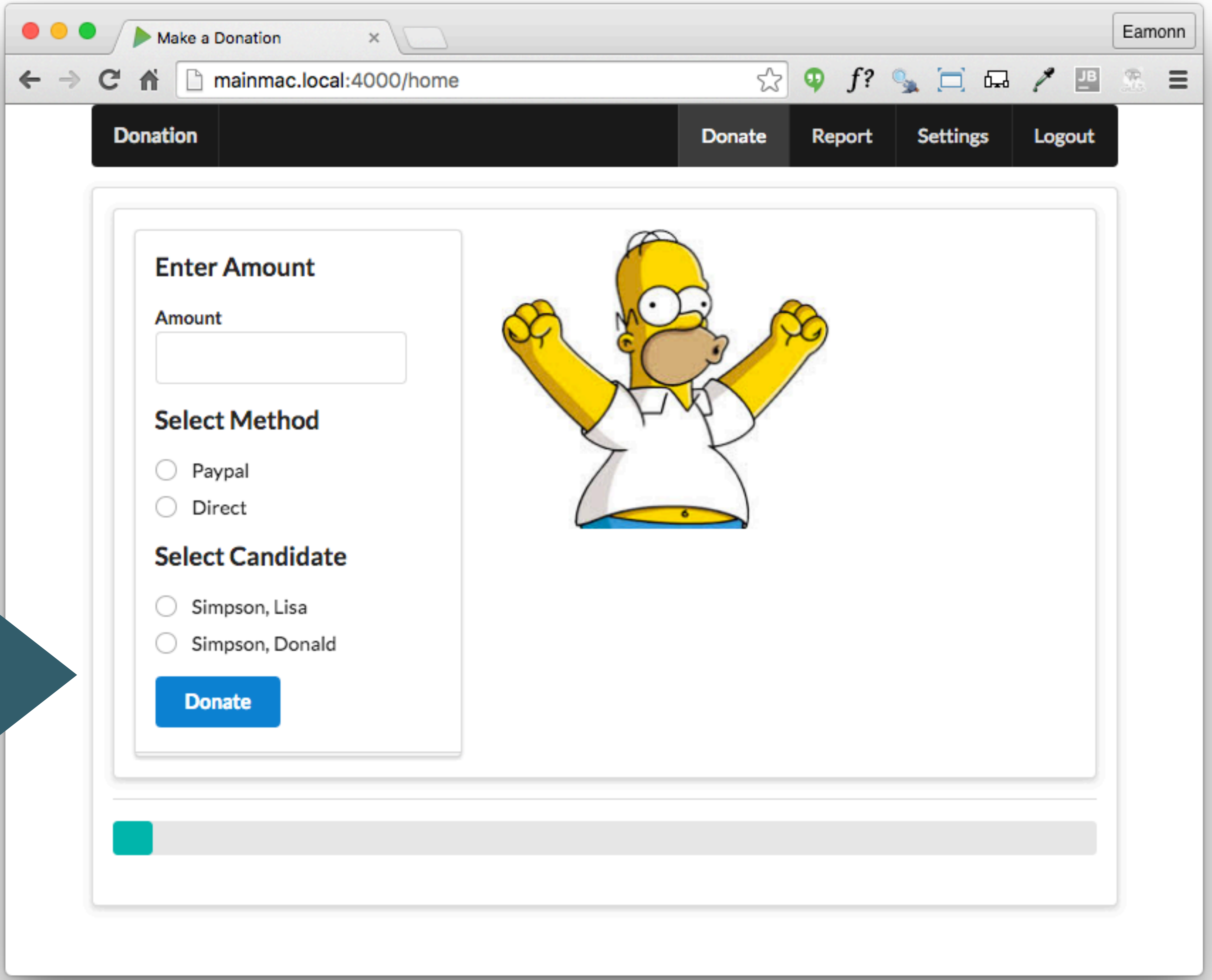
Introducing a Candidate Model

Candidates

- Extend the application to to support multiple candidates



Enter Amount

Amount

Select Method

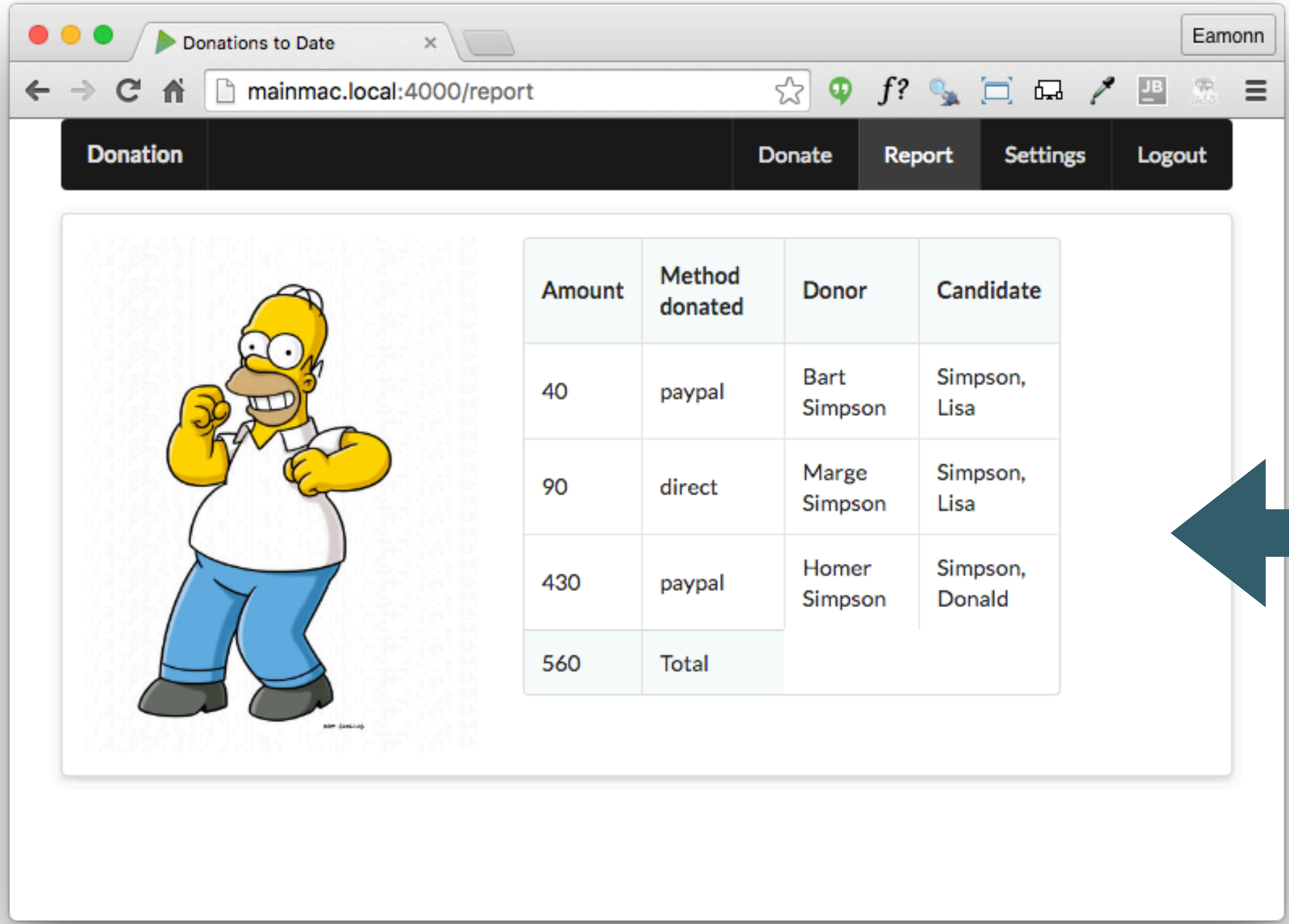
- Paypal
- Direct

Select Candidate

- Simpson, Lisa
- Simpson, Donald

Donate

Donations to Candidates



Amount	Method donated	Donor	Candidate
40	paypal	Bart Simpson	Simpson, Lisa
90	direct	Marge Simpson	Simpson, Lisa
430	paypal	Homer Simpson	Simpson, Donald
560	Total		

