

Preparation

Mac/Linux

- Start Terminal
- `curl https://install.meteor.com/ | sh`
- Install NodeJS - <https://nodejs.org/en/download/>

Windows

- <https://install.meteor.com/windows>
- <https://nodejs.org/en/download/>
- Add NPM path to window env variables for command line to execute NPM - <http://stackoverflow.com/questions/27864040/fixing-npm-path-in-windows-8>

Technical Issues:

If you have NodeJS installed before installing MeteorJS.
Uninstall NodeJS first, then install MeteorJS
follow by installing the latest NodeJS.



NUS
National University
of Singapore



MeteorJS Framework

Make JavaScript work for You!

Slides by Nicholas Ooi

Major in NUS Information System - E-Commerce (last batch)

Assisted by Tyson Quek

Major in Computer Science

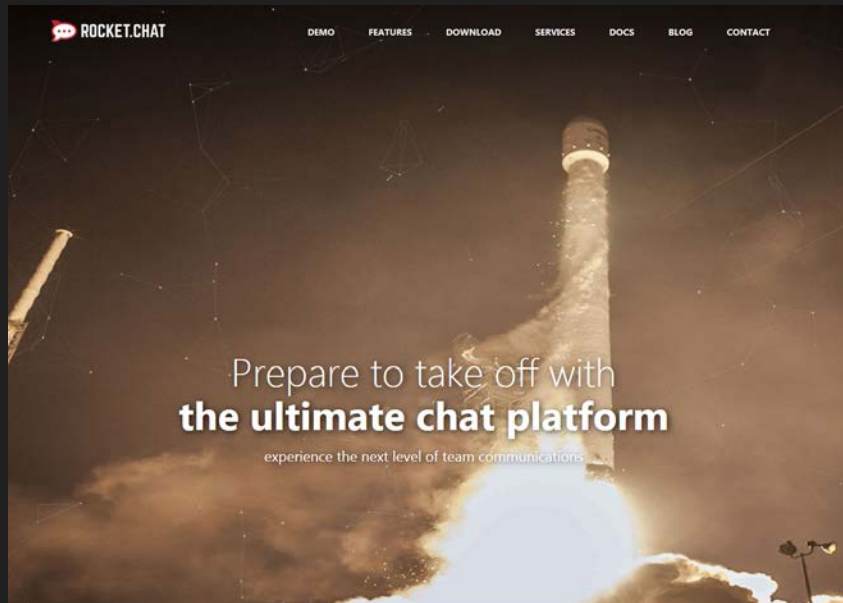
What is MeteorJS?

- MeteorJS is a **full stack JS(JavaScript) Framework**.
- MeteorJS empowers you to develop **frontend and backend** using JS.
- Write JS and **deploy everywhere**: web app, iOS, Android native apps.
- MeteorJS is known for its **real-time capabilities**. Useful for IoT (Internet of Things)
- MeteorJS framework contains **NodeJS** (as its server) and **MongoDB** (as its database)

Why MeteorJS?

- You want to write **one language** for both frontend and backend.
- You want to **deploy to almost any devices**.
- You want a **real-time effect** without having messy coding procedures.
- You want to build something **fast, rapid, and quick**.

Examples



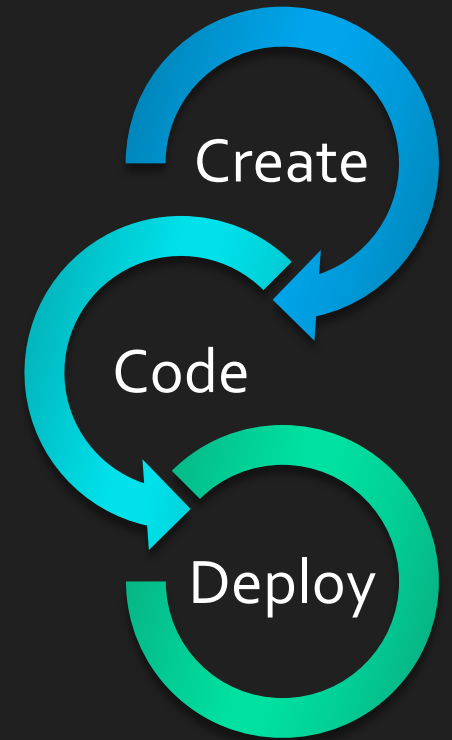
<https://rocket.chat/>



https://explore-uk.illustrstreets.com/app/crime-rates-in-england?view=ByJ2Qolhe&grid=ByJ2Qolhe&map=-0.2757*51.4752*11

