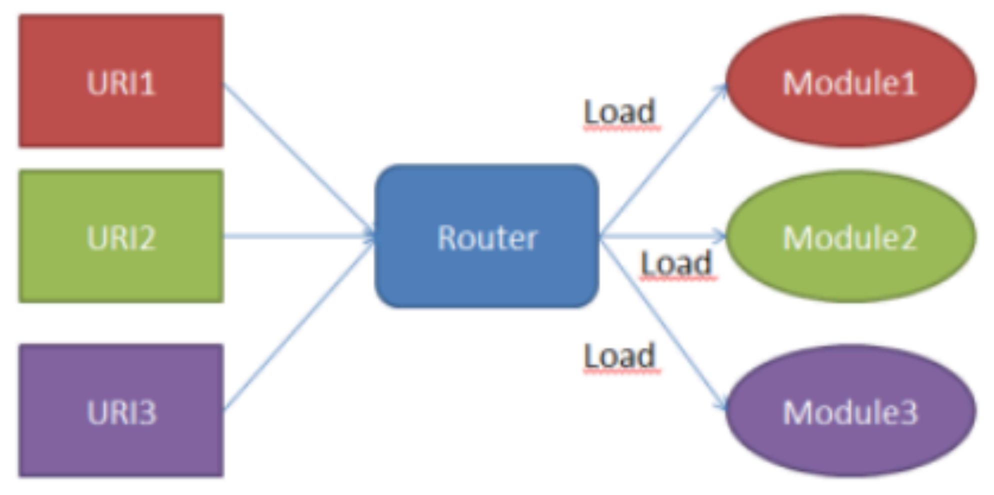
Aurelia Routers

Routers



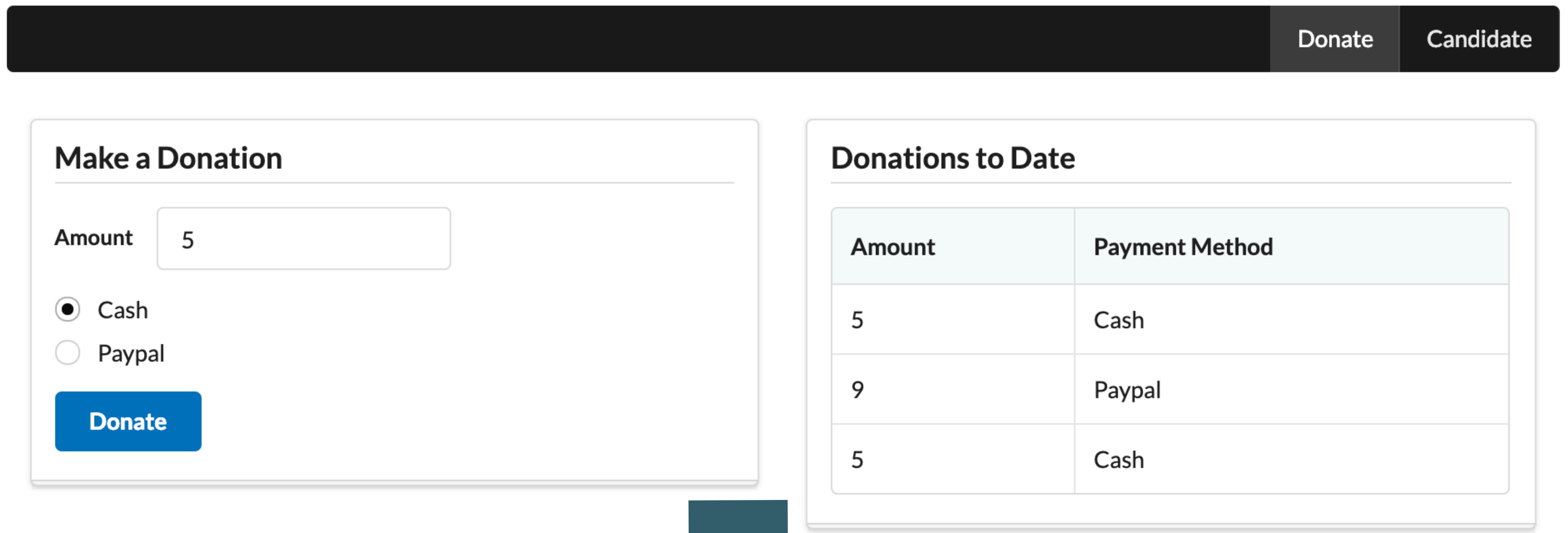
Client side routing in Aurelia

Agenda

- The Back Button
- Fragment Identifiers
- Hash Based Routing
- Routing in Aurelia

The Back Button

- For an SPA, the Back button can have the effect of terminating the app
- Back goes to the previous page - but the user may be already several screens into the app.
- The previous page may not concur with the users perception



