Aurelia First Steps

Aurelia

“Aurelia is a JavaScript client framework for mobile, desktop and web leveraging simple conventions and empowering creativity”

Technical Principles:

- Forward-thinking
- Two-Way Databinding
- Routing & UI Composition
- Broad Language Support
- Modern Architecture
- Extensible HTML
- MV* with Conventions
- Testable

The Aurelia CLI

The Aurelia CLI Edit

by Rob Eisenberg

Learn how to get setup with the Aurelia CLI and use its basic capabilities.

Keywords: Getting Started, ES2015, ES2016, TypeScript

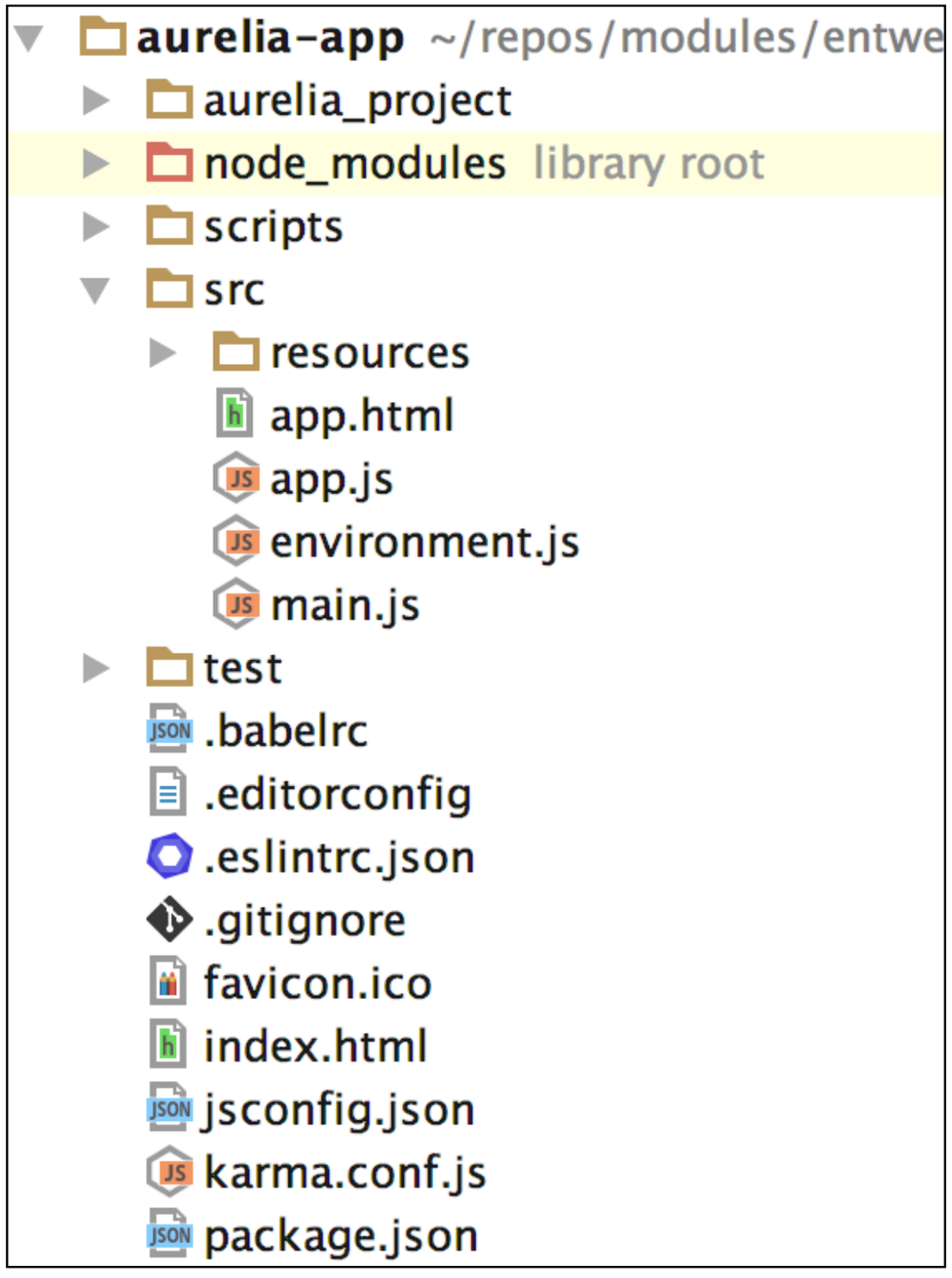
- A simple command line program to create and run aurelia apps

```
npm install aurelia-cli -g
```

New Command

```
au new donation-client
```

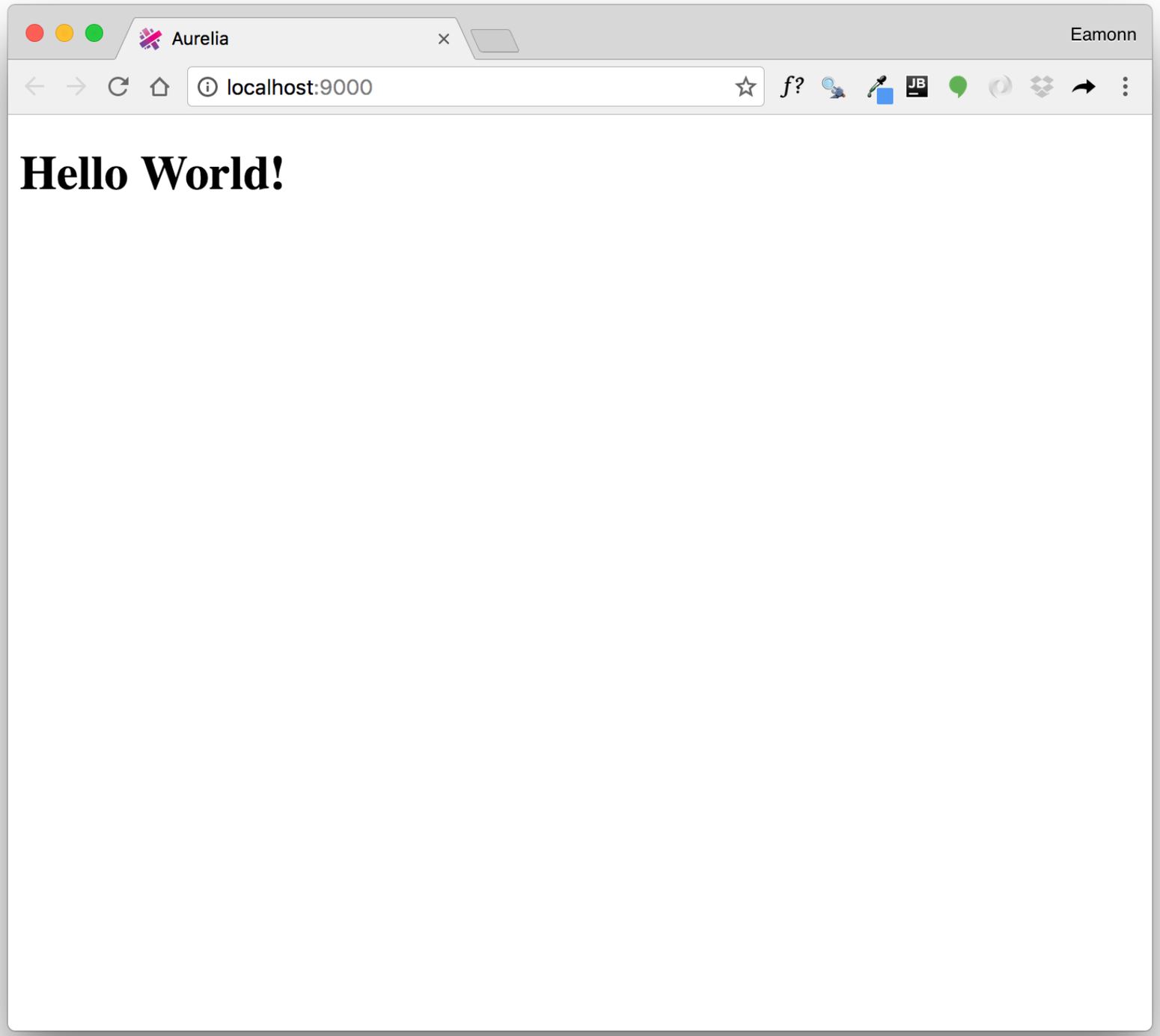
- Creates a complete starter application.
- Extensive scaffolding and support infrastructure
- Developer only needs to focus on:
  - **src** folder - containing the application sources
  - index.html - the start page for the app
- Remaining files + folders are build files, scaffolding or generated by aurelia



