THE UNIVERSITY OF WESTERN AUSTRALIA

Topic 10: Develop in the MEAN Architecture

CITS3403 Agile Web Development

Getting MEAN with Mongo, Express, Angular and Node, Chapter 3.

Semester 1, 2018

Agile Project Development

THE UNIVERSITY OF WESTERN AUSTRALIA

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:

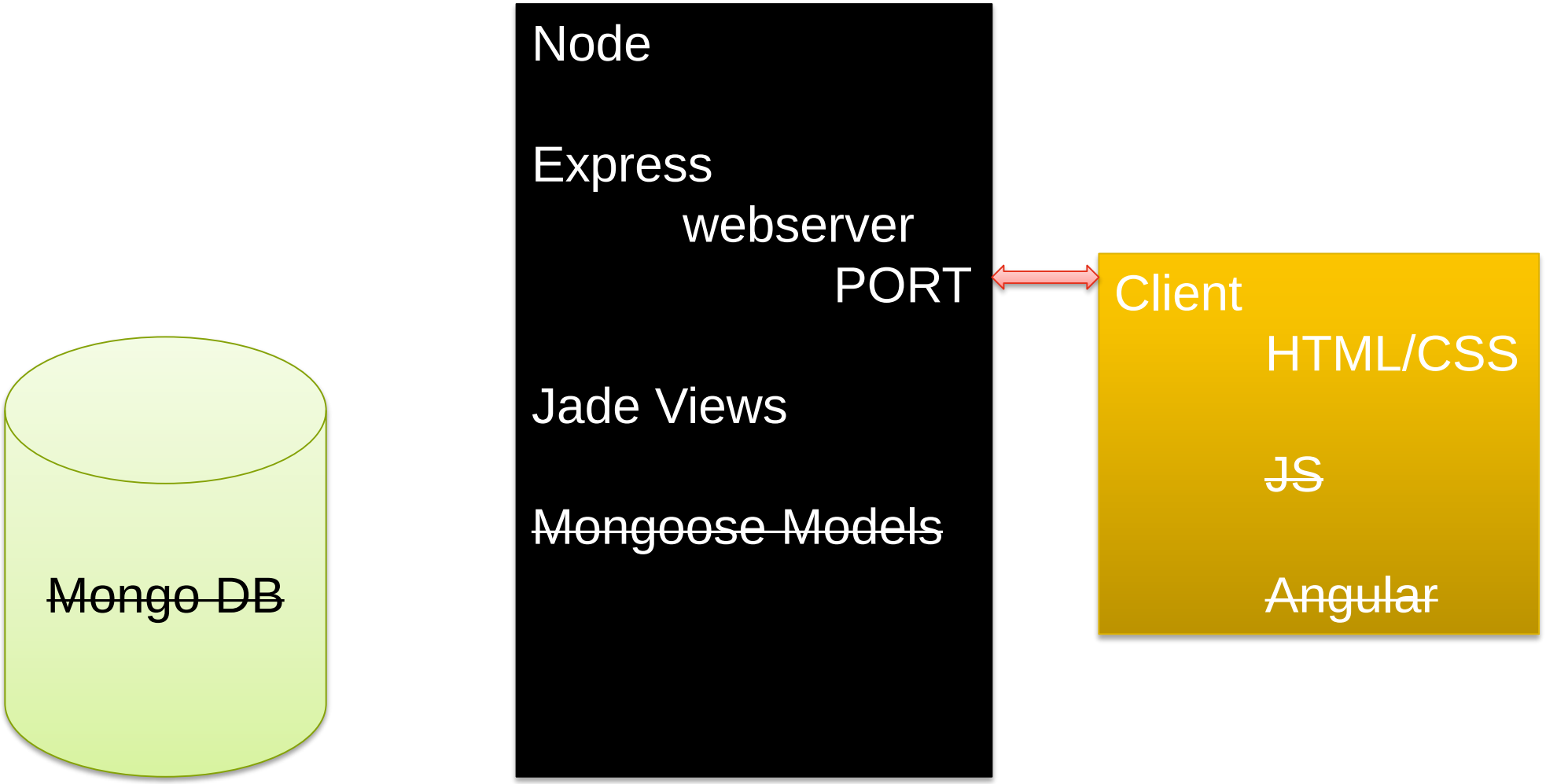
1. Build a static site
2. Design the data model and create a database
3. Build a data API
4. Hook the database into the application
5. Augment the application