Make a Donation

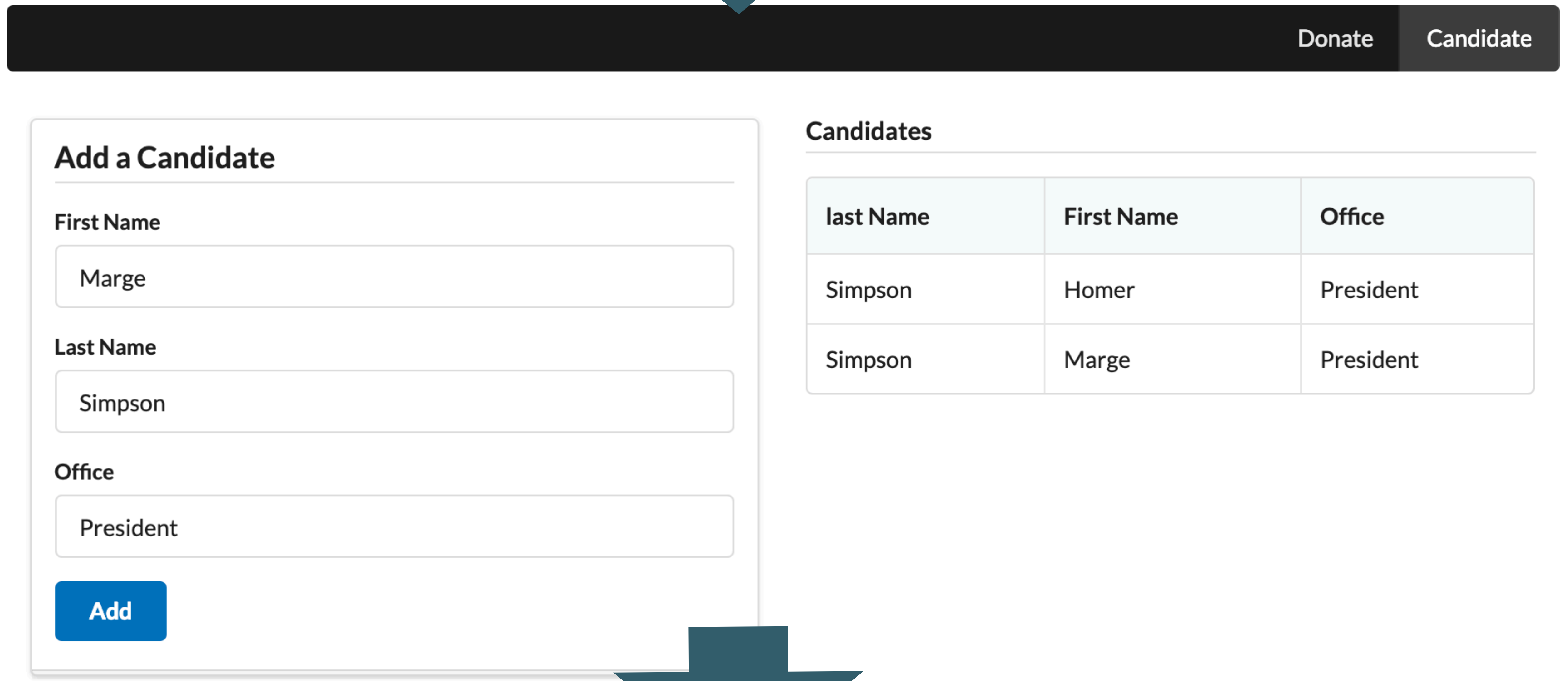
Amount 5

● Cash
○ Paypal

Donate

Donations to Date

Amount	Payment Method
5	Cash
9	Paypal
5	Cash



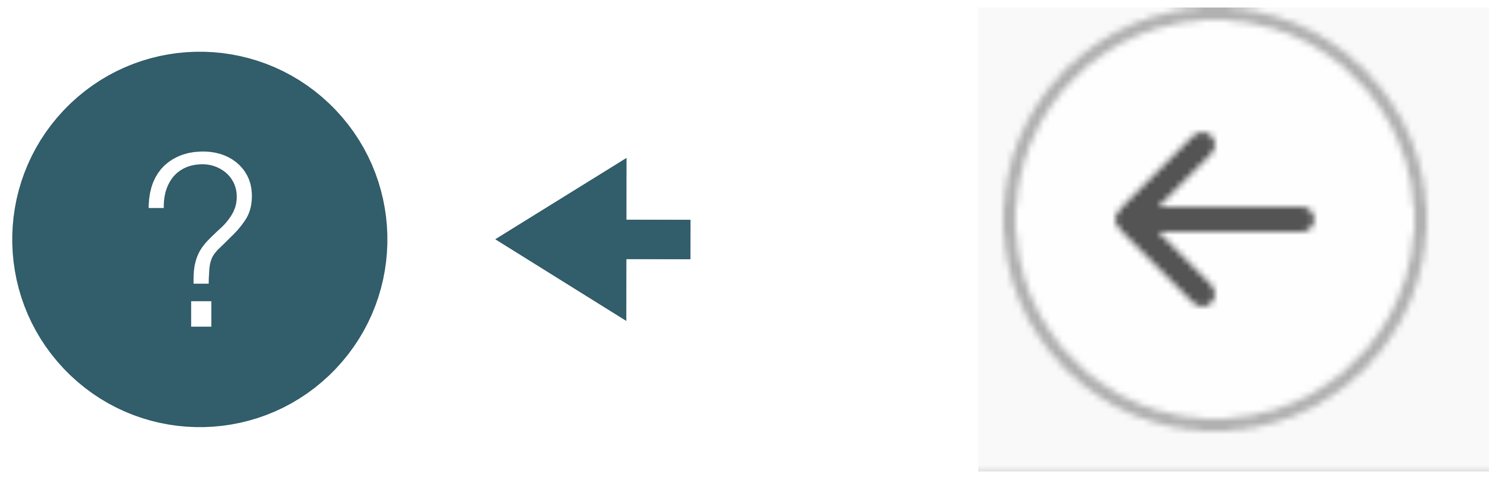
Add a Candidate

First Name	Marge
Last Name	Simpson
Office	President

Add

Candidates

last Name	First Name	Office
Simpson	Homer	President
Simpson	Marge	President



Fragment Identifiers:

- In URIs a hashmark # introduces the optional fragment near the end of the URL
- The fragment identifier functions differently than the rest of the URI:
  - its processing is exclusively client-side with no participation from the web server
  - Eg:

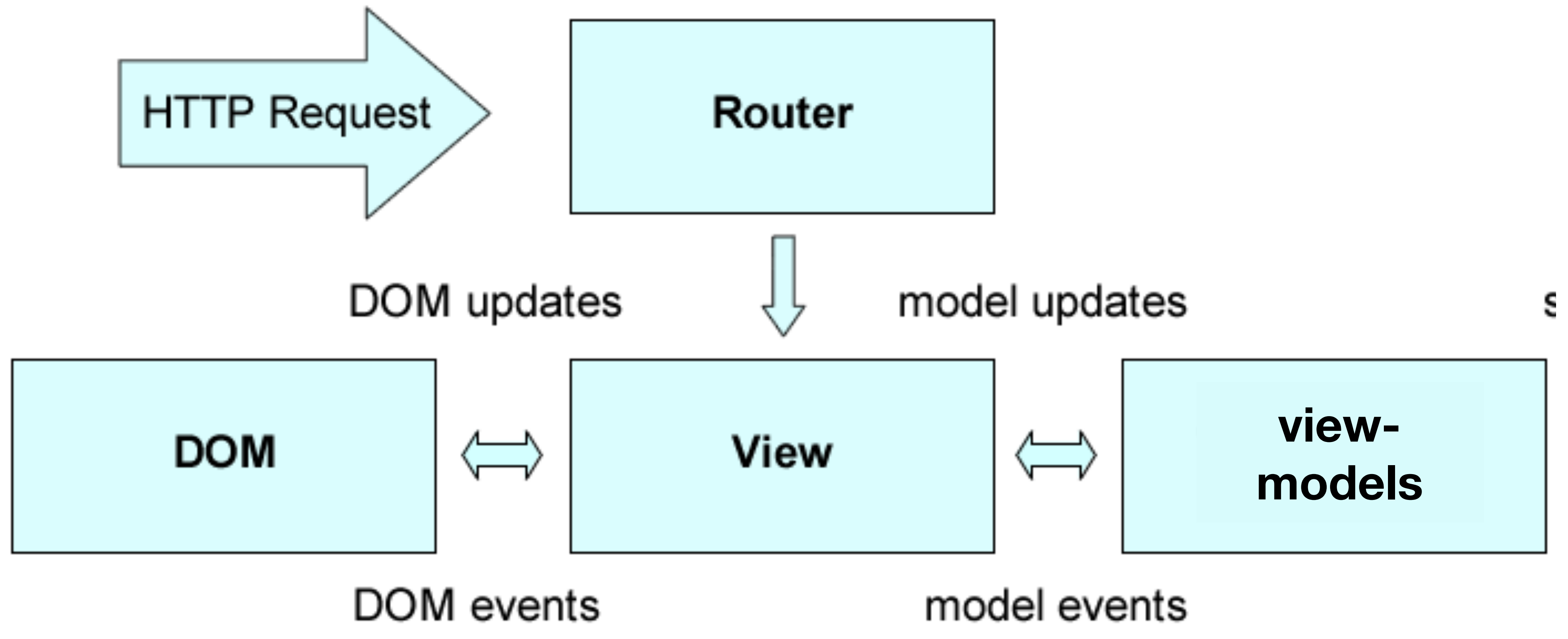
https://en.wikipedia.org/wiki/Fragment_identifier#References

Hash-based routing

- Use the fragment part of the URL to simulate different content.
  - For example http://site.com/#/products/list leads to displaying a list of products.
  - “#/products/list” is never sent to the server and is completely processed on the client side
- Javascript in the client can collaborate with the browser to maintain the illusion of page navigation
- Requires ‘pushstate’ api, which permits Javascript to explicitly manipulate the address bar, specifically back button interventions.

