidates></candidates>
    </div>
  </div>
</template>
```

import candidates

place candidates component

Using a View Component

- src
  - resources
  - services
    - donation-types.ts
  - views
    - candidates.html
    - candidates.ts
  - app.html
  - app.ts
  - environment.ts
  - main.ts

Add a Candidate

First Name

Marge

Last Name

Simpson

Office

Office

Add

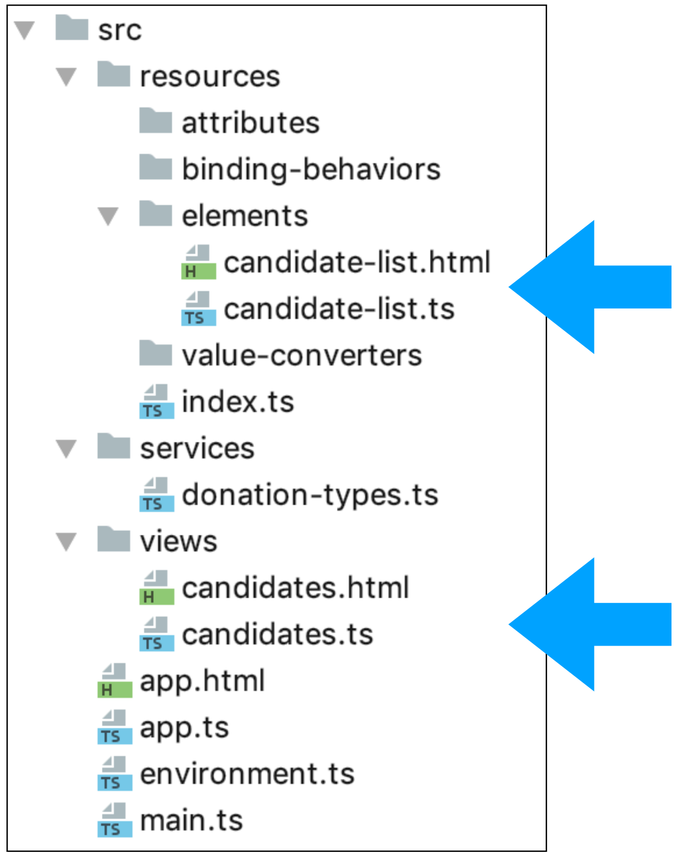
Candidates

Last Name	First Name	Office
Simpson	Homer	Office
Simpson	Marge	Office

Custom Element Components

Custom Elements Component

- Custom Elements are another type of component
- More reusable than View components
- Data in Custom Element can be ‘bound’ to other components



Custom Elements Component generally placed in ‘resource/elements’ folder

View Components generally placed in ‘views’ folder (sometimes also called ‘pages’ folder)

candidate-list Custom Element

src/resources/elements/candidate-list.ts

```
import { bindable } from 'aurelia-framework';
import { Candidate } from '../../services/donation-types';

export class CandidateList {
  @bindable
  candidates: Candidate[];
}
```

- candidates array ‘@bindable’

- This means the array is defined in another component

Candidates

Last Name	First Name	Office
Simpson	Homer	Office
Simpson	Marge	Office

src/resources/elements/candidate-list.html

```
<template>
  <h4 class="ui dividing header"> Candidates </h4>
  <table class="ui celled table segment">
    <thead>
      <tr>
        <th>last Name</th>
        <th>First Name</th>
        <th>Office</th>
      </tr>
    </thead>
    <tbody>
      <tr repeat.for="candidate of candidates">
        <td>
          ${candidate.lastName}
        <td>
          ${candidate.firstName}
        </td>
        <td>
          ${candidate.office}
        </td>
      </tr>
    </tbody>
  </table>
</template>
```

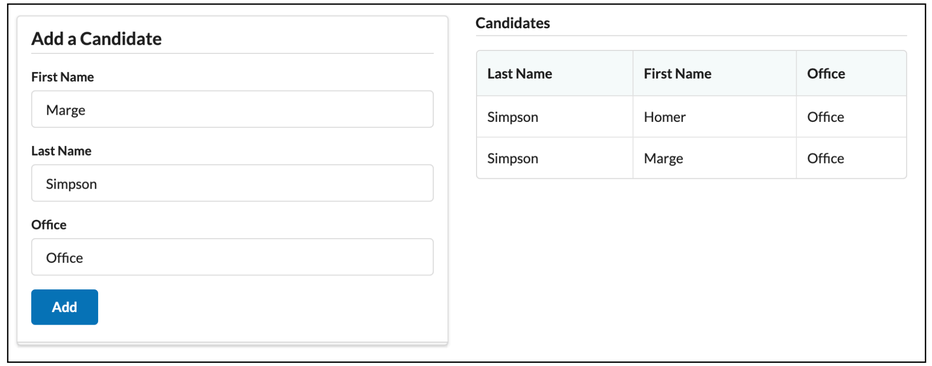
Importing & using a Custom Element

src/views/candidates.html