Run Command

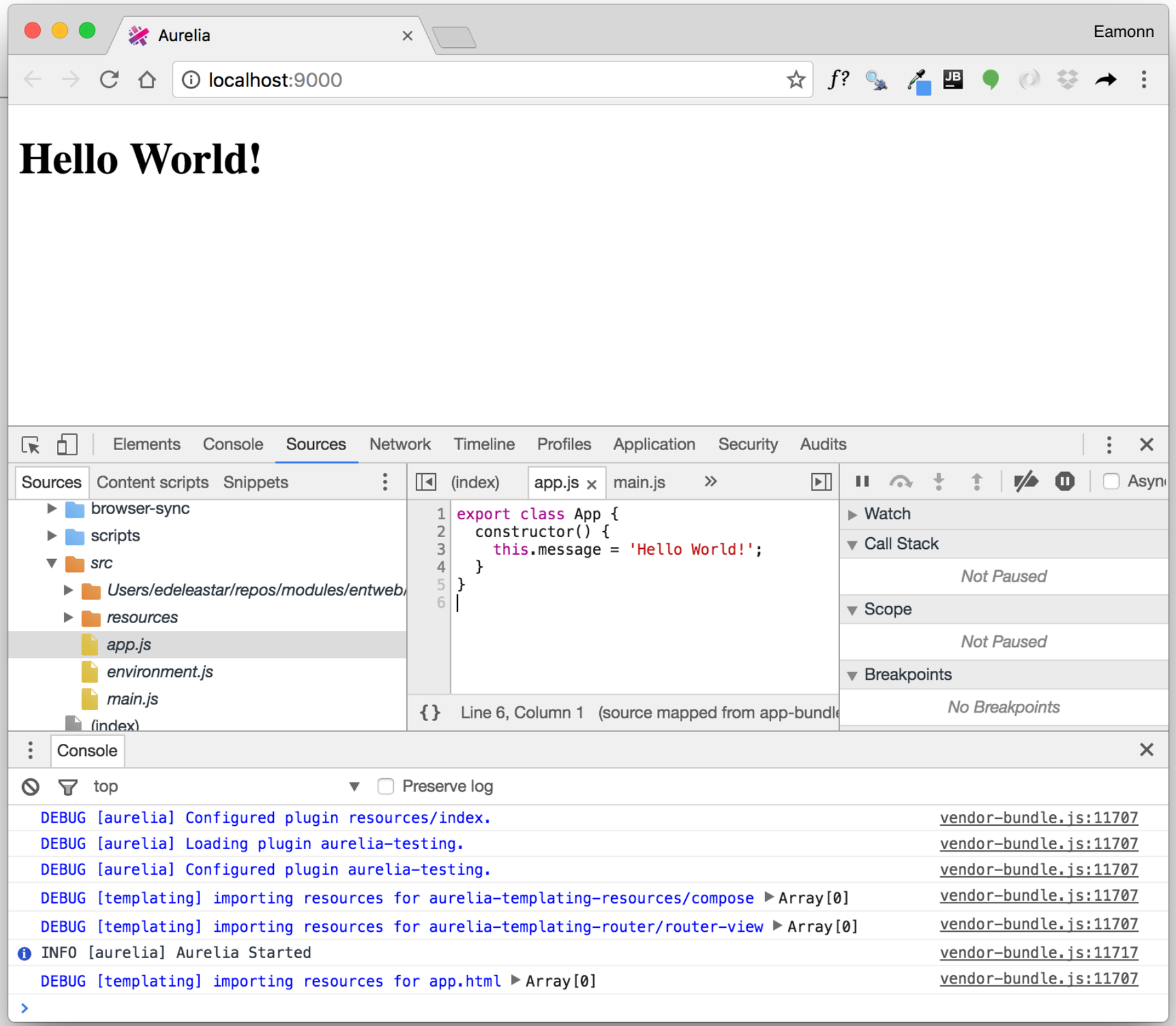
```
au run --watch
```

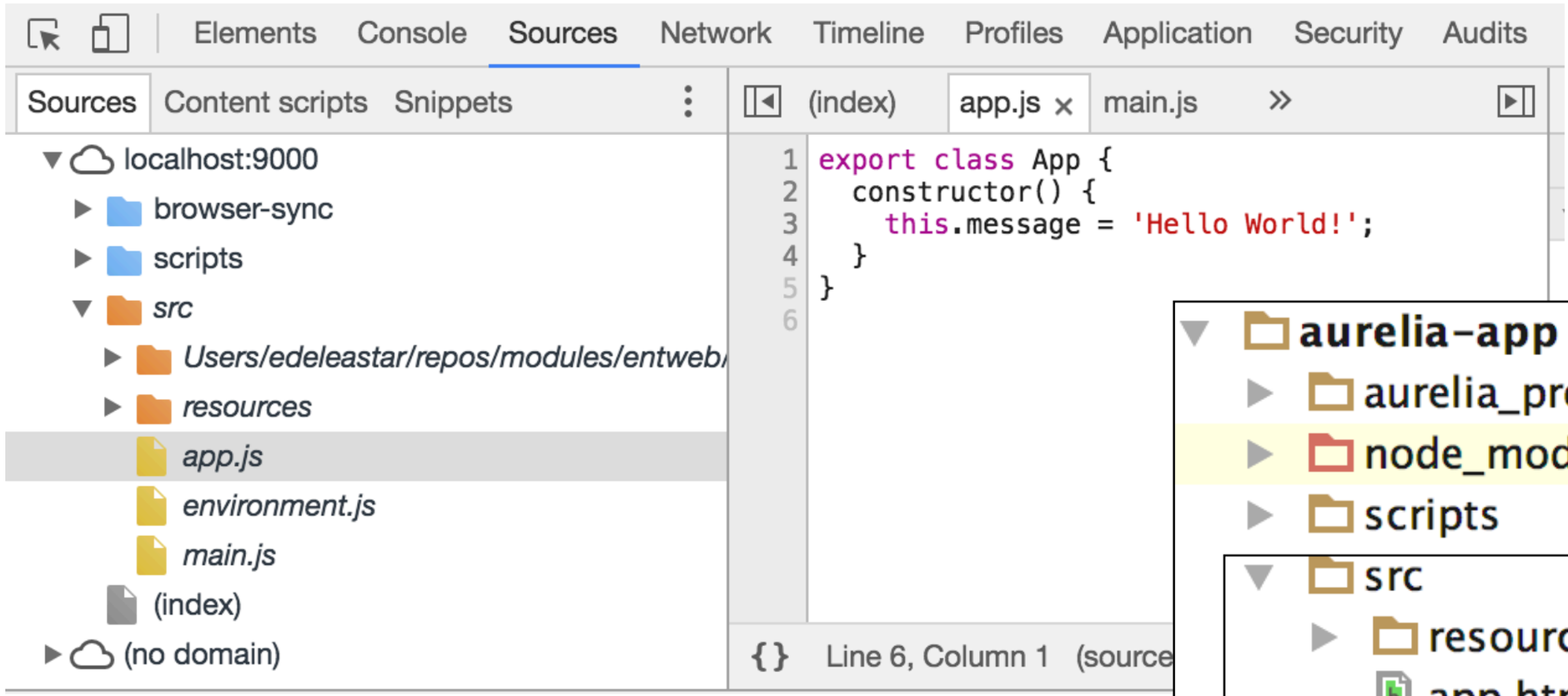
- Launches the app.
- Browsable on http://localhost:9000/
