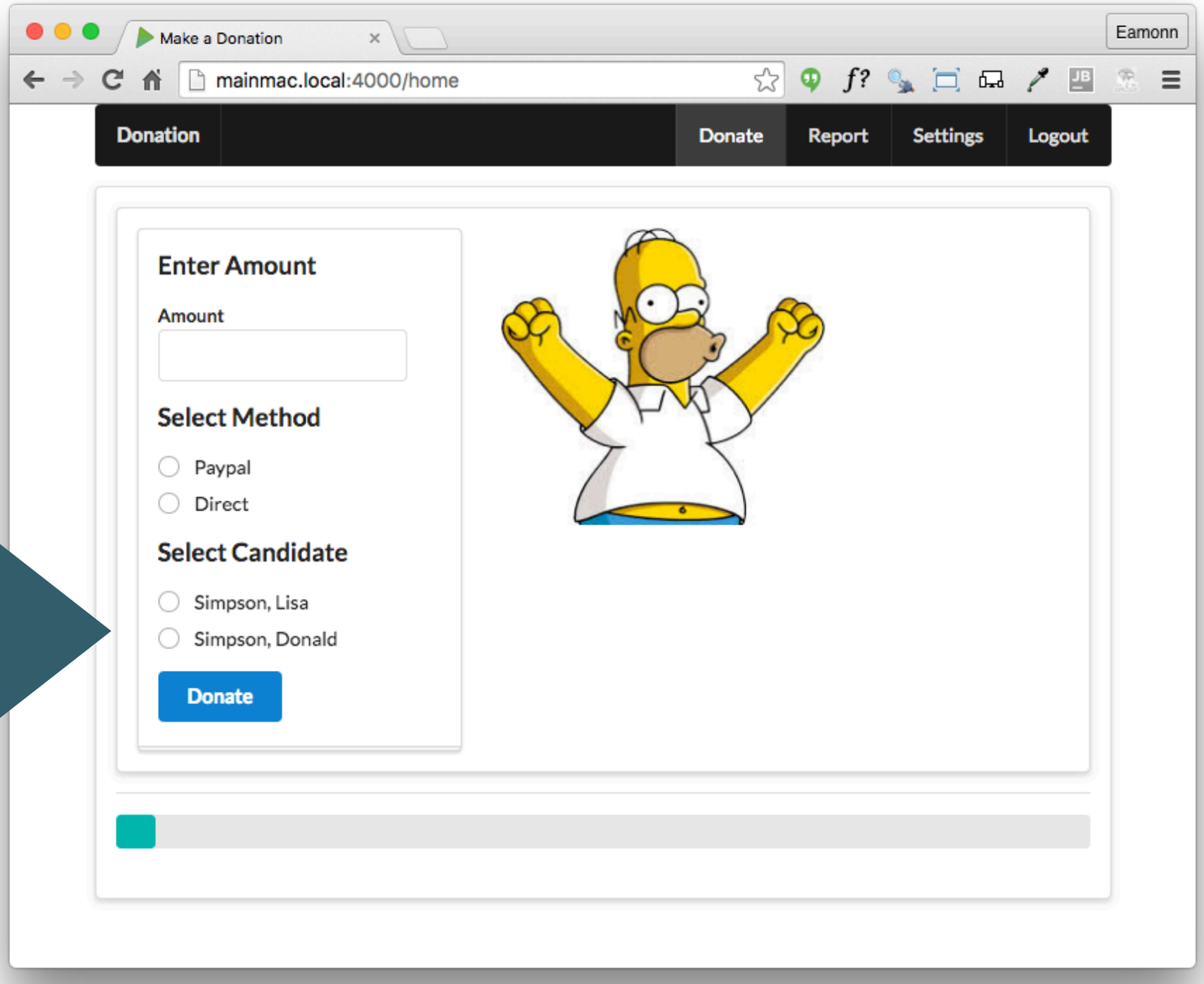