```
<template>
  <require from="../resources/elements/candidate-list"></require>

  <div class="ui stackable two column grid">
    <div class="column">
      <form submit.trigger="addCandidate()" class="ui form stacked segment">
        <h3 class="ui dividing header"> Add a Candidate </h3>
        <div class="field">
          <label>First Name </label> <input value.bind="firstName">
        </div>
        <div class="field">
          <label>Last Name </label> <input value.bind="lastName">
        </div>
        <div class="field">
          <label>Office </label> <input value.bind="office">
        </div>
        <button class="ui blue submit button">Add</button>
      </form>
    </div>
    <div class="column">
      <candidate-list candidates.bind="candidates"></candidate-list>
    </div>
  </div>
</template>
```

Import the candidate-list custom element



Place the component in the second column

Custom Element Binding

src/views/candidates.ts

```
import { Candidate } from '../services/donation-types';

export class Candidates {
  firstName: string;
  lastName: string;
  office: string;
  candidates: Candidate[] = [];

  addCandidate() {
    const candidate = {
      firstName: this.firstName,
      lastName: this.lastName,
      office: this.office
    };
    this.candidates.push(candidate);
    console.log(candidate);
  }
}
```

Candidates array defined here

'Binding' from this array to the custom element

```
<div class="column">
  <candidate-list candidates.bind="candidates"></candidate-list>
</div>
```

src/resources/elements/candidate-list.ts

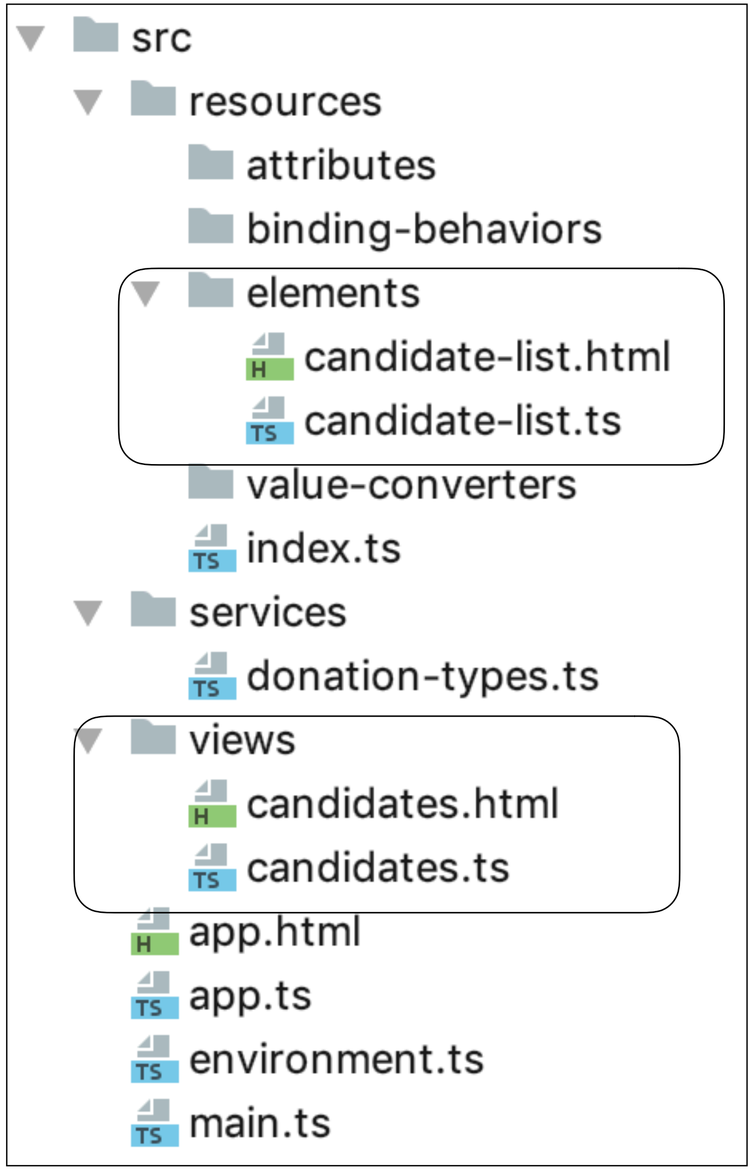
```
import { bindable } from 'aurelia-framework';
import { Candidate } from '../../services/donation-types';

export class CandidateList {
  @bindable
  candidates: Candidate[];
}
```

```
<tr repeat.for="candidate of candidates">
  <td>
    ${candidate.lastName}
  <td>
    ${candidate.firstName}
  </td>
  <td>
    ${candidate.office}
  </td>
</tr>
```

candidates array here is bound to view

candidates array here is now refers to the array defined in candidates.ts



<u>Custom Elements Components - *Fine Grained*</u>

Reusable components - can appear in many different views

<u>View Components - *Course Grained*</u>

Represent complete rendered page - composed of multiple Custom Elements