- Donation reports candidate donated to

Candidate Model

candidate.js

```
'use strict';

const Mongoose = require('mongoose');
const Schema = Mongoose.Schema;

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

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

- Represent a Candidate

Seed the Candidate Model

initdata.json

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

candidate.js

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

db.js

```javascript
async function seed() {
  var seeder = require('mais-mongoose-seeder')(Mongoose);
  const data = require('./initdata.json');
  const Donation = require('./donation');
  const Candidate = require('./candidate.js');
  const User = require('./user');
  const dbData = await seeder.seed(data, { dropDatabase: false, dropCollections: true });
  console.log(dbData);
}
```

Candidate Reference in Donation

donation.js

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

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

initdata.json

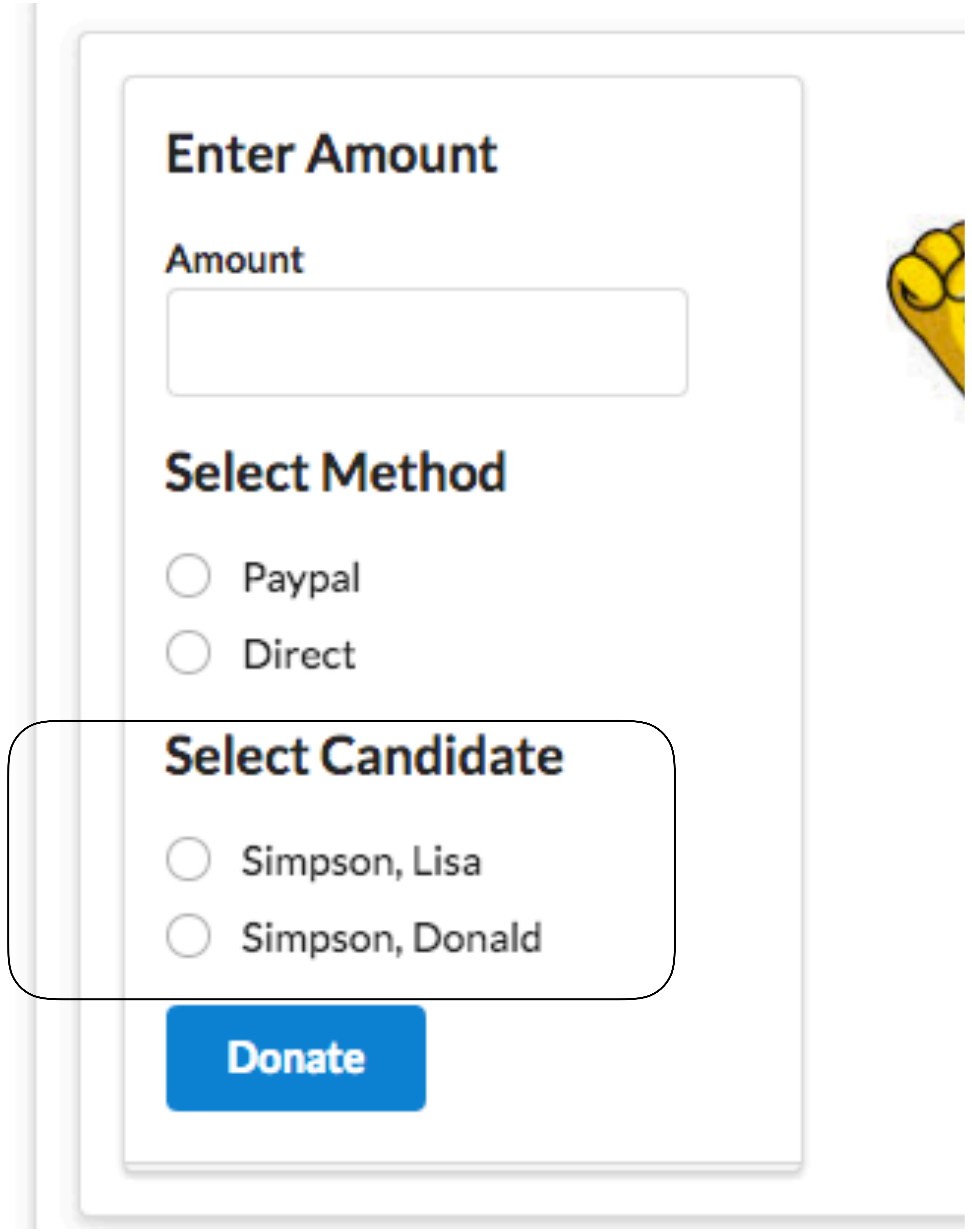
```
"donations": {
  "_model": "Donation",
  "one": {
    "amount": 40,
    "method": "paypal",
    "donor": "->users.bart",
    "candidate": "->candidates.lisa"
  },
  "two": {
    "amount": 90,
    "method": "direct",
    "donor": "->users.marge",
    "candidate": "->candidates.lisa"
  },
  "three":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