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

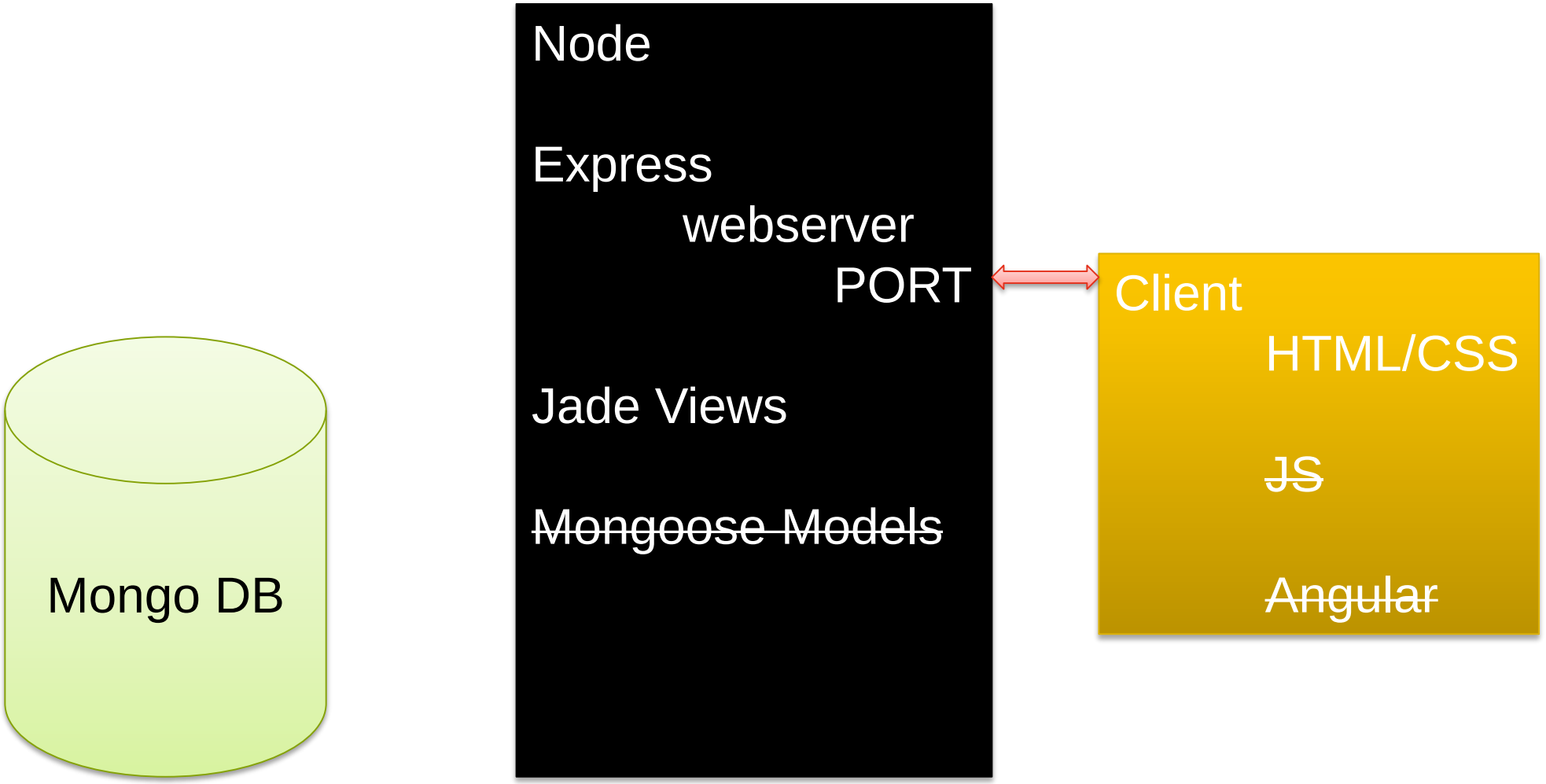


Stage 2. Design the data model

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

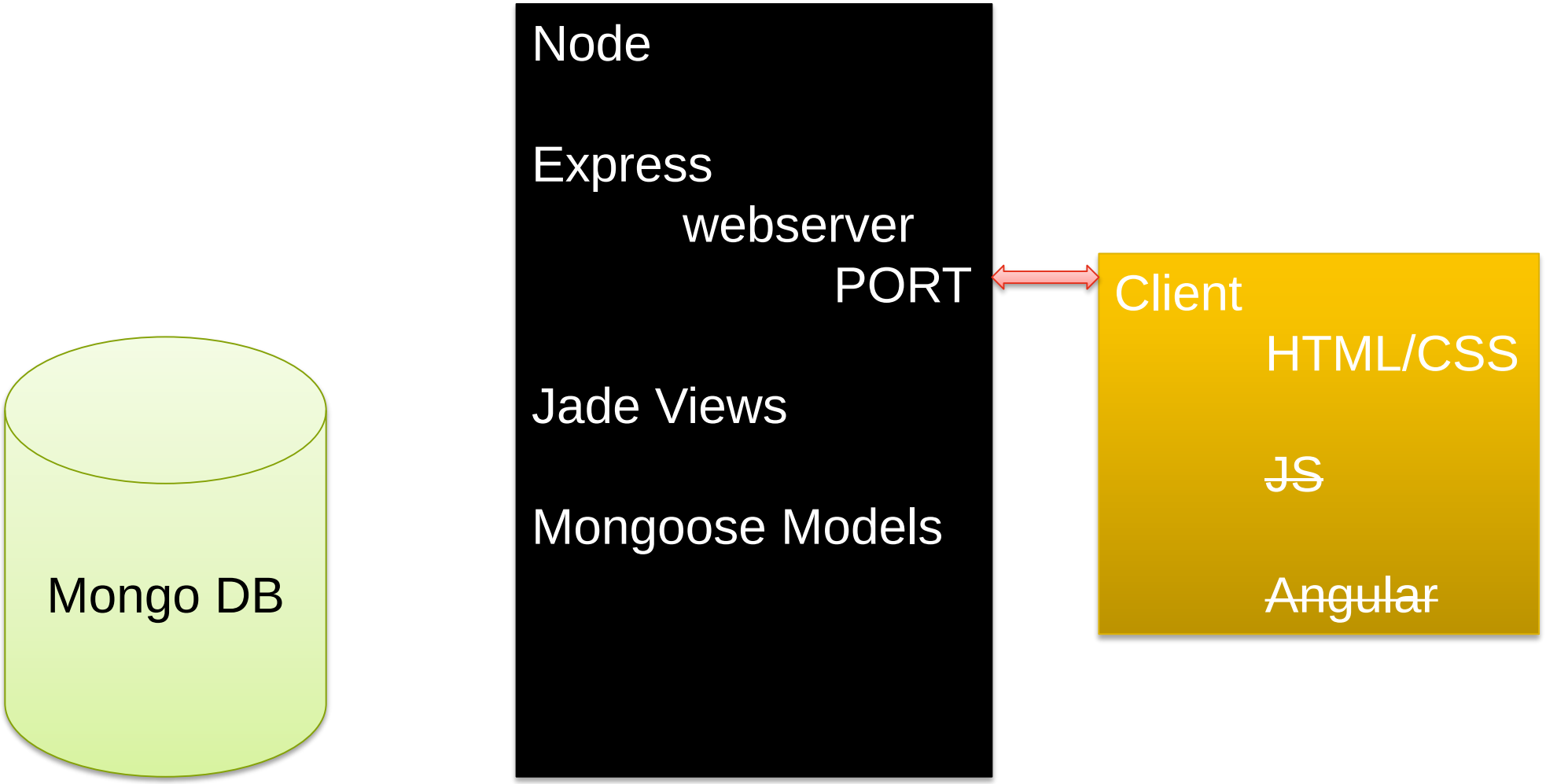