https://developer.mozilla.org/en-US/docs/Web/API/History_API

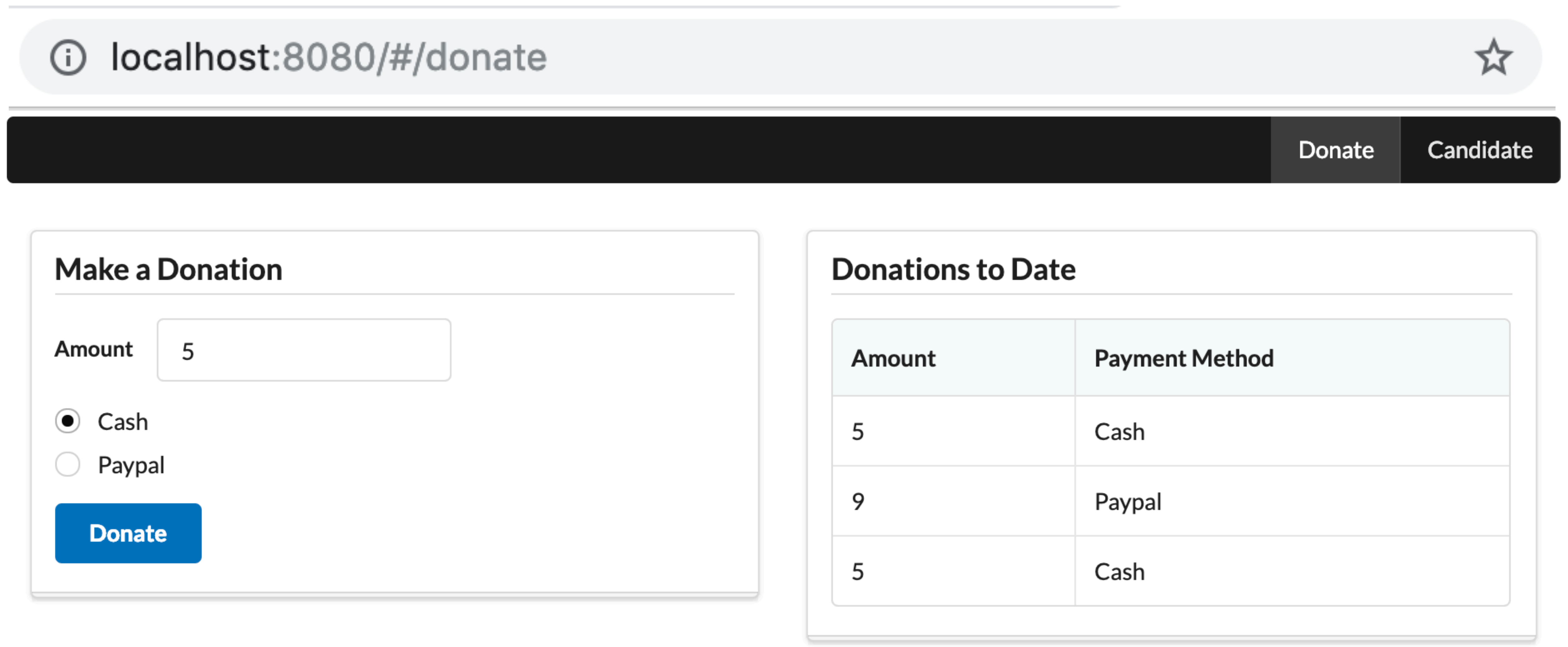
Routing in Aurelia



- Aurelia, link most SPA frameworks, provides a comprehensive client side routing support.
- This mirrors the type of routing familiar from server side development
- As the user navigates a client side app, the routes are inserted into the address bar (after ‘#’ character).
- Back button rolls back to a previous route.

- Forward/Back button navigates as expected between views

- Address bar explicitly indicates which “View” is current



```
localhost:8080/#/donate
```