Rendering the Donate view

- Donate hander needs candidate list for the view:

```
home: {
  handler: async function(request, h) {
    const candidates = await Candidate.find();
    return h.view('home', {
      title: 'Make a Donation',
      candidates: candidates
    });
  }
},
```

```
<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>
```



Donation Model

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

```
donate: {
  handler: async function(request, h) {
    try {
      const id = request.auth.credentials.id;
      const user = await User.findById(id);
      const data = request.payload;

      const rawCandidate = request.payload.candidate.split(',');
      const candidate = await Candidate.findOne({
        lastName: rawCandidate[0],
        firstName: rawCandidate[1]
      });

      const newDonation = new Donation({
        amount: data.amount,
        method: data.method,
        donor: user._id,
        candidate: candidate._id
      });
      await newDonation.save();
      return h.redirect('/report');
    } catch (err) {
      return h.view('main', { errors: [{ message: err.message }] });
    }
  }
}
```

```
donate: {
  handler: async function(request, h) {
    try {
      const id = request.auth.credentials.id;
      const user = await User.findById(id);
      const data = request.payload;

      const rawCandidate = request.payload.candidate.split(',');
      const candidate = await Candidate.findOne({
        lastName: rawCandidate[0],
        firstName: rawCandidate[1]
      });

      const newDonation = new Donation({
        amount: data.amount,
        method: data.method,
        donor: user._id,
        candidate: candidate._id
      });
      await newDonation.save();
      return h.redirect('/report');
    } catch (err) {
      return h.view('main', { errors: [{ message: err.message }] });
    }
  }
}
```