Stage 3: Build the data API

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.

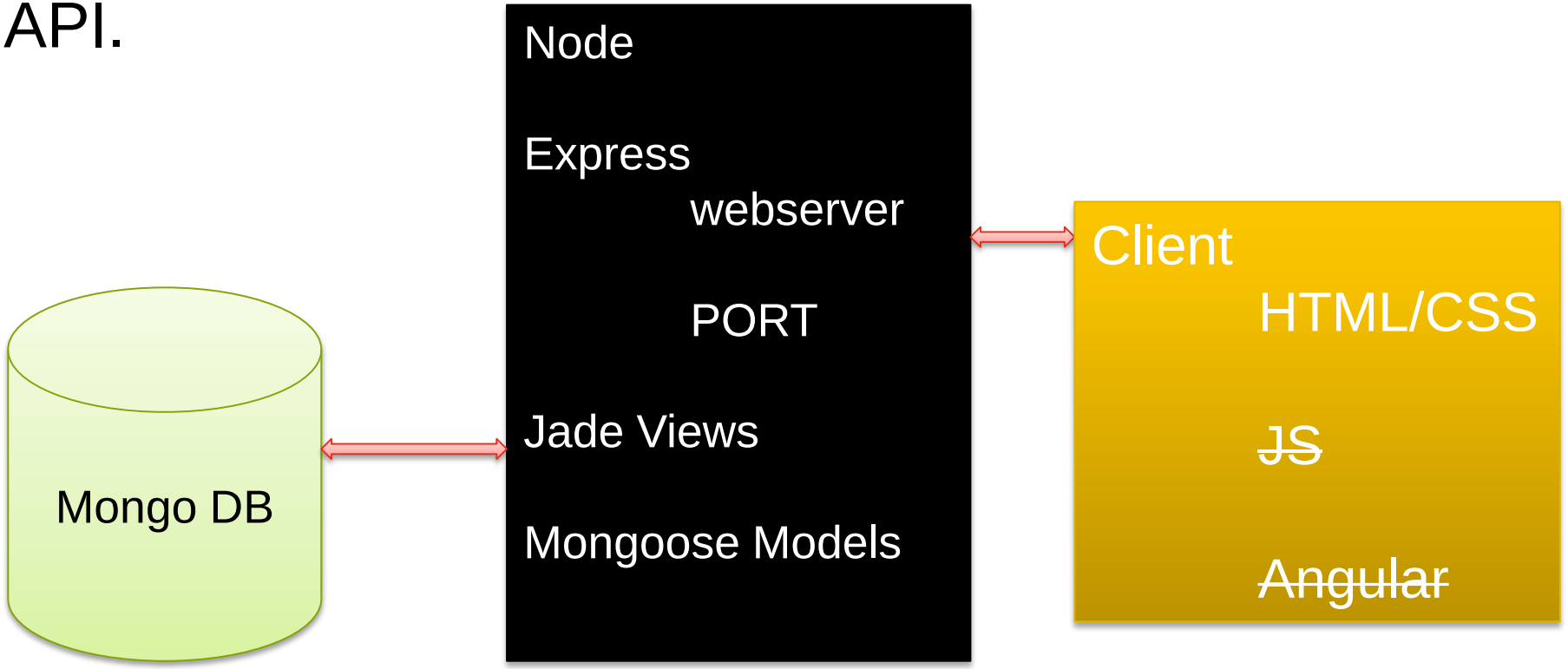
To do this we create a REST (REpresentative STate) API that will allow our application to interact with the database.