- Any changes to source will refresh app

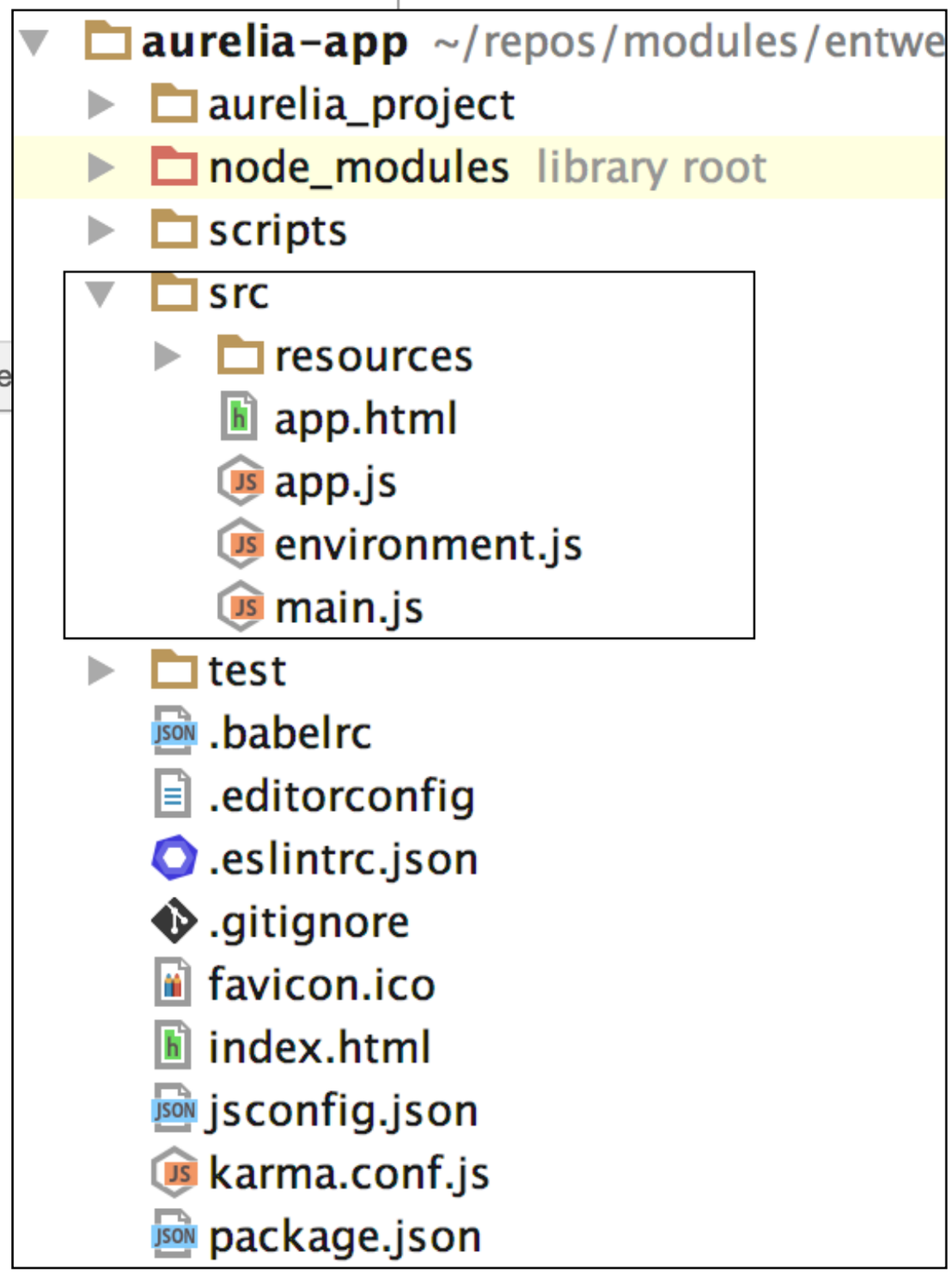


Debug

- App can be debugged in browser:
  - breakpoints
  - single step
  - variable inspection
  - console messages