Donate | Candidate

Make a Donation

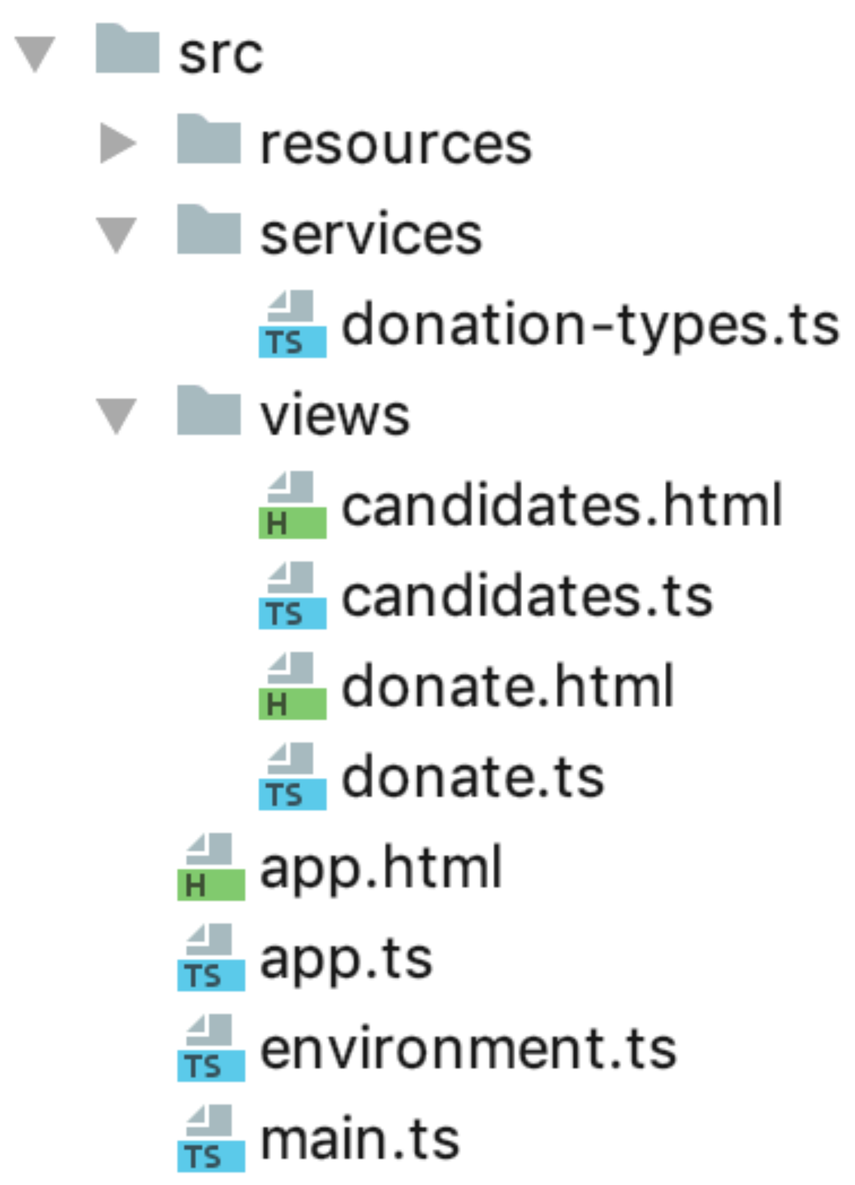
Amount	5

● Cash
○ Paypal

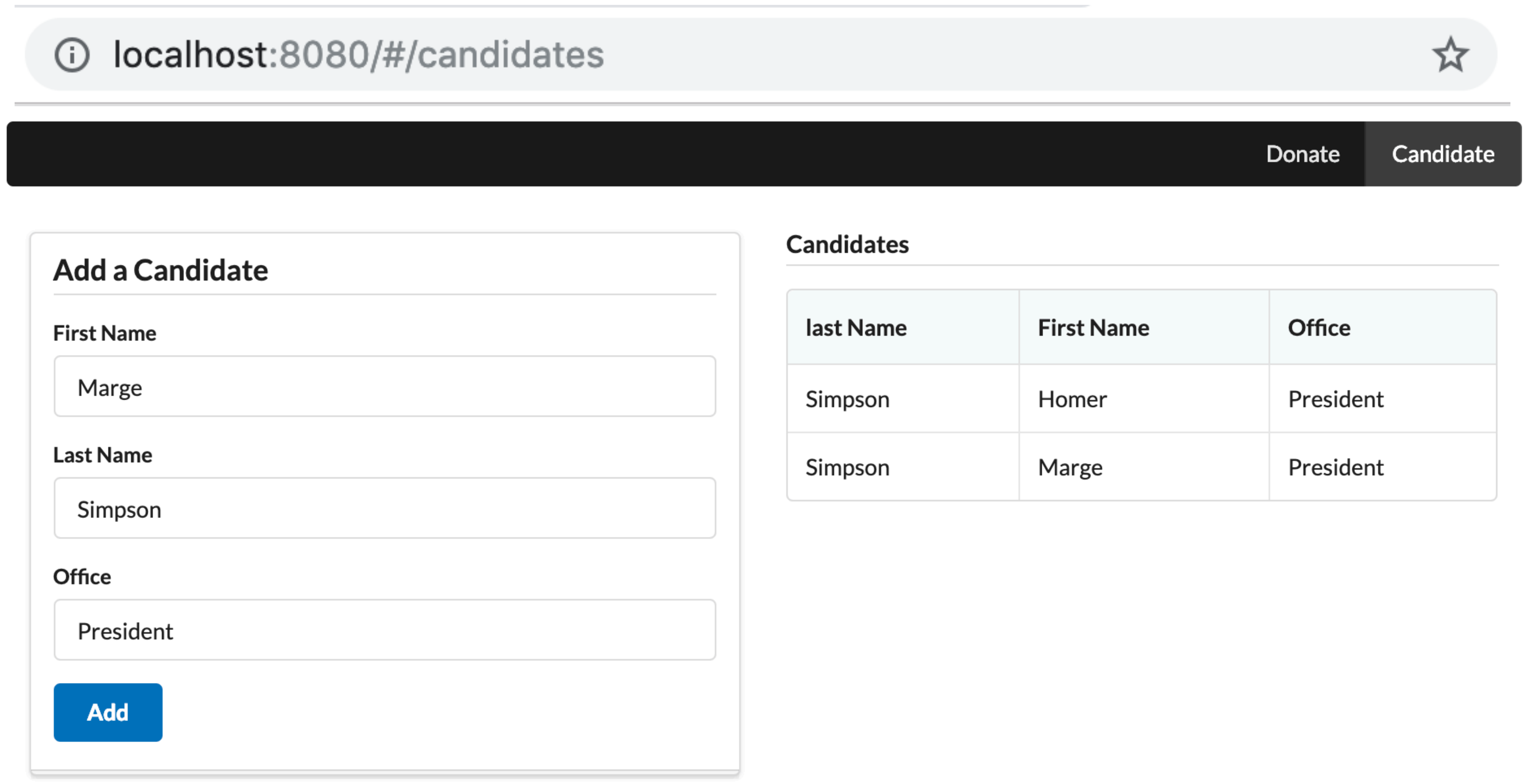
Donate

Donations to Date

Amount	Payment Method
5	Cash
9	Paypal
5	Cash



```
▼ src
  ▶ resources
  ▼ services
      donation-types.ts
  ▼ views
      candidates.html
      candidates.ts
      donate.html
      donate.ts
    app.html
    app.ts
    environment.ts
    main.ts
```



```
localhost:8080/#/candidates
```

Donate | Candidate

Add a Candidate

First Name	Marge
Last Name	Simpson
Office	President

Add

Candidates

last Name	First Name	Office
Simpson	Homer	President
Simpson	Marge	President

Configuring a Router: app.html

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

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

…with generic **<router-view>**

Replace explicit inclusion of **<donate>** & **<candidates>** custom elements

```html
<template>
  …
  <div class="ui container">
    <section class="ui raised segment">
      <h3 class="ui headder"> Donation </h3>
    </section>
    <nav-bar router.bind="router"></nav-bar>
    <div class="ui basic segment">
      <router-view></router-view>
    </div>
  </div>
</template>
```