A simple MEAN architecture



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

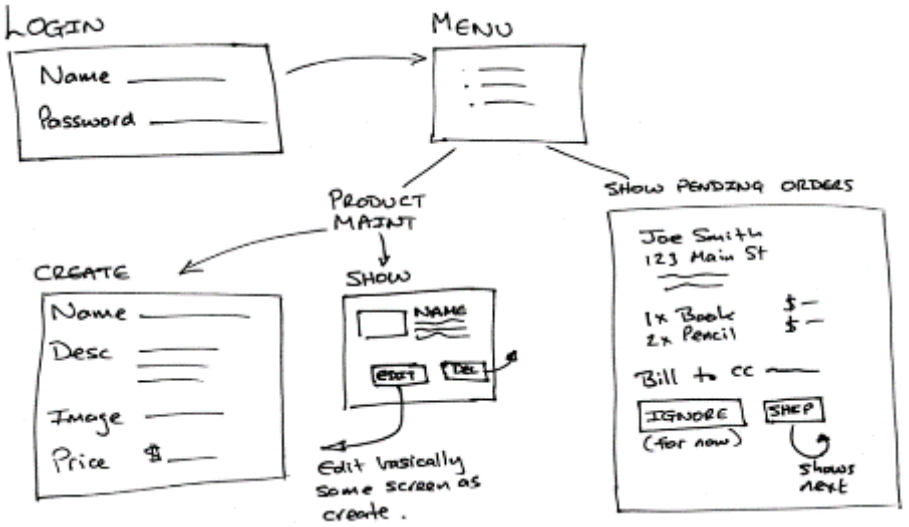
Here we add the finishing touches. This could include:

- 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.



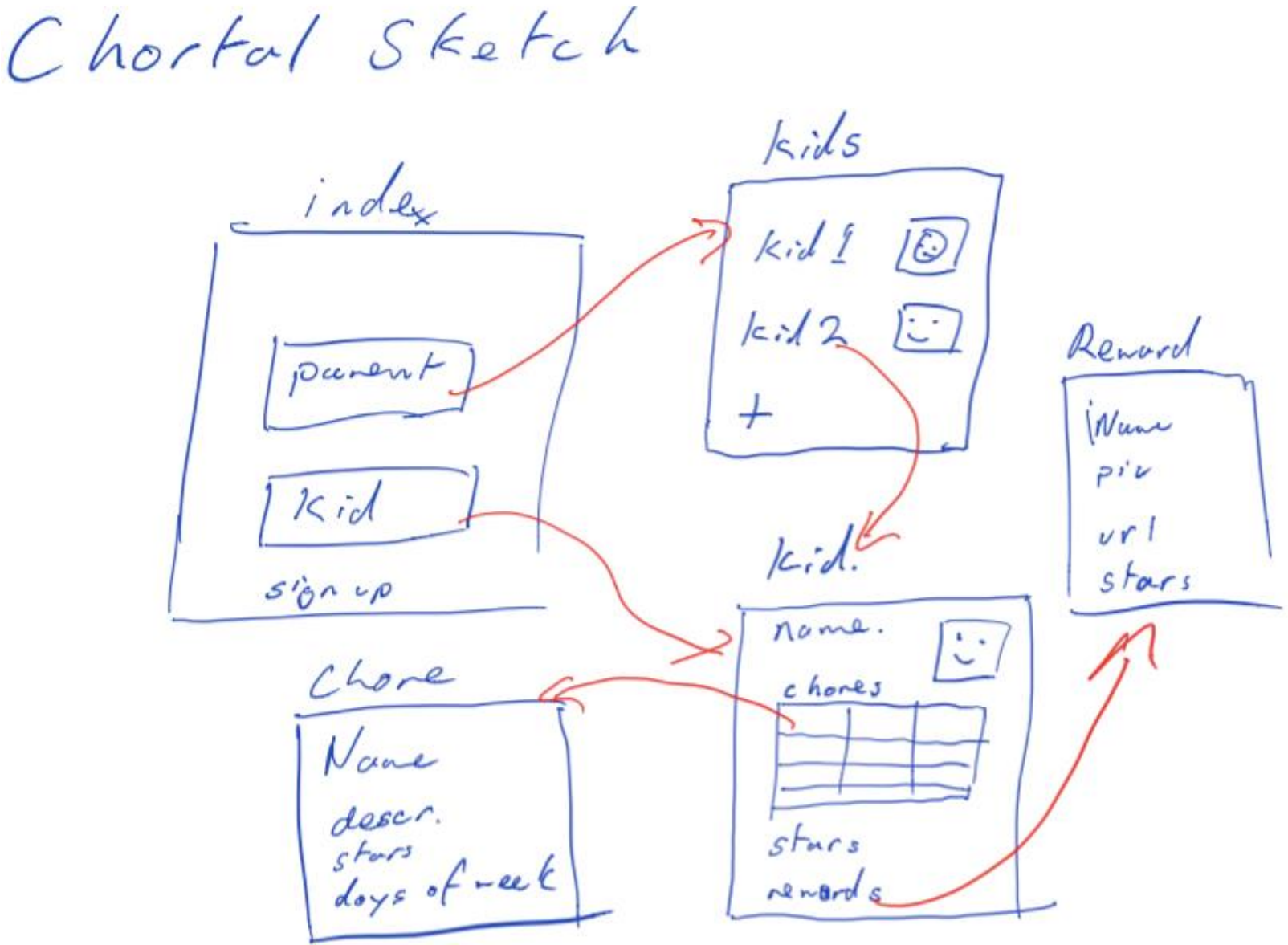
Our app will be call *Chortal*, and it will be used for recording and managing a child's chores.

