Agile Project Development

In this lecture we will look at the stages in building a fullstack MEAN project, and start build a static Node project.

The development will run over the next 5-6 lectures and will go through the following stages:

1. Build a static site
2. Design the data model and create a database
3. Build a data API
4. Hook the database into the application
5. Augment the application
Stage 1. Build a static site

The first stage is to build a static version of the application, which is essentially a number of HTML screens. The aims of this stage are

- To quickly figure out the layout – use pen and paper
- To ensure that the user flow makes sense
- At this all we want to do is create a working mockup of the main screens and journeys that a user will take through the application.
A simple MEAN architecture

- Node
- Express (webserver)
- Jade Views
- Mongoose Models

Mongo DB

Client
- HTML/CSS
- JS
- Angular
Stage 2. Design the data model

The next thing to do is look at any hard-coded data in the static application and put it into a database. The aims of this stage are

- To define a data model that reflects the requirements of the application
- To create a database to work with the model
- The first part of this is to define the data model. Stepping back to a bird’s-eye view, what are the objects we need data about, how are the objects connected, and what data is held in them?
A simple MEAN architecture

- Node
- Express
  - webserver
  - PORT
- Jade Views
- Mongoose Models

Mongo DB

Client
- HTML/CSS
- JS
- Angular
Stage 3: Build the data API

Once we have a static site on one hand and a database on the other. This stage and the next take the natural steps of linking them together.

To do this we create a REST (REpresentative STate) API that will allow our application to interact with the database.
A simple MEAN architecture

- Node
- Express
- Jade Views
- Mongoose Models

Client
- HTML/CSS
- Angular

Mongo DB
Stage 4: Add database to the application

When we get to this stage we have a static application and an API exposing an interface to our database. The aim of this stage is to get our application to talk to our API.
Stage 5: Augment the App

Here we add the finishing touches. This could include:
• Using JQuery/Angular for a more responsive UX.
• Adding in session management and user authentication.
• Improving the presentation.
• Making the project easier to maintain, with tests, and templates.
• Using sockets for real-time interactivity
Planning the application

- Planning the app is best done with pen and paper.
- Sketch the different pages the user will go through.

This is important to get the flow of control right and understand how the app will be used.

Our app will be called Chortal, and it will be used for recording and managing a child’s chores.
Chortal Sketch
Setting up the project in Express

- The steps to set up the project are:
  1. Ensure node is installed and use npm to install the express generator:

```
$ npm install -g express-generator

/Users/tim/.npm-packages/bin/express -> /Users/tim/.npm-packages/lib/node_modules/express-generator/bin/express
/Users/tim/.npm-packages/lib
└── express-generator@4.13.1
```

2. Generate an express project:

```
$ express chortal

create : chortal
create : chortal/package.json
create : chortal/app.js
create : chortal/public
create : chortal/public/javascripts
create : chortal/public/stylesheets
create : chortal/public/stylesheets/style.css
create : chortal/public/images
create : chortal/routes
create : chortal/routes/index.js
create : chortal/routes/users.js

create : chortal/views
create : chortal/views/index.jade
create : chortal/views/layout.jade
create : chortal/views/error.jade
create : chortal/bin
create : chortal/bin/www

install dependencies:
  $ cd chortal && npm install

run the app:
  $ DEBUG=chortal:* npm start
```
The parts of our project: package.json

- package.json is all of the node modules the project uses.
- use npm install --save <new-module> to automatically have the package added to package.json
- package.json contains metadata about the project, and particularly all the dependencies.
- dependencies have indexes: major version; minor version; patch version.
- use ~ to match the latest patch version, (and there are various other wildcards too.)
```json
{
   "name": "chortal",
   "version": "0.0.0",
   "private": true,
   "scripts": {
      "start": "node ./bin/www"
   },
   "dependencies": {
      "body-parser": "^1.13.2",
      "cookie-parser": "^1.3.5",
      "debug": "^2.2.0",
      "express": "^4.13.1",
      "jade": "^1.11.0",
      "morgan": "^1.6.1",
      "serve-favicon": "^2.3.0"
   }
}
```
The parts of our project: app.js

- app.js contains the main code to run the application
- This is called by bin/www

The parts of our project: public

- the public folder contains all the public assets we use to display pages to the user, such as:
  - CSS
  - images
  - client side javascript
  - bootstrap themes
The parts of our project: routes

- the routes folder normally contains the server side javascript to respond to client requests. We’re going to change this a bit to use a model-view-controller architecture.

The parts of our project: views

- The views folder contains jade files for rendering the html responses sent to the user.
- This separates presentation from content.

The parts of our project: git

- Finally, use `git init` to make a git repo.
Running the project

• You can now run the app using `npm start`
• This will run the app and you can access it at `http://localhost:3000`

Express

Welcome to Express

• You can install the nodemon package with `npm`
• This package will detect changes in the source and restart the app automatically.
The Express Process

Express runs on the server, listening for requests, then uses javascript to create responses to send to the browser.
Model-View-Controller (MVC)

MVC is a design pattern for data driven applications:
1. A request comes into the application.
2. The request gets routed to a controller.
3. The controller, if necessary, makes a request to the model.
4. The model responds to the controller.
5. The controller sends a response to a view.
6. The view sends a response to the original requester.
MVC and express:

- To make express MVC we need to add controllers and models:
  1. make a new folder `app_server` and move the routes, and views directory there.
  2. add a models and controllers directory to `app_server`.
  3. Update `app.js` with the new routes location

```javascript
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var routes = require('./app_server/routes/index');
var users = require('./app_server/routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'app_server', 'views'));
app.set('view engine', 'jade');</code>
Adding controllers to routes

• By default, the routes contain the logic for handling requests

```javascript
1 var express = require('express');
2 var router = express.Router();
3
4 /* GET users listing. */
5 router.get('/', function(req, res, next) {
6    res.send('respond with a resource');
7 });
8
9 module.exports = router;
```

• Instead we get the routes file to call a controller

```javascript
1 var express = require('express');
2 var router = express.Router();
3 var ctrlMain = require('./controllers/main');
4
5 /* GET home page. */
6 router.get('/', ctrlMain.index);
7
8 module.exports = router;
```

• … and put the logic in the controller instead

```javascript
1 /*Get Home page*/
2 module.exports.index = function(req,res){
3    res.render('index',{title: 'Chortal'});
4 }
```
Making a view...

- `res.render(index, {title: 'Chortal'})` is an Express function that builds a HTML page from a Jade template and a Javascript object.

- Inside the views folder we’ll find `index.jade`:

```
extends layout
block content
  h1 title
  p Welcome to #{title}
```

- Which extends the generic template, `layout.jade`:

```
doctype html
html
  head
    title title
    link(rel='stylesheet', href='/stylesheets/style.css')
body
  block content
```
Next lecture…

- We’ll look at building Jade views and linking them to routes….