How and what will we be learning?

- 1. We will be **creating** a **simple secure** feedback MeteorJS web app.
- 1. We will **code** both frontend and backend of the feedback app.
- 1. We will **use and manipulate data** in MongoDB (a NoSQL database).
- 1. We will **deploy** the app as a web app.
- 1. We will **try out** the feedback app.



Install MeteorJS

Mac/Linux

- Start Terminal
- `curl https://install.meteor.com/ | sh`
- Install NodeJS - <https://nodejs.org/en/download/>

Windows

- <https://install.meteor.com/windows>
- <https://nodejs.org/en/download/>
- Add NPM path to window env variables for command line to execute NPM - <http://stackoverflow.com/questions/27864040/fixing-npm-path-in-windows-8>

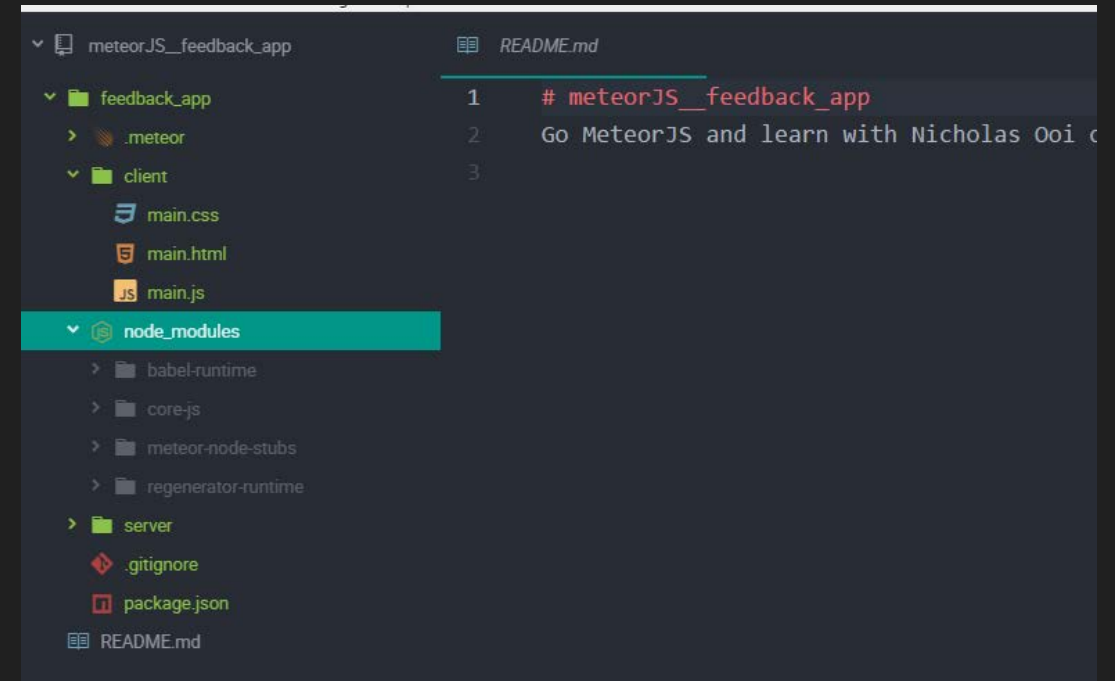
To get latest installation – visit <https://www.meteor.com/install>

Technical Issues:

If you have **NodeJS installed before installing MeteorJS.**
Uninstall NodeJS first, then install MeteorJS
follow by installing the latest NodeJS.