A basic project

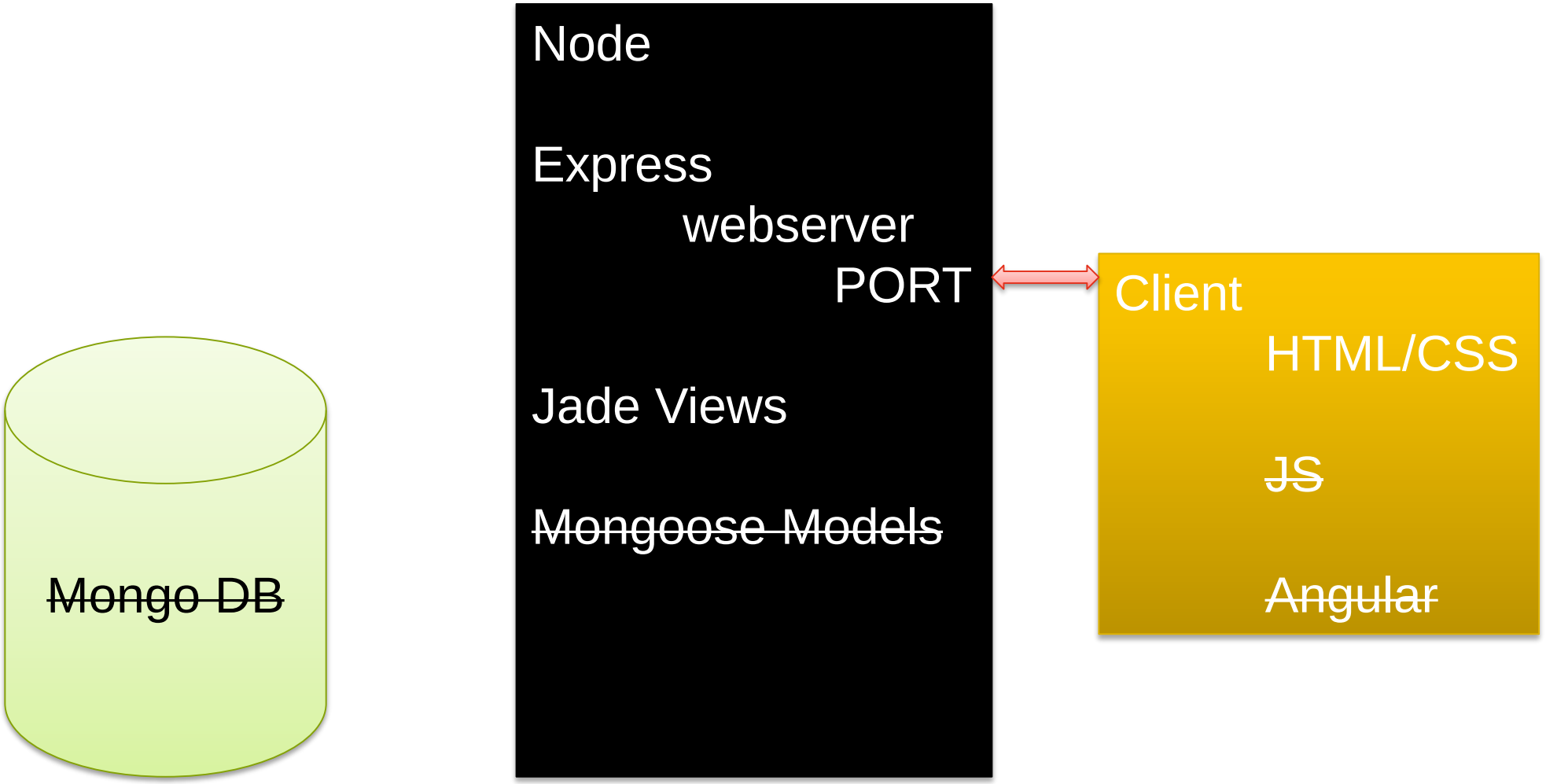
THE UNIVERSITY OF WESTERN AUSTRALIA

- The project is to build a web application that can allow children to record their chores, and allow parents to track rewards for children.
- There are two users (parents and children), and several entities (parents, children, chores, rewards).
- In Agile development, the process is often broken up into user stories.
- User stories are short descriptions of features from the perspective of a user:
  - *As a Parent I want to set a daily task for my child so that they are reminded of it everyday.*
- *See https://www.mountaingoatsoftware.com/agile/user-stories*

