The dark version

better to burn out than to fade away

March 21, 2011

Chapter 1

Preface

This book is meant for young enterpreneurs and those who like to fiddle with web based startups but don't have a clue about the technology involved. Lets get a few things right.

Identify problem and focus
Lot of time people start building upon an idea but don't have a clue what they are doing after months into the work. Then they start doing something else, change their focus according to what's hot or the 'in' thing. This guarantees that whatever work you have done so far is a waste.
On the contrary, some startups may have the clarity but give in to the customer demands. Both the scenario reduces the brightness of a startup. Its important to be crystal clear about **what** you are offering to **whom** you are offering.

Aggressive about solving problem
No matter what it costs you are gonna do it. This is the best determiner for the success of your startup. Paul Graham even suggested using 'benevolence' as a strategy to keep your spirits high. It gives you a sharp focus. It effectively disconnects you from the world into a boxing ring where you are fighting with your problem. Airbnb came up with a nifty idea and sold cereal boxes at a Democratic Convention in Denver and collected a good amount for funding. This is an example of being 'relentlessly resourceful' which is required from

every single person in a startup.

Figure out the right solution
Is web based application the right solution for you? Do you really need to hire a team of hackers to solve your problem. Can you use existing platforms? The resources you need might even be freely available. Say, a blog, to reach out to your customers. Its free complete with all the SEO. In fact, Groupon started off as a wordpress blog.

Chapter 2

Introduction

What the beast?

I will follow a particular set of technology to demonstrate how to build a web application for your needs. The demo app will be a movie site, much like IMDB, just that it would be more user centric.

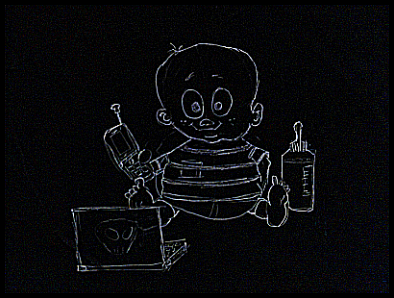
When you are stuck Stackoverflow will be your best friend. Most of your queries will be already answered there. Next best friend is Google. And when you feel like expert of nodejs and can't find answers online, you can even join the mailing list.

Chapter 3

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.



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

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

Chapter 4

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.

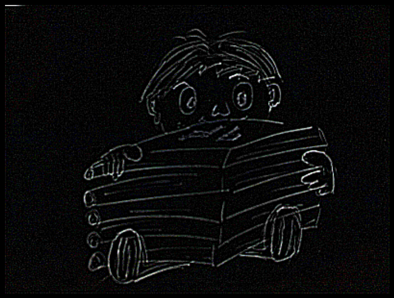
```
http = require 'http'
express = require 'express'

app = express.createServer()

app.get '/', (req, res) ->
        res.send '257. Manage Movies. Beta invites via 257.invite@gmail.com'

app.get '/movie/:id', (req, res) ->
        res.send 'movie ' + req.params.id

app.listen 8080
```

Chapter 5

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.



```
http = require 'http'
express = require 'express'
mongoose = require 'mongoose'

#db init stuff
mongoose.connect 'mongodb://localhost/db'

movieSchema = new mongoose.Schema({
        id: Number,
        name: String
});
```

```

mongoose.model 'movie', movieSchema;
movie = mongoose.model 'movie';

#scumbag never disconnect
#mongoose.disconnect()

app = express.createServer()

app.get '/', (req, res) ->
        res.send '257. Manage Movies. Beta invites via 257.invite@gmail.com'

app.get '/movie/:id', (req, res) ->
        movie.findOne {
                id: req.params.id
        }, (err, m) ->
                if (err || !m)
                        res.send 'not found'
                        return
                res.send 'movie ' + m.name

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/

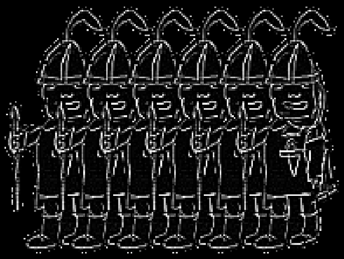
Tip ./mongodb-linux-i686-1.6.5/bin/mongo localhost/db

Chapter 6

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

```
http = require 'http'
express = require 'express'
mongoose = require 'mongoose'
jade = require 'jade'

mongoose.connect 'mongodb://localhost/db'
app = express.createServer();

require './models/movie'
app.movie = mongoose.model 'movie';

homeController = require './controllers/home'
homeController app

movieController = require './controllers/movie'
movieController app

app.listen 8080
```

movie controller

```
mongoose = require 'mongoose'

movieSchema = new mongoose.Schema({
        id: Number,
        name: String
});

mongoose.model 'movie', movieSchema
```

homepage controller

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

movie model

```
module.exports = (app) ->
        app.get '/movie/:id', (req, res) ->
                app.movie.findOne {
                        id: req.params.id
                }, (err, m) ->
                        if (err || !m)
                                res.send 'not found'
                                return
                        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.