Locate User Object

Locate Candidate Object

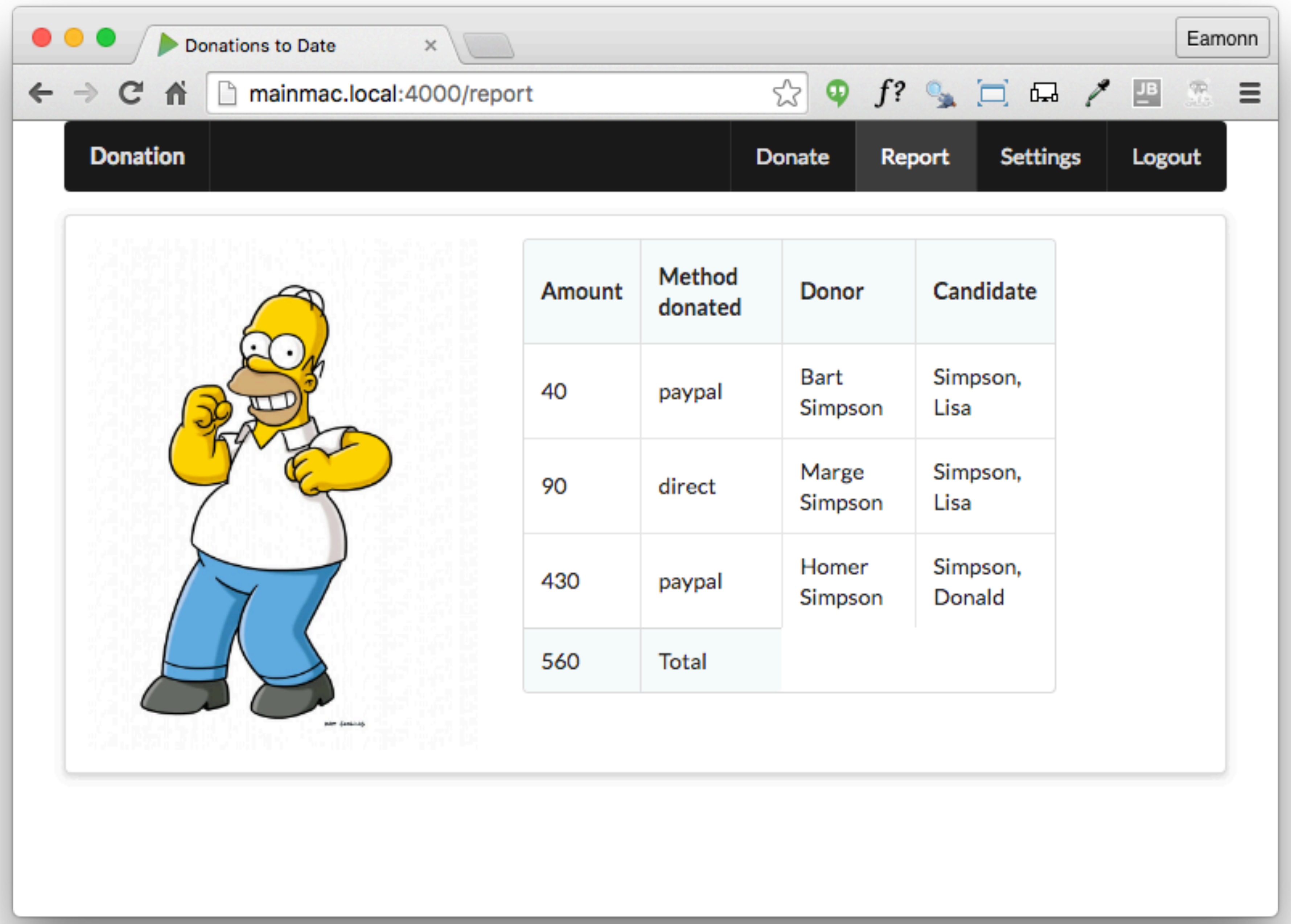
Create New Donation

Initalize New Donation with User and Candidate IDs

Save Donation

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>
```



Amount	Method donated	Donor	Candidate
40	paypal	Bart Simpson	Simpson, Lisa
90	direct	Marge Simpson	Simpson, Lisa
430	paypal	Homer Simpson	Simpson, Donald
560	Total		

Populating the Report

```
report: {
  handler: async function(request, h) {
    try {
      const donations = await Donation.find().populate('donor').populate('candidate');
      return h.view('report', {
        title: 'Donations to Date',
        donations: donations
      });
    } catch (err) {
      return h.view('main', { errors: [{ message: err.message }] });
    }
  }
},
```

Alternative Donation Handler - using Callbacks

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

Alternative Donation Handler - using Callbacks

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

```
donate: {
  handler: async function(request, h) {
    try {
      const id = request.auth.credentials.id;
      const user = await User.findById(id);
      const data = request.payload;

      const rawCandidate = request.payload.candidate.split(',');
      const candidate = await Candidate.findOne({
        lastName: rawCandidate[0],
        firstName: rawCandidate[1]
      });

      const newDonation = new Donation({
        amount: data.amount,
        method: data.method,
        donor: user._id,
        candidate: candidate._id
      });
      await newDonation.save();
      return h.redirect('/report');
    } catch (err) {
      return h.view('main', { errors: [{ message: err.message }] });
    }
  }
}
```

Alternative Donation Handler - using Callbacks

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

```
donate: {
  handler: async function(request, h) {
    try {
      const id = request.auth.credentials.id;
      const user = await User.findById(id);
      const data = request.payload;

      const rawCandidate = request.payload.candidate.split(',');
      const candidate = await Candidate.findOne({
        lastName: rawCandidate[0],
        firstName: rawCandidate[1]
      });

      const newDonation = new Donation({
        amount: data.amount,
        method: data.method,
        donor: user._id,
        candidate: candidate._id
      });
      await newDonation.save();
      return h.redirect('/report');
    } catch (err) {
      return h.view('main', { errors: [{ message: err.message }] });
    }
  }
}
```

Alternative Donate Handler - using Promises

```
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('/');
  });
},
```