Chortal Sketch



A simple MEAN architecture



Setting up the project in Express

THE UNIVERSITY OF WESTERN AUSTRALIA

- The steps to set up the project are:

1. Ensure node is installed and use npm to install the express generator:

```
TMBA:Chortal tim$ 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:

```
C:\Users\wei>express chortal

  warning: the default view engine will not be jade in future releases
  warning: use `--view=jade' or `--help' for additional options


   create : chortal
   create : chortal/package.json
   create : chortal/app.js
   create : chortal/views
   create : chortal/views/index.jade
   create : chortal/views/layout.jade
   create : chortal/views/error.jade
   create : chortal/public
   create : chortal/routes
   create : chortal/routes/index.js
   create : chortal/routes/users.js
   create : chortal/bin
   create : chortal/bin/www
   create : chortal/public/images
   create : chortal/public/javascripts
   create : chortal/public/stylesheets
   create : chortal/public/stylesheets/style.css

install dependencies:
  > cd chortal && npm install

run the app:
  > SET DEBUG=chortal:* & npm start
```

The parts of our project: package.json

THE UNIVERSITY OF WESTERN AUSTRALIA

- `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.

package.json

Windows only. Use `less`, `more` or `cat` for MacOS or Linux

```
C:\Users\wei\chortal>type package.json
{
  "name": "chortal",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "body-parser": "~1.18.2",
    "cookie-parser": "~1.4.3",
    "debug": "~2.6.9",
    "express": "~4.15.5",
    "jade": "~1.11.0",
    "morgan": "~1.9.0",
    "serve-favicon": "~2.4.5"
  }
}
```

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

```
TMBA:public tim$ ls -R
images          javascripts     stylesheets

./images:

./javascripts:

./stylesheets:
style.css
```

```
C:\Users\wei\chortal\public>dir /s /b
C:\Users\wei\chortal\public\images
C:\Users\wei\chortal\public\javascripts
C:\Users\wei\chortal\public\stylesheets
C:\Users\wei\chortal\public\stylesheets\style.css
```

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.