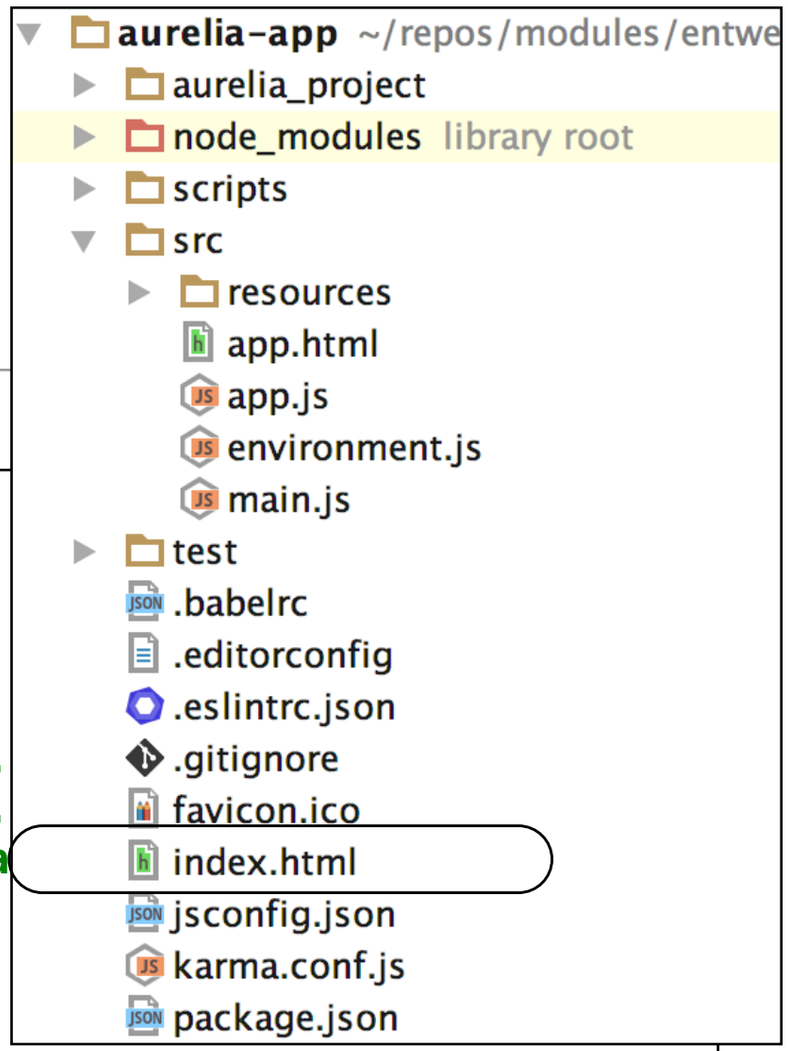




- src is a mapping of the project folder
- all Javascript files will be mapped here

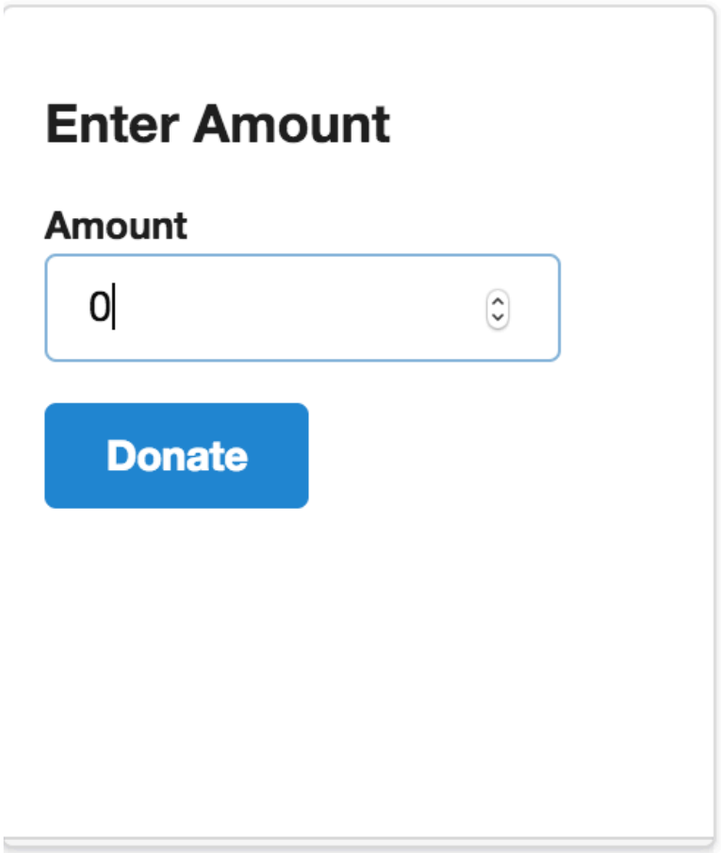
index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Donation</title>
    <meta charset="UTF-8">
    <script type="text/javascript" src="https://cdnjs.cloudflare.
    <script type="text/javascript" src="https://cdnjs.cloudflare.
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/aja

  </head>

  <body aurelia-app="main">
    <script src="scripts/vendor-bundle.js" data-main="aurelia-bootstrapper"></script>
  </body>
</html>
```



- The main entry point for application.
- Include main stylesheet here - but not further modification necessary
- Entire application is generated by aurelia build process into **scripts** folder and automatically loaded

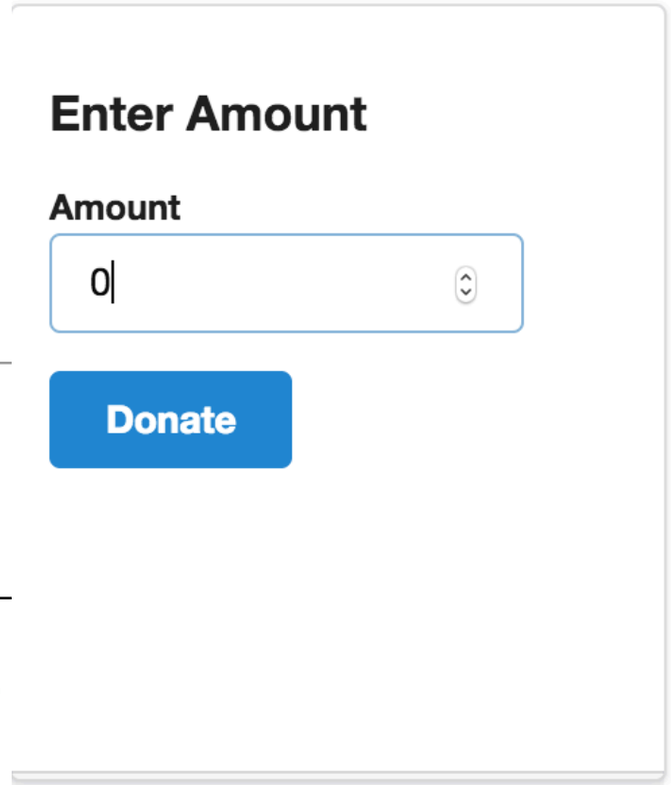
A view