Configuring a Router: app.ts

```
export class App {
}
```

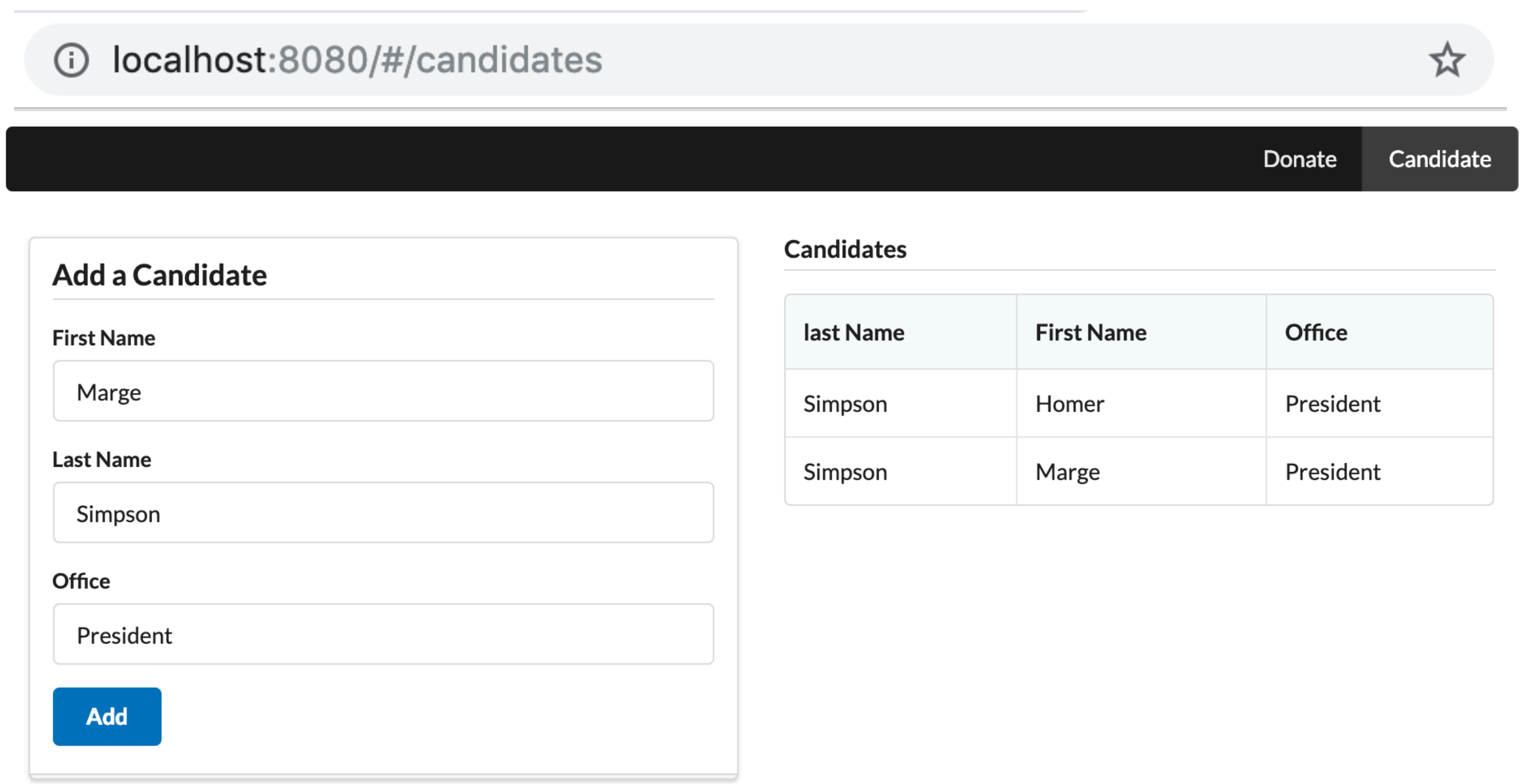
Define route pattern and associate with View Components

```
import { RouterConfiguration, Router } from 'aurelia-router';
import { PLATFORM } from 'aurelia-pal';

export class App {
  router: Router;

  configureRouter(config: RouterConfiguration, router: Router) {
    config.map([
      {
        route: ['', 'donate'],
        name: 'Donate',
        moduleId: PLATFORM.moduleName('views/donate'),
        nav: true,
        title: 'Donate'
      },
      {
        route: 'candidates',
        name: 'candidates',
        moduleId: PLATFORM.moduleName('views/candidates'),
        nav: true,
        title: 'Candidate'
      }
    ]);
    this.router = router;
  }
}
```