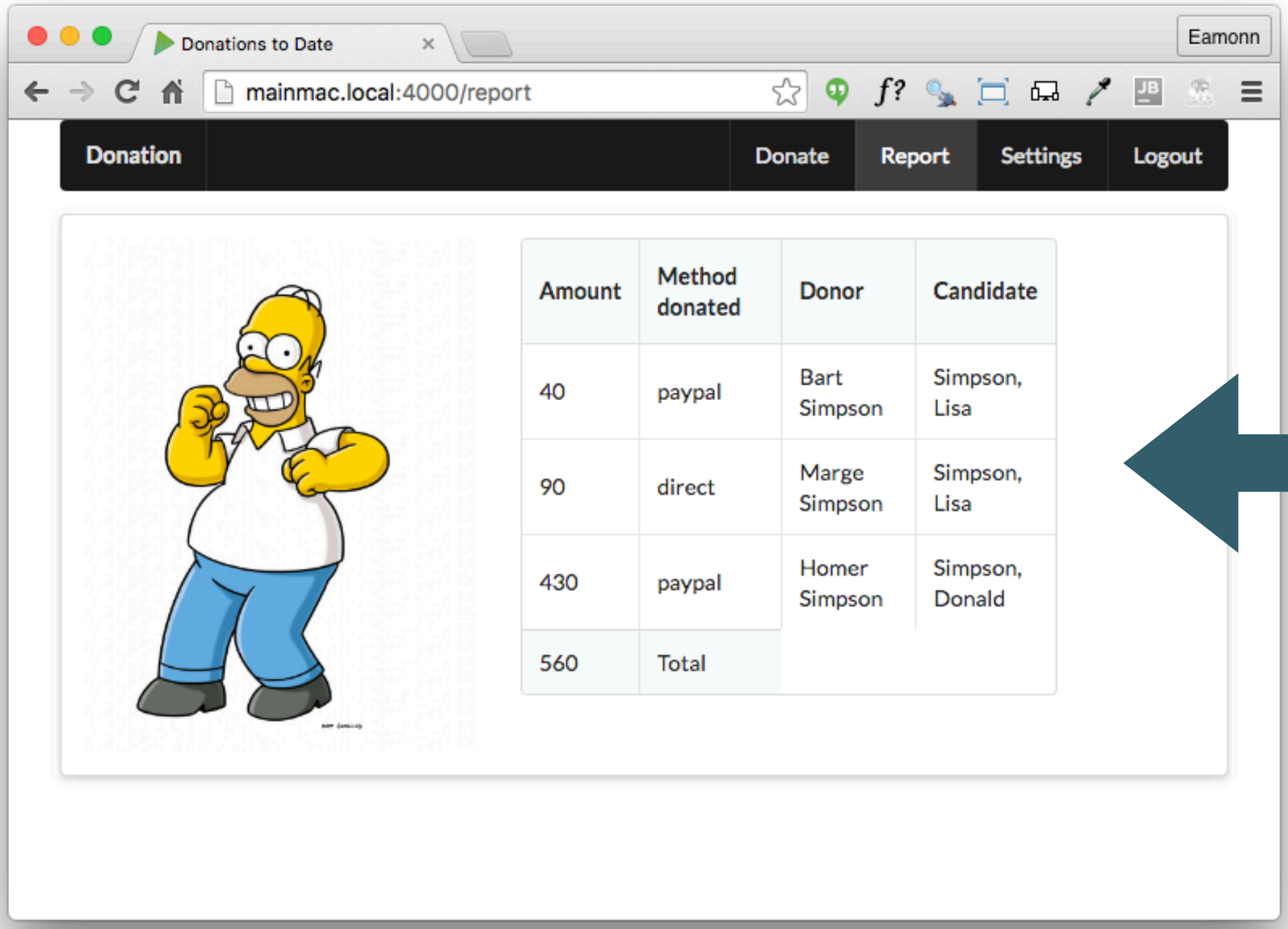
Introducing a Candidate Model