```
C:\Users\wei\chortal>git init
Initialized empty Git repository in C:/Users/wei/chortal/.git/
```

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.

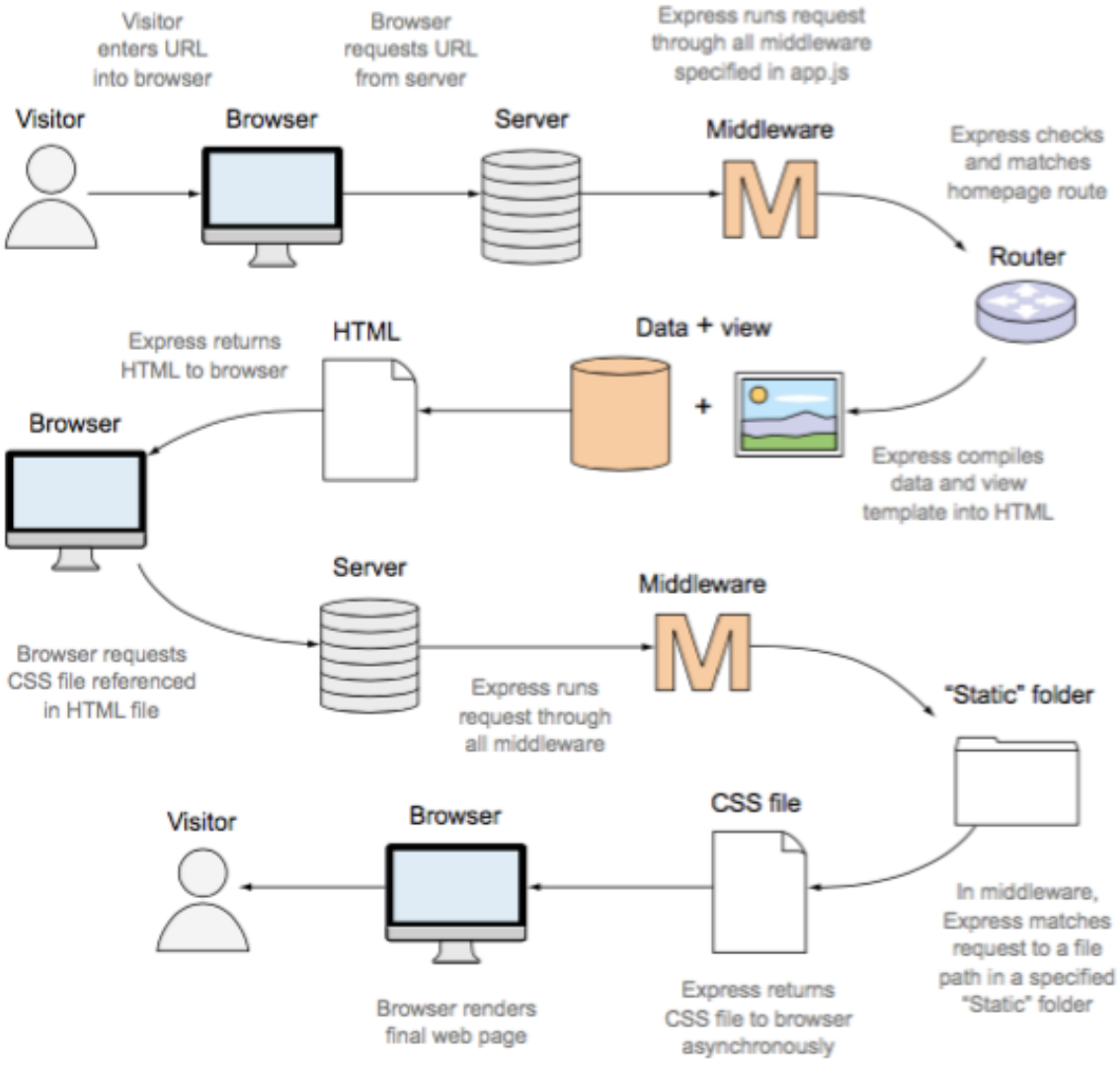


Figure 3.4 The key interactions and processes that Express goes through when responding to the request for the default landing page. The HTML page is processed by Node to compile data and a view template, and the CSS file is served asynchronously from a static folder.

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.

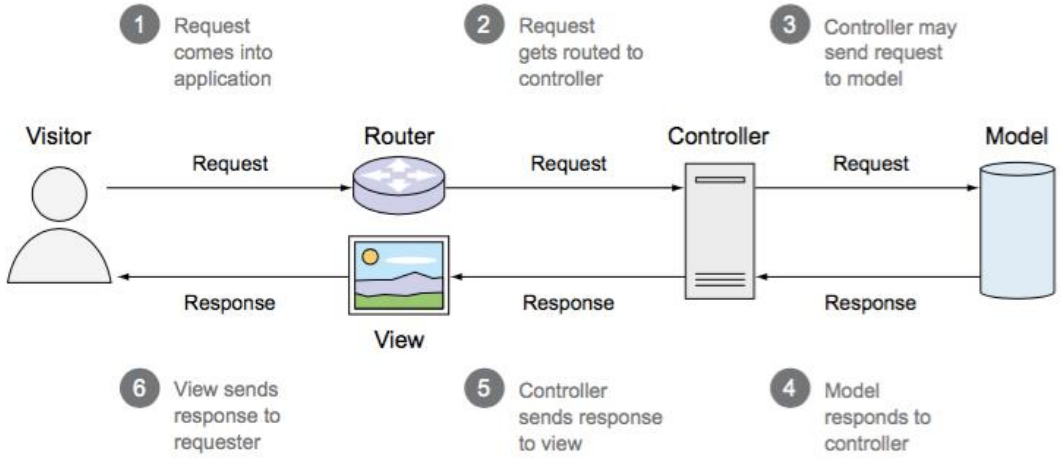


Figure 3.5 Request–response flow of a basic MVC architecture

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

```
1 var express = require('express');
2 var path = require('path');
3 var favicon = require('serve-favicon');
4 var logger = require('morgan');
5 var cookieParser = require('cookie-parser');
6 var bodyParser = require('body-parser');
7
8 var routes = require('./app_server/routes/index');
9 var users = require('./app_server/routes/users');
10
11 var app = express();
12
13 // view engine setup
14 app.set('views', path.join(__dirname, 'app_server', 'views'));
15 app.set('view engine', 'jade');
16
```

Adding controllers to routes

THE UNIVERSITY OF WESTERN AUSTRALIA

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

```
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

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

- … and put the logic in the controller instead

```
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:

```
1  extends layout
2
3  block content
4    h1= title
5    p Welcome to #{title}
6
```

  - Which extends the generic template, layout.jade

```
1  doctype html
2  html
3    head
4      title= title
5      link(rel='stylesheet', href='/stylesheets/style.css')
6    body
7      block content
8
```

  - Note, the url's given are relative to the public directory.

Next …

THE UNIVERSITY OF WESTERN AUSTRALIA

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

Adding controllers

- For each collection (e.g minion) we can add a controller (minion.js) in the controller directory:

```
1 /*Get Minion page*/
2 module.exports.minionList = function(req,res){
3   res.render('index',{title: 'Chortal'});
4 };
5
6
7 module.exports.minion = function(req,res){
8   res.render('index',{title: 'Minion'});
9 };
</2016/CITS3403/Chortal/app_server/controllers/minion.js
```

- This exports two functions: minionList and minion.

- We call these functions in routes/index.js to respond to requests from the browser.