```
<template>
  <form submit.trigger="makeDonation()" class="ui form four ...”>
    <div class="grouped inline fields">
      <h3>Enter Amount </h3>
      <div class="field">
        <label>Amount</label> <input type="number" value.bind="amount">
      </div>
    </div>
    <button class="ui blue submit button">Donate</button>
  </form>
</template>
```

- Encapsulated in an html <template>
- A standard form in this instance

Binding - Html side




```
<template>
  <form submit.trigger="makeDonation()" class="ui form four
    <div class="grouped inline fields">
      <h3>Enter Amount </h3>
      <div class="field">
        <label>Amount</label> <input type="number" value.bind="amount">
      </div>
    </div>
    <button class="ui blue submit button">Donate</button>
  </form>
</template>
```

- *makeDonation()* and *amount* are ***bound*** to a javascript object
- Which object is not specified in the template - it is determined by context

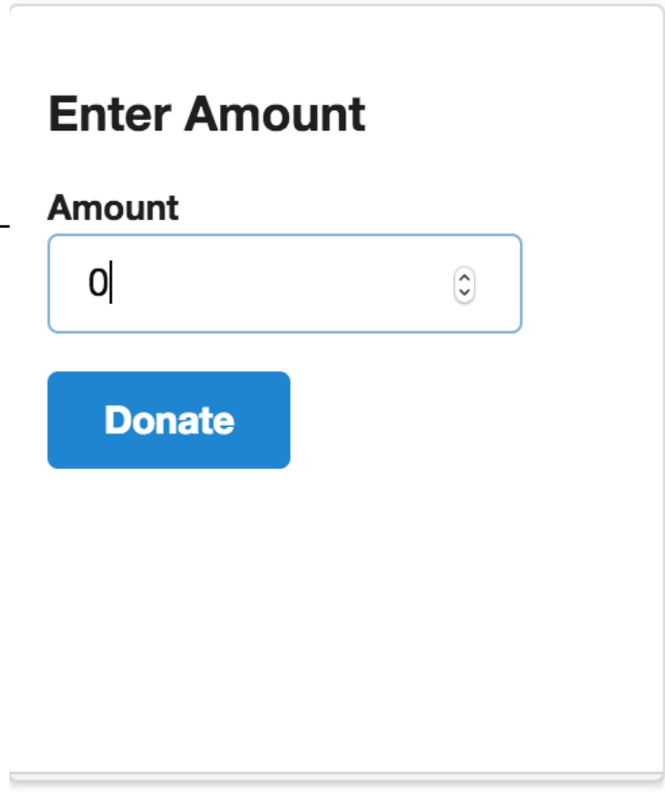
A view-model

- A standard javascript class
- Usually same name as view
- Is the default bind target for views

```
export class Donate {

  amount = 0;

  makeDonation() {
    console.log(`Amount = ${this.amount}`);
  }
}
```

```
<template>
  <form submit.trigger="makeDonation()" class="ui form four ...”>
    <div class="grouped inline fields">
      <h3>Enter Amount </h3>
      <div class="field">
        <label>Amount</label> <input type="number" value.bind="amount">
      </div>
    </div>
    <button class="ui blue submit button">Donate</button>
  </form>
</template>
```

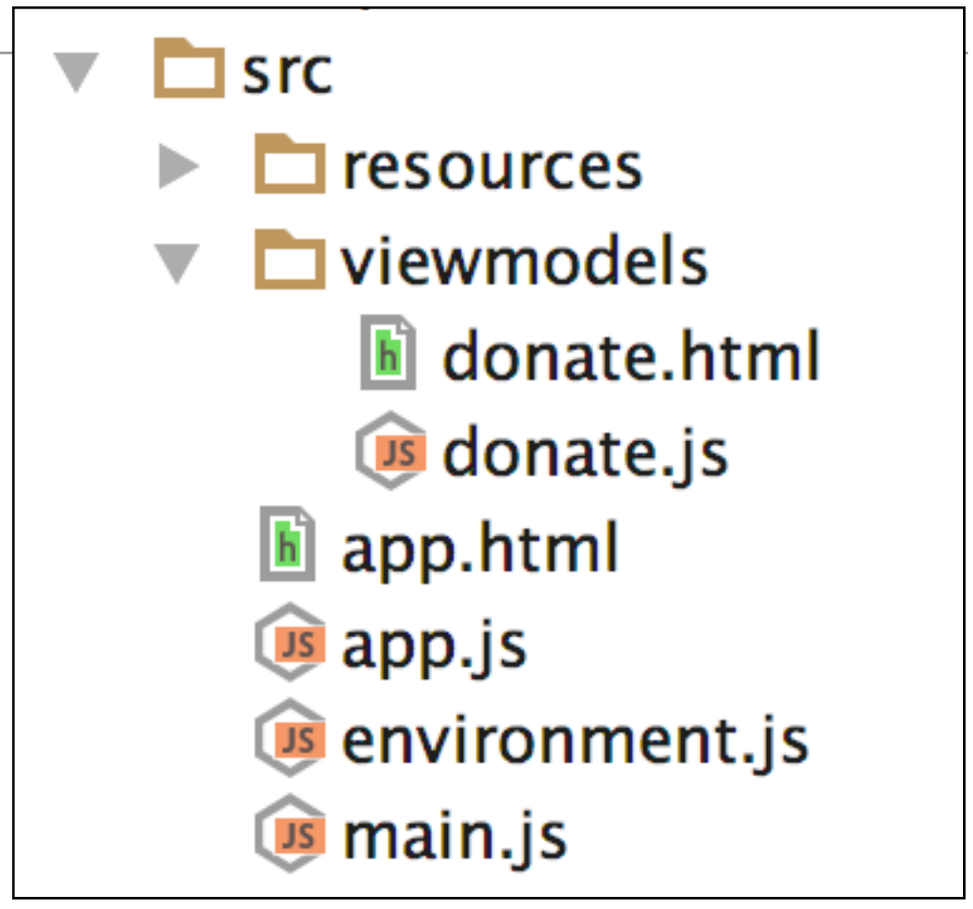
```
export class Donate {

  amount = 0;