Candidates

- Extend the application to to support multiple candidates



Donations to Candidates



- Donation reports candidate donated to

Candidate Model

candidate.js

```
'use strict';

const mongoose = require('mongoose');

const candidateSchema = mongoose.Schema({
  firstName: String,
  lastName: String,
  office: String,
});

const Candidate = mongoose.model('Candidate', candidateSchema);
module.exports = Candidate;
```

- Represent a Candidate

Seed the Candidate Model

candidate.js

```
const candidateSchema = mongoose.Schema({
  firstName: String,
  lastName: String,
  office: String,
});
```

initdata.json

```
"candidates": {
  "_model": "Candidate",
  "lisa": {
    "firstName": "Lisa",
    "lastName": "Simpson",
    "office": "President"
  },
  "donald": {
    "firstName": "Donald",
    "lastName": "Simpson",
    "office": "President"
  }
},
```

db.js

```
var seeder = require('mongoose-seeder');
const data = require('./initdata.json');
const Donation = require('./donation');
const User = require('./user');
const Candidate = require('./candidate.js');
seeder.seed(data, { dropDatabase: false, dropCollections: true }).then(dbData => {
  ...
}).catch(err => {
  ...
});
```

Candidate Reference in Donation

donation.js

```
const donationSchema = mongoose.Schema({
  amount: Number,
  method: String,
  donor: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
  },
  candidate:  {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Candidate',
  },
});
```

- Donations new refer to candidate
- Seeded model must also be updated

initdata.json

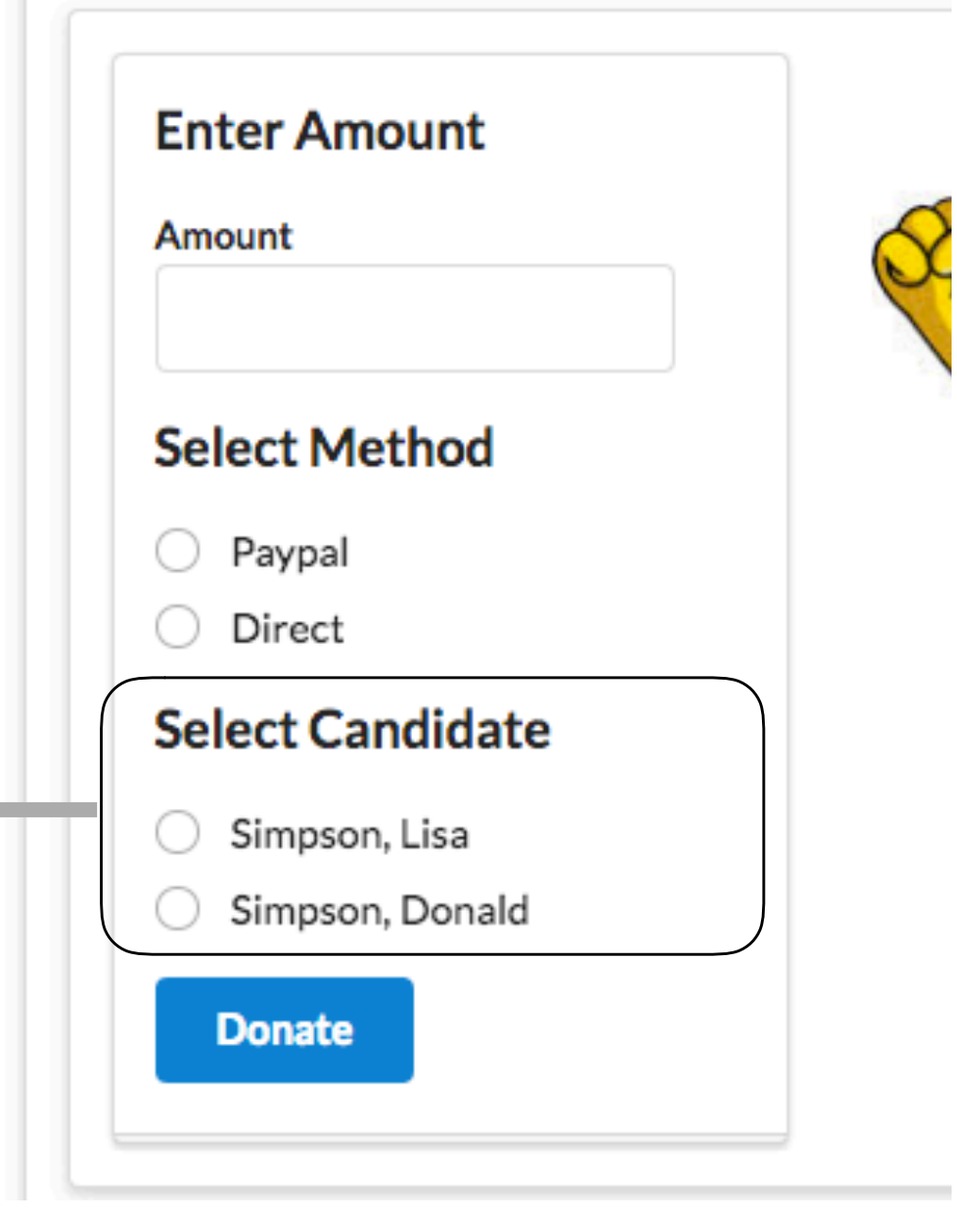
```
"donations": {
  "_model": "Donation",
  "one": {
    "amount": 40,
    "method": "paypal",
    "donor": "->users.bart",
    "candidate": "->candidates.lisa"
  },
  "two": {
    "amount": 90,
    "method": "direct",
    "donor": "->users.marge",
    "candidate": "->candidates.lisa"
  },
  "three": {
    "amount": 430,
    "method": "paypal",
    "donor": "->users.homer",
    "candidate": "->candidates.donald"
  }
}
```