candidates route

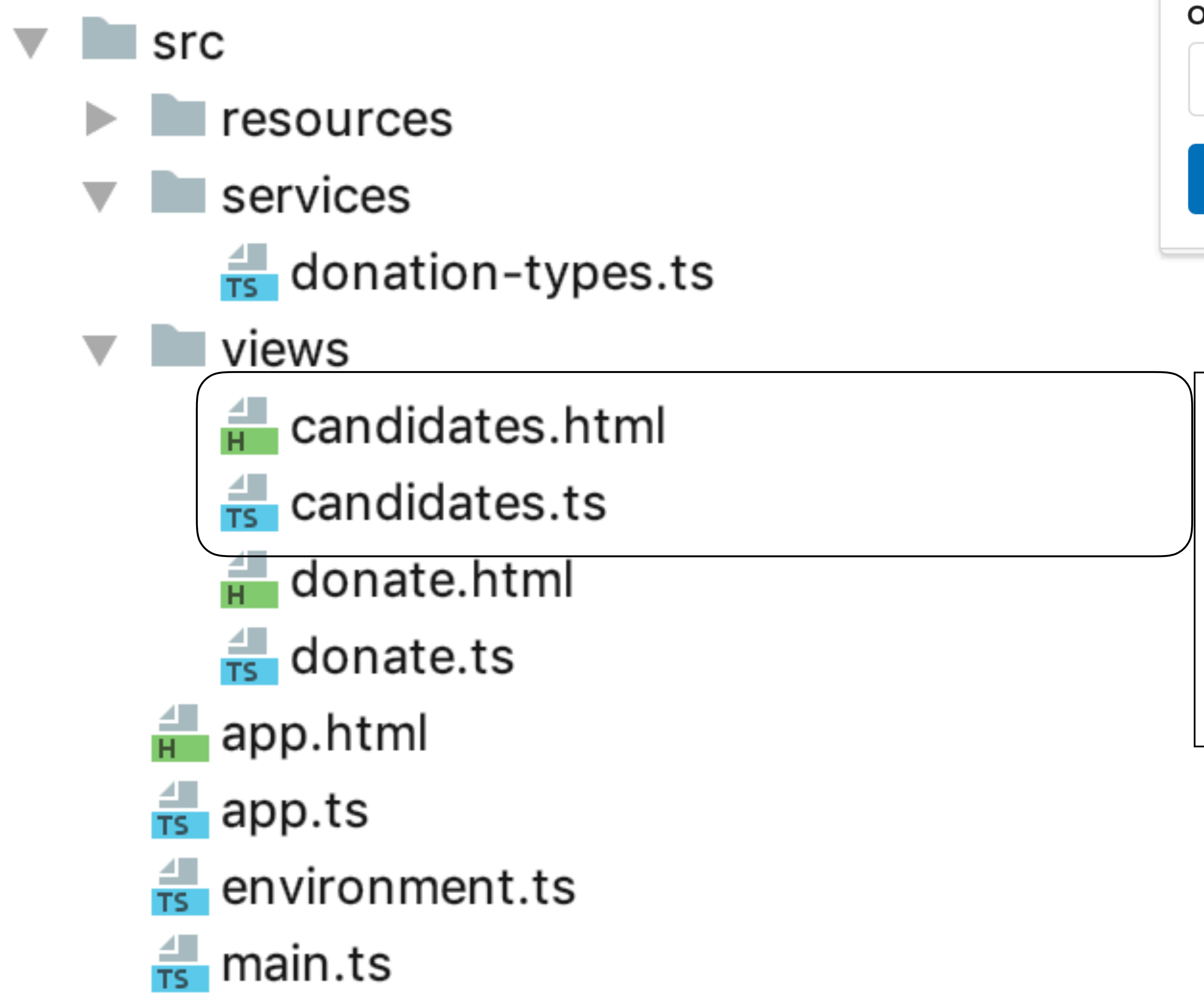

localhost:8080/#/candidates

Add a Candidate

First Name	Marge
Last Name	Simpson
Office	President

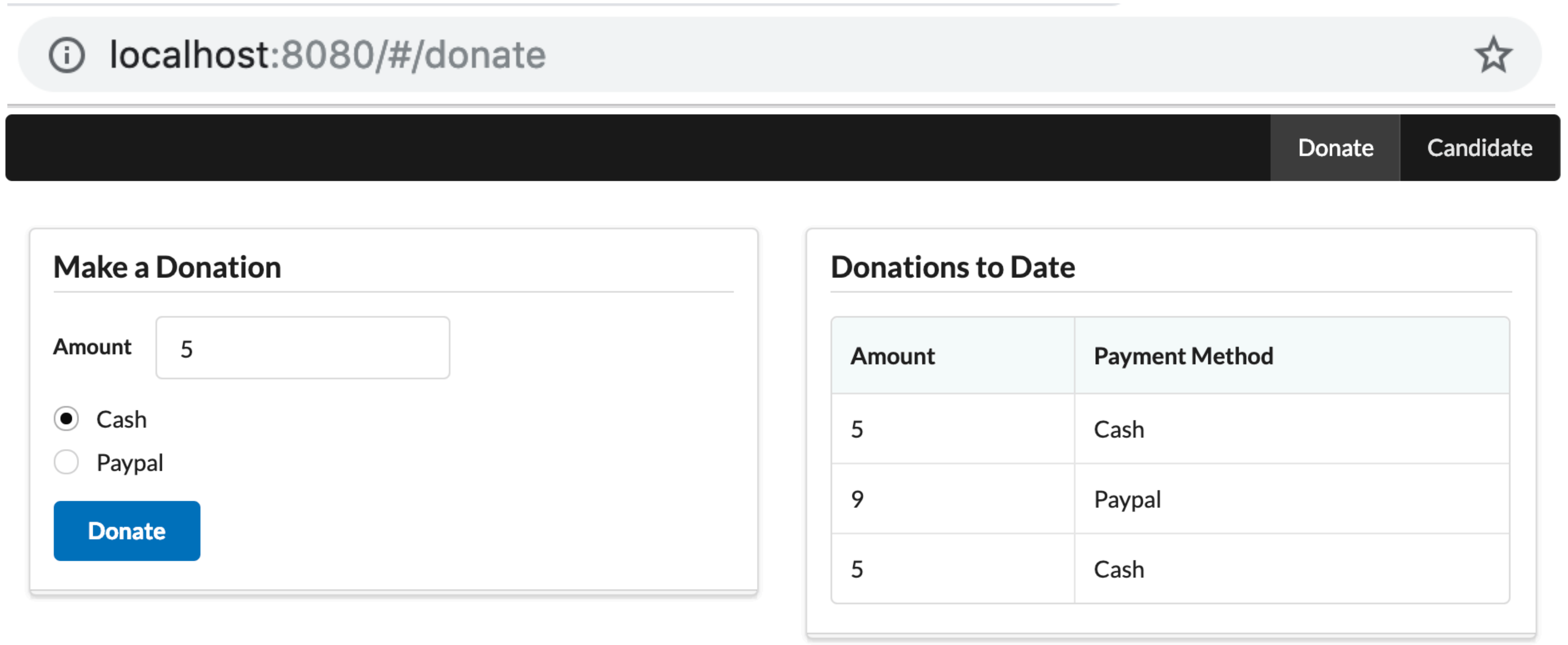
Candidates

last Name	First Name	Office
Simpson	Homer	President
Simpson	Marge	President


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

```
{
  route: 'candidates',
  name: 'candidates',
  moduleId: PLATFORM.moduleName('views/candidates'),
  nav: true,
  title: 'Candidate'
}
```