  makeDonation() {
    console.log(`Amount = ${this.amount}`);
  }
}
```



- No further code needed to establish event flow
- Values entered in bound fields automatically updated in class and vice-versa
- Pressing ‘Donate’ button triggers invocation of makeDonation() function

app.js & app.html

- Another view / view-model pair
- Simply loads the **donate** view / view-model via the **compose** element

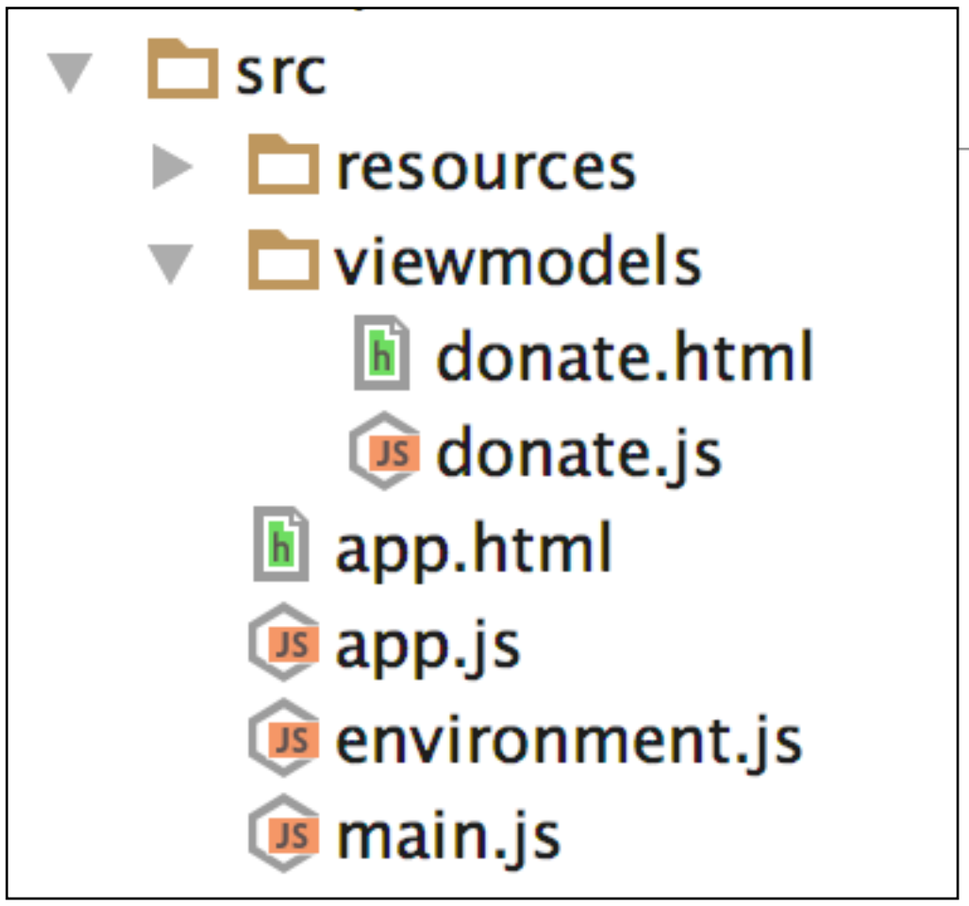



```
<template>
  <compose view-model="./viewmodels/donate"></compose>
</template>
```

```
export class App {
  constructor() {
    this.message = 'Hello World!';
  }
}
```

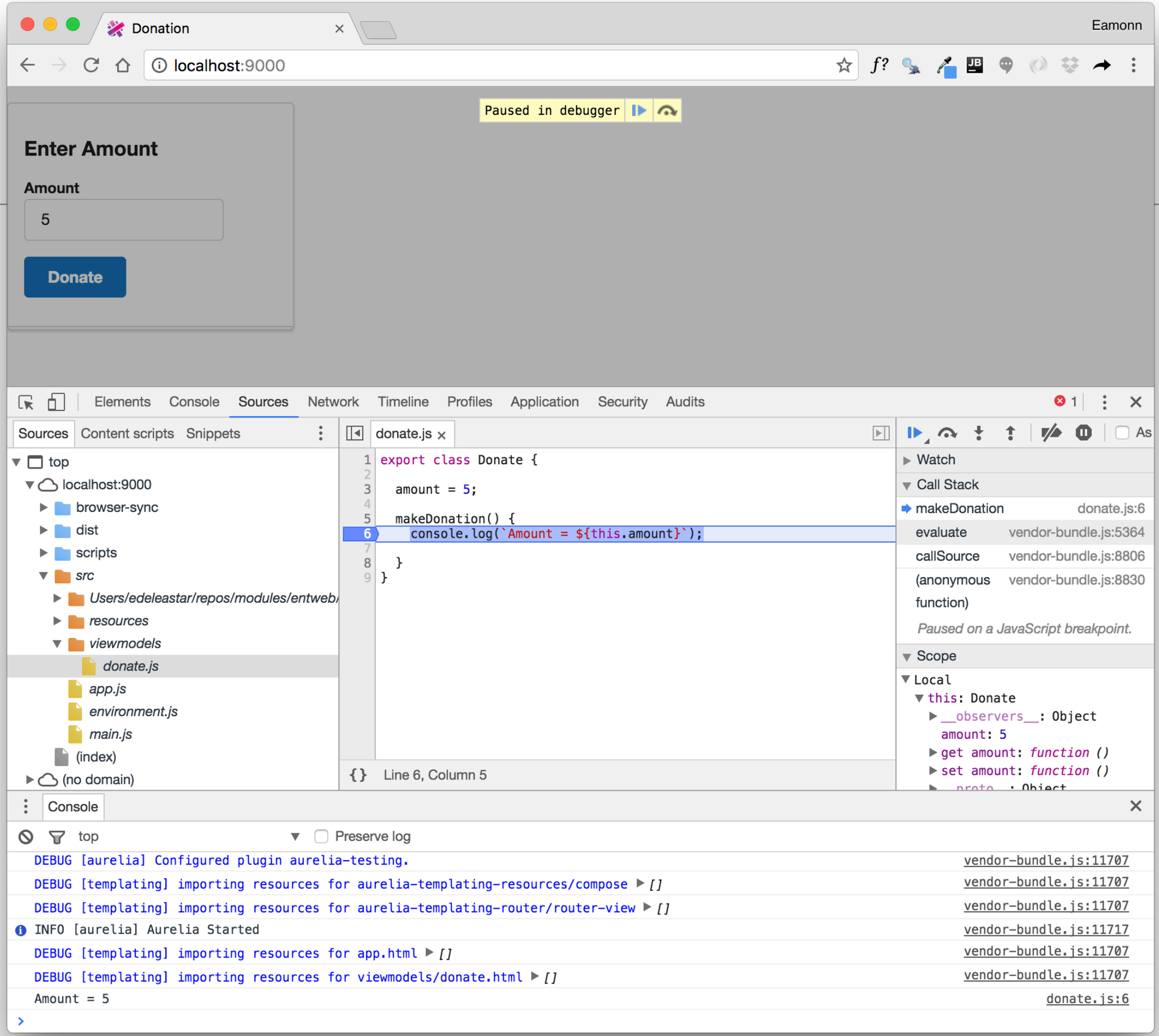
main & environment