Rendering the Donate view

- Donate hander needs candidate list for the view:

```
handler: function (request, reply) {
  Candidate.find({}).then(candidates => {
    reply.view('home', {
      title: 'Make a Donation',
      candidates: candidates,
    });
  }).catch(err => {
    reply.redirect('/');
  });
},
```

```
<div class="grouped inline fields">
  <h3> Select Candidate </h3>
  {{#each candidates }}
    <div class="field">
      <div class="ui radio checkbox">
        <input type="radio" name="candidate"
               value="{{lastName}},{{firstName}}">
        <label>{{lastName}}, {{firstName}}</label>
      </div>
    </div>
  {{/each}}
</div>
```



Donate Handler

- To create a donation we need:
  - id of donor
  - id of candidate
- This requires 2 database read operations on 2 different collections

donation.js

```
const donationSchema = mongoose.Schema({
  amount: Number,
  method: String,
  donor: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
  },
  candidate:  {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Candidate',
  },
});
```

Donate Handler

```
handler: function (request, reply) {
  var userEmail = request.auth.credentials.loggedInUser;
  let userId = null;
  let donation = null;
  User.findOne({ email: userEmail }).then(user => {
    let data = request.payload;
    userId = user._id;
    donation = new Donation(data);
    const rawCandidate = request.payload.candidate.split(',');
    return Candidate.findOne({ lastName: rawCandidate[0], firstName: rawCandidate[1] });
  }).then(candidate => {
    donation.donor = userId;
    donation.candidate = candidate._id;
    return donation.save();
  }).then(newDonation => {
    reply.redirect('/report');
  }).catch(err => {
    reply.redirect('/');
  });
},
```

Donate Handler

```
handler: function (request, reply) {
  var userEmail = request.auth.credentials.loggedInUser;
  let userId = null;
  let donation = null;
  User.findOne({ email: userEmail }).then(user => {
    let data = request.payload;
    userId = user._id;
    donation = new Donation(data);
    const rawCandidate = request.payload.candidate.split(',');
    return Candidate.findOne({ lastName: rawCandidate[0], firstName: rawCandidate[1] });
  }).then(candidate => {
    donation.donor = userId;
    donation.candidate = candidate._id;
    return donation.save();
  }).then(newDonation => {
    reply.redirect('/report');
  }).catch(err => {
    reply.redirect('/');
  });
},
```