donate route



localhost:8080/#/donate

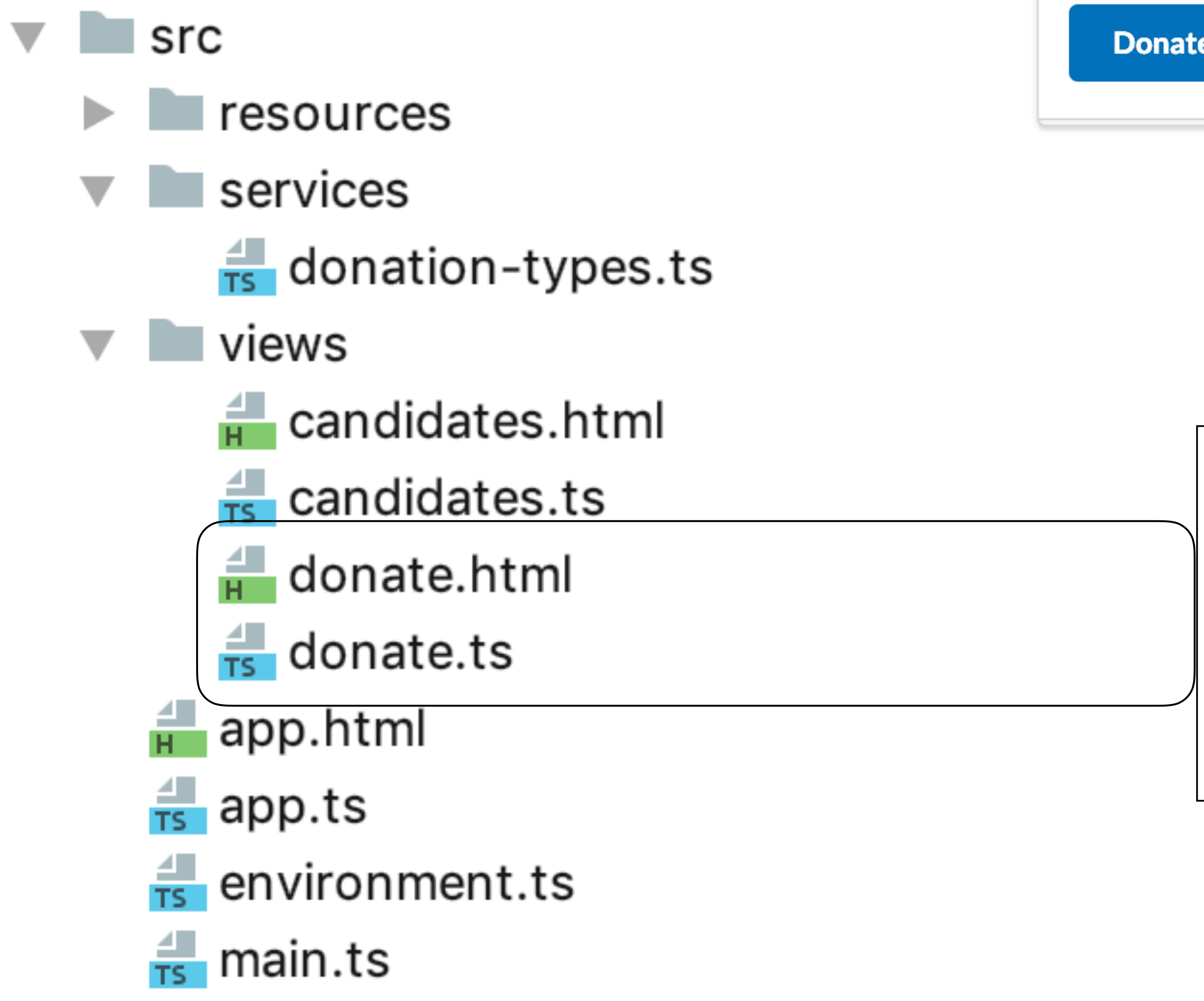
Make a Donation

Amount: 5

[✓ Cash]
Paypal

Donations to Date

Amount	Payment Method
5	Cash
9	Paypal
5	Cash



```
▼ src
  ▶ resources
  ▼ services
      donation-types.ts
  ▼ views
      candidates.html
      candidates.ts
      donate.html
      donate.ts
    app.html
    app.ts
    environment.ts
    main.ts
```

```
      {
        route: ['', 'donate'],
        name: 'Donate',
        moduleId: PLATFORM.moduleName('views/donate'),
        nav: true,
        title: 'Donate'
      }
```

nav-bar.html

```
<template bindable="router">
  <nav class="ui inverted menu">
    <div class="right menu">
      <div repeat.for="row of router.navigation">
        <a class="${row.isActive ? 'active' : ''} item"  href.bind="row.href">${row.title}</a>
      </div>
    </div>
  </nav>
</template>
```

import the nav-bar

app.html

```
<template>
  <require from=“resources/elements/nav-bar.html"></require>

  <div class="ui container">
    <nav-bar router.bind="router"></nav-bar>
    <div class="ui basic segment">
      <router-view></router-view>
    </div>
  </div>
</template>
```

render and bind the nav-bar

placeholder for loaded view-models

iterate through the router array

nav-bar.html

```
<template bindable="router">
  <nav class="ui inverted menu">
    <div class="right menu">
      <div repeat.for="row of router.navigation">
        <a class="${row.isActive ? 'active' : ''} item"  href.bind="row.href">${row.title}</a>
      </div>
    </div>
  </nav>
</template>
```

render a single menu item in the router array

```
config.map([
  {
    route: ['', 'donate'],
      ne: 'Donate',
      uleId: PLATFORM.moduleName('views/donate'),
    na   true,
    title   'Donate'
  },
  {
    route: 'candidates',
    name: 'candidates',
    moduleId: PLATFORM.moduleName('views/candidates'),
    nav: true,
    title: 'Candidate'
  }
]);
```

the #path (fragment)

these are the items - including title, navigable, active etc…

Route Attributes

- **route** (required) Either a string or an array of multiple routes (which are strings). If the value supplied is '' then it will be used as the default route. Additional route values can be supplied.

- **name** (required) A unique identifying name for our route. This allows you to reference the route, by generating links or navigating directly to it.

- **moduleId** This is the view-model that you want to be loaded when the route is hit. In the case of our welcome route, it is saying that you want to load src/welcome.js whenever the welcome route is loaded.

- **nav** A boolean value indicating whether or not this route is to be included in the navigation routes array. If this value is true, when iterating over the nav routes array, you'll be able to see it and construct it into a menu. If it is false, then it won't be seen in the navigation routes array.

- **title** The page title value when this route is hit. It will append itself to the provided title value supplied inside of config.title (if supplied).

```
config.map([
  {
    route: ['', 'donate'],
    name: 'Donate',
    moduleId: PLATFORM.moduleName('views/donate'),
    nav: true,
    title: 'Donate'
  },
```