```
export default {
  debug: true,
  testing: true
};
```

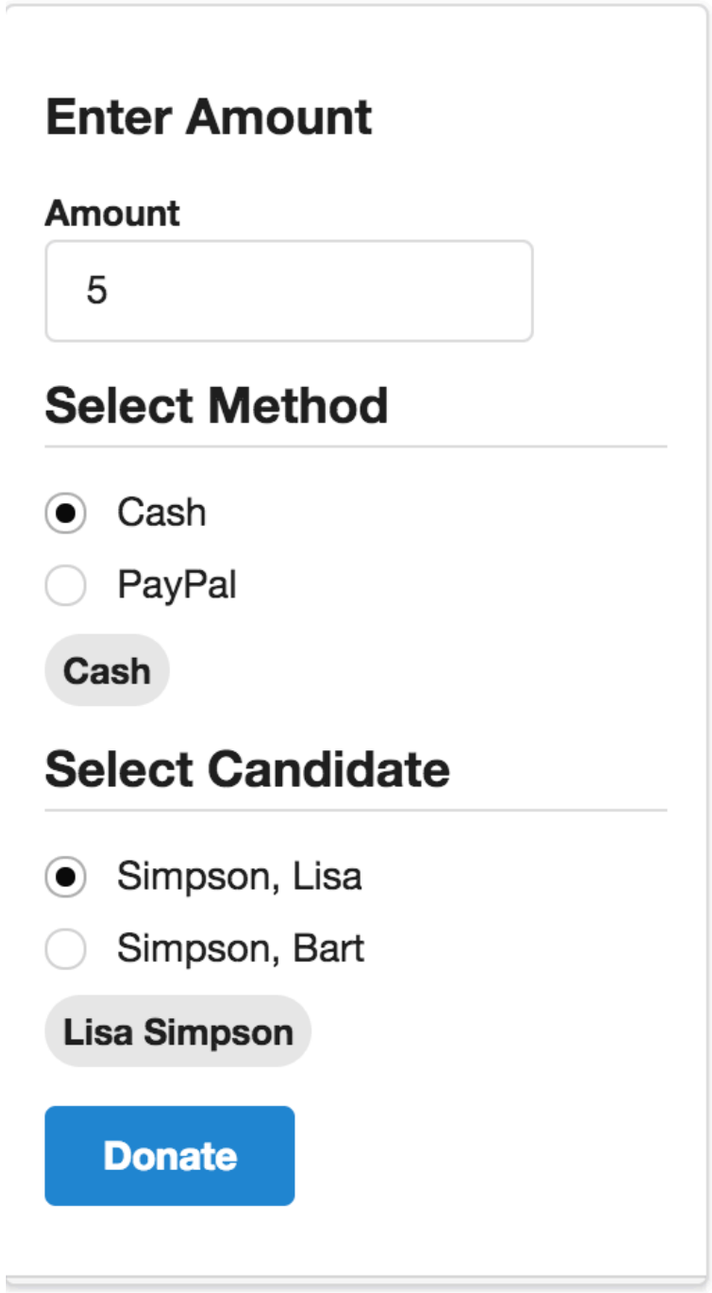



```
import environment from './environment';

export function configure(aurelia) {
  aurelia.use
    .standardConfiguration()
    .feature('resources');

  if (environment.debug) {
    aurelia.use.developmentLogging();
  }

  if (environment.testing) {
    aurelia.use.plugin('aurelia-testing');
  }

  aurelia.start().then(() => aurelia.setRoot());
}
```

- Program entry point
- Sets up config params and starts app
- Default behaviour of aurelia.setRoot() is to load ‘app’ view/ view-model

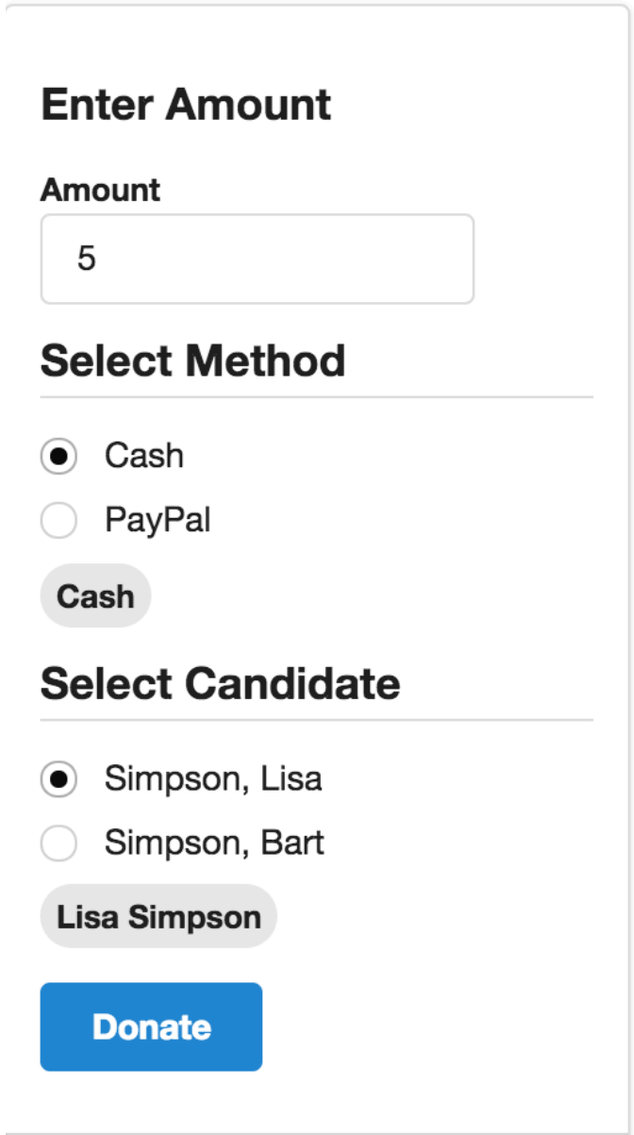

Donation
Eamonn
localhost:9000
Paused in debugger
Enter Amount
Amount
5
Donate
Elements
Console
Sources
Network
Timeline
Profiles
Application
Security
Audits
Sources
Content scripts
Snippets
top
localhost:9000
browser-sync
dist
scripts
src
Users/edeleastar/repos/modules/entweb/
resources
viewmodels
donate.js
app.js
environment.js
main.js
(index)
(no domain)
donate.js
export class Donate {
amount = 5;
makeDonation() {
console.log(`Amount = ${this.amount}`);
}
}
Line 6, Column 5
Watch
Call Stack
makeDonation donate.js:6
evaluate vendor-bundle.js:5364
callSource vendor-bundle.js:8806
(anonymous function) vendor-bundle.js:8830
Paused on a JavaScript breakpoint.
Scope
Local
this: Donate
__observers__: Object
amount: 5
get amount: function ()
set amount: function ()
Console
top
Preserve log
DEBUG [aurelia] Configured plugin aurelia-testing. vendor-bundle.js:11707
DEBUG [templating] importing resources for aurelia-templating-resources/compose [] vendor-bundle.js:11707
DEBUG [templating] importing resources for aurelia-templating-router/router-view [] vendor-bundle.js:11707
INFO [aurelia] Aurelia Started vendor-bundle.js:11717
DEBUG [templating] importing resources for app.html [] vendor-bundle.js:11707
DEBUG [templating] importing resources for viewmodels/donate.html [] vendor-bundle.js:11707
Amount = 5 donate.js:6

Extending the view-model