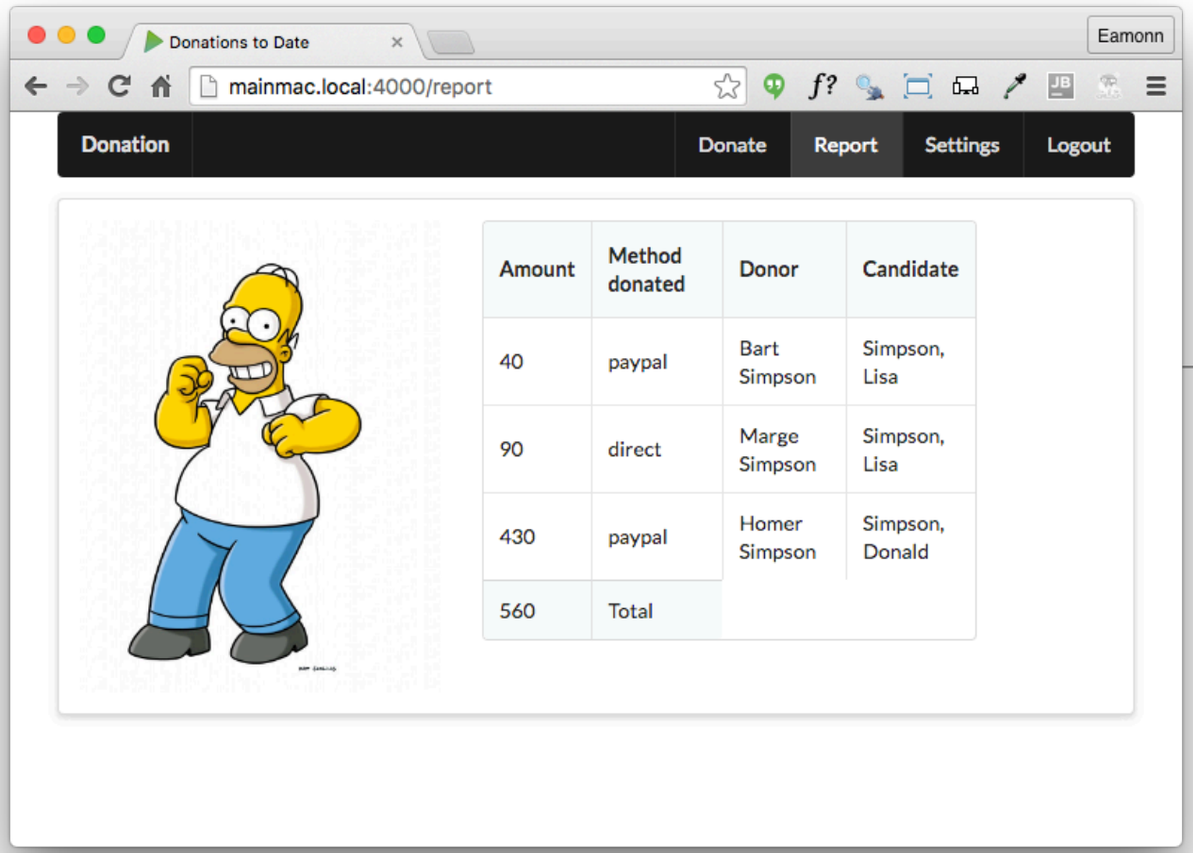
Locate User Object

Create New Donation

Locate Candidate Object

Initalize New Donation with User and Candidate IDs

Save Donation



Populating the Donations

doationlist.hbs

```
<section class="ui raised segment">
  <div class="ui grid">
    <aside class="six wide column">
      <img src="images/homer5.jpg" class="ui medium image">
    </aside>
    <article class="eight wide column">
      <table class="ui celled table segment">
        <thead>
          <tr>
            <th>Amount</th>
            <th>Method donated</th>
            <th>Donor</th>
            <th>Candidate</th>
          </tr>
        </thead>
        <tbody>
          {{#each donations}}
            <tr>
              <td> {{amount}} </td>
              <td> {{method}} </td>
              <td> {{donor.firstName}} {{donor.lastName}} </td>
              <td> {{candidate.lastName}}, {{candidate.firstName}} </td>
            </tr>
          {{/each}}
        </tbody>
      </table>
    </article>
  </div>
</section>
```

```
handler: function (request, reply) {
  Donation.find({}).populate('donor').populate('candidate').then(allDonations => {
    reply.view('report', {
      title: 'Donations to Date',
      donations: allDonations,
    });
  }).catch(err => {
    reply.redirect('/');
  });
},
```