```
1 var express = require('express');
2 var router = express.Router();
3 var ctrlMinion = require('../controllers/minion');
4 var ctrlTask = require('../controllers/task');
5 var ctrlReward = require('../controllers/reward');
6
7 /* GET home page. */
8 router.get('/', ctrlMinion.minionList);
9
10 /*GET Minion pages*/
11 router.get('/minion', ctrlMinion.minion);
<ver/routes/index.js   CWD: /Users/tim/Dropbox/Tim/tea
```

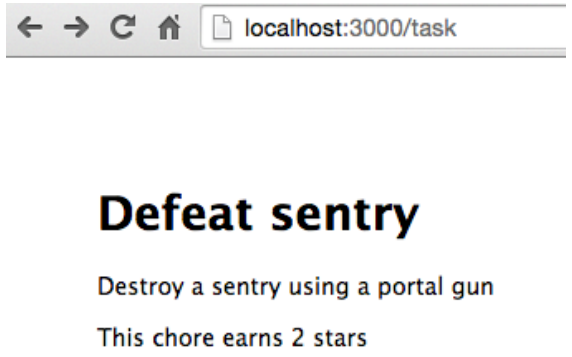
- We can then do the same for tasks, rewards and other models.

Adding views

- The `res.render` function takes a view and populates it with data.
- We can write a view for each controller…

```
1 extends layout
2
3 block content
4
5   h1= name
6   p= description
7   p This chore earns #{value} stars
8
9
<al/app_server/controllers   Line: 1 of 9 Col: 1
```

```
1 /*Get Task page*/
2 module.exports.taskInfo = function(req,res){
3   res.render('task',{name: 'Defeat sentry',
  description: 'Destroy a sentry using a portal gun',
  value: 2});
4 };
~
~
~
<03/Chortal/app_server/controllers   Line: 1 of 4 Col: 1
```



- Pug can refer to the objects passed into the render function.
- See: http://jade-lang.com/ for a guide on using pug.

Pug (formerly Jade)

THE UNIVERSITY OF WESTERN AUSTRALIA

```
doctype html
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript').
      if (foo) {
         bar(1 + 5)
      }
  body
    h1 Jade - node template engine
    #container.col
      if youAreUsingJade
        p You are amazing
      else
        p Get on it!
      p.
        Jade is a terse and simple
        templating language with a
        strong focus on performance
        and powerful features.
```

Pug is a succinct, programmatic way of writing html.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Jade</title>
    <script type="text/javascript">
      if (foo) {
         bar(1 + 5)
      }
    </script>
  </head>
  <body>
    <h1>Jade - node template engine</h1>
    <div id="container" class="col">
      <p>You are amazing</p>
      <p>
        Jade is a terse and simple
        templating language with a
        strong focus on performance
        and powerful features.
      </p>
    </div>
  </body>
</html>
```

Other view engines

- You can vary the view engine in app.js
- Embedded JavaScript is a popular alternative as it keeps the html form.
- In ejs, <%...> is used to mix js in html. <%=…> will output text into the html stream.

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

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');

```

```
<h1><%= title %></h1>
<ul>
    <% for(var i=0; i<supplies.length; i++) { %>
        <li>
            <a href='supplies/<%= supplies[i] %>'>
                <%= supplies[i] %>
            </a>
        </li>
    <% } %>
</ul>
```

Adding style…

- You can add in bootstrap and jQuery if you like. They both should be added to the public directory in your project.
- You can also set up some css styles in the public/styles/ directory…

```
1  doctype html
2  html
3    head
4      title= title
5      link(rel='stylesheet', href='/bootstrap-3.3.6-dist/css/bootstrap.css')
6      link(rel='stylesheet', href='/styles/style.css')
7    body
8      block content
9
10     script(src='/javascripts/jqery-2.2.3.min.js')
11     script(src='/bootstrap-3.3.6-dist/js/bootstrap.min.js')
~
~
~
~/Dropbox/Tim/teaching/2016/CITS3403/Chortal/app_server/views/layout.jade    CWD:
```

```
1  body {
2    margin: 50px;
3    padding: 50px;
4    font: 14px "Lucida Grande", Helvetica, Arial, sans-serif;
5  }
6
7  a {
8    color: #00B7FF;
9  }
10
11 table.minions {
12   border: 1px solid black;
13   text-align: left;
14   background-color: blue;
15   color: orange;
16 }
<g/2016/CITS3403/Chortal/public/stylesheets    Line: 4 of 16 Col: 1
```

Adding a navigation bar

```
4     title= title
5     link(rel='stylesheet', href='/bootstrap-3.3.6-dist/css/bootstrap.css')
6     link(rel='stylesheet', href='/stylesheets/style.css')
7
8   body
9     .navbar.navbar-default
10      .container-fluid
11        .navbar-header
12          a(href='/') Chortal
13          button.navbar-toggle(type='button', data-toggle='collapse', data-target='#navbar-main')
14               span.icon-bar
15               span.icon-bar
16               span.icon-bar
17        #navbar-main.navbar-collapse.collapse
18          ul.nav.navbar-nav
19            li
20              a(href='/tasks/') Chores
21      .container
22        block content
23        footer
24          .row
25            .col-xs-12
26              small CITS3403 Agile Web Development
27
28    script(src='/javascripts/jqery-2.2.3.min.js')
29    script(src='/bootstrap-3.3.6-dist/js/bootstrap.min.js')
```