```
export class Donate {

  amount = 5;
  methods = ['Cash', 'PayPal'];
  selectedMethod = 'Cash';

  candidates = [
    {
      firstName: 'Lisa',
      lastName: 'Simpson'
    },
    {
      firstName: 'Bart',
      lastName: 'Simpson'
    }
  ];
  selectedCandidate = this.candidates[0];

  makeDonation() {
    console.log(`Amount = ${this.amount}`);
    console.log(`Method = ${this.selectedMethod}`);
    console.log(`Candidate = ${this.selectedCandidate.firstName}
                             ${this.selectedCandidate.lastName}`);
  }
}
```

Extending the view




```
<form submit.trigger="makeDonation()" class="ui form four wide column stacked segment">

  <div class="grouped inline fields">
    <h3>Enter Amount </h3>
    <div class="field">
      <label>Amount</label> <input type="number" value.bind="amount">
    </div>
  </div>

  <div class="grouped inline fields">
    <h3 class="ui dividing header"> Select Method </h3>
    <div class="field" repeat.for="method of methods">
      <div class="ui radio checkbox">
        <input type="radio" model.bind="method" checked.bind="selectedMethod">
        <label>${method}</label>
      </div>
    </div>
    <label class="ui circular label"> ${selectedMethod} </label>
  </div>

  <div class="grouped inline fields">
    <h3 class="ui dividing header"> Select Candidate </h3>
    <div class="field" repeat.for="candidate of candidates">
      <div class="ui radio checkbox">
        <input type="radio" model.bind="candidate" checked.bind="selectedCandidate">
        <label>${candidate.lastName}, ${candidate.firstName}</label>
      </div>
    </div>
    <label class="ui circular label"> ${selectedCandidate.firstName}
                                      ${selectedCandidate.lastName}</label>
  </div>

    <button class="ui blue submit button">Donate</button>

</form>
```

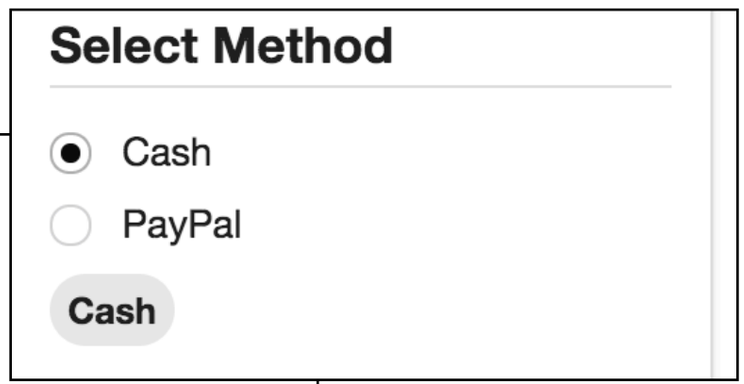
```
export class Donate {

  amount = 5;
  methods = ['Cash', 'PayPa
  selectedMethod = 'Cash';

  candidates = [
    {
      firstName: 'Lisa',
      lastName: 'Simpson'
    },
    {
      firstName: 'Bart',
      lastName: 'Simpson'
    }
  ];
  selectedCandidate = this.candidates[0];

  makeDonation() {
    console.log(`Amount = ${this.amount}`);
    console.log(`Method = ${this.selectedMethod}`);
    console.log(`Candidate = ${this.selectedCandidate.firstName}
                            ${this.selectedCandidate.lastName}`);
  }
}
```

```
...
  <div class="grouped inline fields">
    <h3 class="ui dividing header"> Select Method </h3>
    <div class="field" repeat.for="method of methods">
      <div class="ui radio checkbox">
        <input type="radio" model.bind="method" checked.bind="selectedMethod"
        <label>${method}</label>
      </div>
    </div>
    <label class="ui circular label"> ${selectedMethod} </label>
  </div>

  <div class="grouped inline fields">
    <h3 class="ui dividing header"> Select Candidate </h3>
    <div class="field" repeat.for="candidate of candidates">
      <div class="ui radio checkbox">
        <input type="radio" model.bind="candidate" checked.bind="selectedCandidate
        <label>${candidate.lastName}, ${candidate.firstName}</label>
      </div>
    </div>
    <label class="ui circular label"> ${selectedCandidate.firstName}
                                      ${selectedCandidate.lastName}</label>

  </div>

...
```

view / view-model binding

Radio button binding - payment method

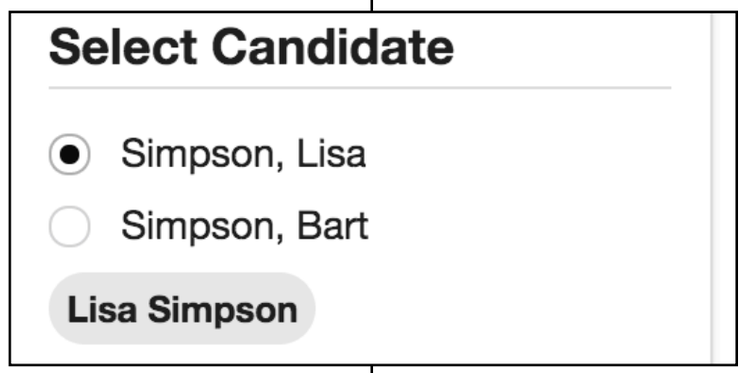
```
export class Donate {
  ...
  methods = ['Cash', 'PayPal'];
  selectedMethod = 'Cash';
  ...
 }
```




```
...
    <div class="field" repeat.for="method of methods">
      <div class="ui radio checkbox">
        <input type="radio" model.bind="method" checked.bind="selectedMethod">
        <label>${method}</label>
      </div>
    </div>
    <label class="ui circular label"> ${selectedMethod} </label>

...
```

Radio button binding - Candidates

```
export class Donate {
  ...
  candidates = [
    {
      firstName: 'Lisa',
      lastName: 'Simpson'
    },
    {
      firstName: 'Bart',
      lastName: 'Simpson'
    }
  ];
  selectedCandidate = this.candidates[0];
  ...
}
```




```
    ...

      <div class="grouped inline fields">
        <h3 class="ui dividing header"> Select Candidate </h3>
        <div class="field" repeat.for="candidate of candidates">
          <div class="ui radio checkbox">
            <input type="radio" model.bind="candidate" checked.bind="selectedCandidate">
            <label>${candidate.lastName}, ${candidate.firstName}</label>
          </div>
        </div>
        <label class="ui circular label"> ${selectedCandidate.firstName}
                                          ${selectedCandidate.lastName}</label>

      </div>
    ...
```