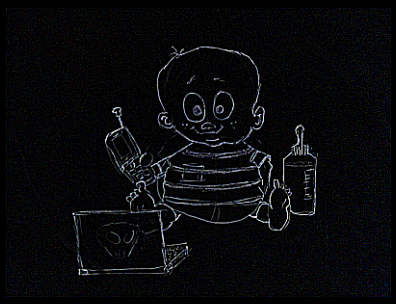
Chapter 1

Quick Launch

It seems I am on speed.

Nothing gives a better kick than working and releasing fast. The following code will get you a "launching soon" page ready with beta invite albeit manually. It requires node.js, an evented IO framework by Ryan Dahl. The code itself is written in coffeescript, a language by Jeremy Ashkenas. First be comfortable with code and the fact that this small code will get us started and then we can move on to get it working.



```
1 http = require 'http'
2 http.createServer (req, res) ->
3         res.writeHead 200, {'Content-Type': 'text/plain'}
4         res.end '257. Manage Movies. Beta invites via 257.invite@gmail.com'
5 .listen 8080
```

Tip Use commercial solutions like unbounce to set up quick sign up page

Chapter 2

Express

Run. Run. Run.

Time to take the express. A quick reorganization of code so as to switch to a big framework which will make life easier in long run. As you can figure out we don't need to parse req.url and add switch statement to handle different url, if we stuck to last example. Express makes routing a breeze.

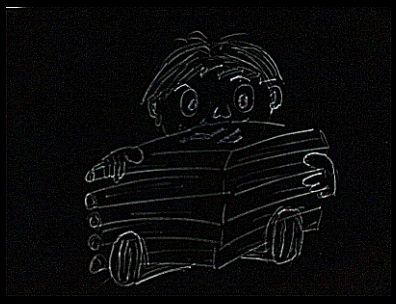

```
1  http = require 'http'
2  express = require 'express'
3
4  app = express.createServer()
5
6  app.get '/', (req, res) ->
7          res.send '257. Manage Movies. Beta invites via 257.invite@gmail.com'
8
9  app.get '/movie/:id', (req, res) ->
10         res.send 'movie ' + req.params.id
11
12 app.listen 8080
```

Chapter 3

Database

You can has data.

MongoDB will store our data. Note that we have to think asynchronously. The database choice here was dictated more by ease rather than a good fit. In fact, a document based database in not really optimal for storing relational data. Lets focus on the working example which pulls the data from the database. Visit mongodb website for installation, building a database and storing dummy data. We will use mongoose module to interact with the server.



```
1  http = require 'http'
2  express = require 'express'
3  mongoose = require 'mongoose'
4
5  #db init stuff
6  mongoose.connect 'mongodb://localhost/db'
7
8  movieSchema = new mongoose.Schema({
9          id: Number,
10         name: String
11 });
12
13 mongoose.model 'movie', movieSchema;
14 movie = mongoose.model 'movie';
15
16 #scumbag never disconnect
```

```
17 #mongoose.disconnect()
18
19 app = express.createServer()
20
21 app.get '/', (req, res) ->
22         res.send '257. Manage Movies. Beta invites via 257.invite@gmail.com'
23
24 app.get '/movie/:id', (req, res) ->
25         movie.findOne {
26                 id: req.params.id
27         }, (err, m) ->
28                 if (err || !m)
29                         res.send 'not found'
30                         return
31                 res.send 'movie ' + m.name
32
33 app.listen 8080
```

Movie data can be inserted via mongo command line interface

Tip ./mongodb-linux-i686-1.6.5/bin/mongod –dbpath ./data/db/

Tip ./mongodb-linux-i686-1.6.5/bin/mongo localhost/db

Chapter 4

Intermission

Organize your data before you die.

We will stick to a common layout in order to keep the code organized in long run. It also makes it easier to understand and maintain. Here, models directory will contain the data models corresponding to the database. Controllers will be handlers of different type of requests. The views will contain Jade templates which will be rendered by the jade module.

```
app.coffee              (our application)
|--models               (contains data models)
|--views                (jade templates)
|--controllers          (handler for different requests)
|--data                 (data storage)
   |--db
|--public               (static content)
   |--js
   |--css
   |--img
```

According to this scheme, the contents will now be

app.coffee

```
1  http = require 'http'
2  express = require 'express'
3  mongoose = require 'mongoose'
4  jade = require 'jade'
5
6  mongoose.connect 'mongodb://localhost/db'
7  app = express.createServer();
8
9  require './models/movie'
10 app.movie = mongoose.model 'movie';
11
12 homeController = require './controllers/home'
13 homeController app
14
15 movieController = require './controllers/movie'
16 movieController app
17
18 app.listen 8080
```

movie controller

```
1  mongoose = require 'mongoose'
2
3  movieSchema = new mongoose.Schema({
4          id: Number,
5          name: String
6  });
7
8  mongoose.model 'movie', movieSchema
```

homepage controller

```
1  module.exports = (app) ->
2          app.get '/', (req, res) ->
3                  res.end '257. Manage Movies. Beta invites via 257.invite@gmail.com'
```

movie model

```
1  module.exports = (app) ->
2          app.get '/movie/:id', (req, res) ->
3                  app.movie.findOne {
4                          id: req.params.id
5                  }, (err, m) ->
6                          if (err || !m)
7                                  res.send 'not found'
8                                  return
9                          res.end 'movie ' + m.name
```

Current example doesn't really use templates because we are just refactoring the code from last chapter. But the organisation reflects how we want to categorize similar type of source files in the same organisation hierarchy under same directory.