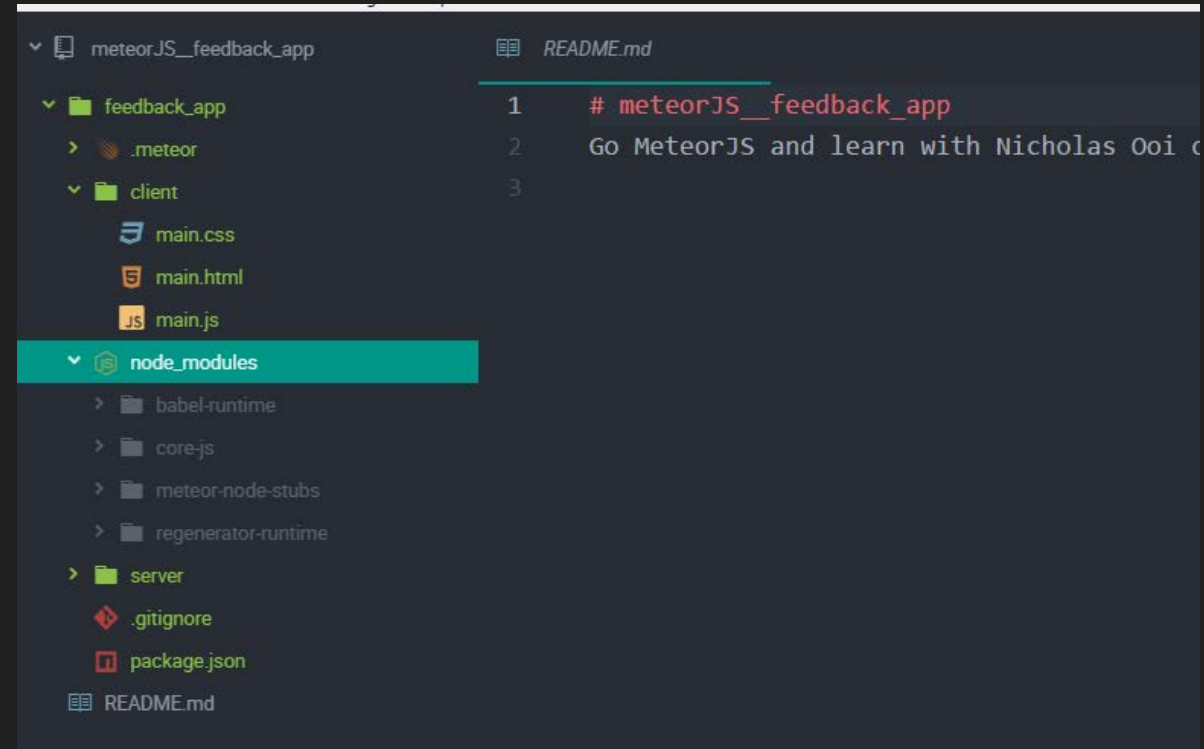
Creating MeteorJS

1. Launch command line.
2. Navigate to a folder you would like to create the project.
3. Type: `meteor create feedback_app`
4. Use IDE or any, open the feedback_app folder
5. You should see a list of **generated files**.
6. By default, MeteorJS **adds jQuery library**.



Creating MeteorJS

- 1. **.meteor folder** contains meteor installed libraries.
- 1. **Client folder** is where you code the **frontend**.
- 1. **Server folder** is where you code the **backend**.
- 1. **Node_modules folder** contains NPM installed libraries.
- 1. **.gitignore file** is for ignoring file when committing to repo.
- 1. **package.json** contains what NPM modules to be installed.



Frontend

1. MeteorJS compiles all HTML,CSS, JS and minify them.
2. It is a new form of programming unlike traditional ways, which you need to import script, css everywhere.
3. **MeteorJS once again, merges and compiles CSS/HTML/JS everything automatically but in certain order according to their standards!**

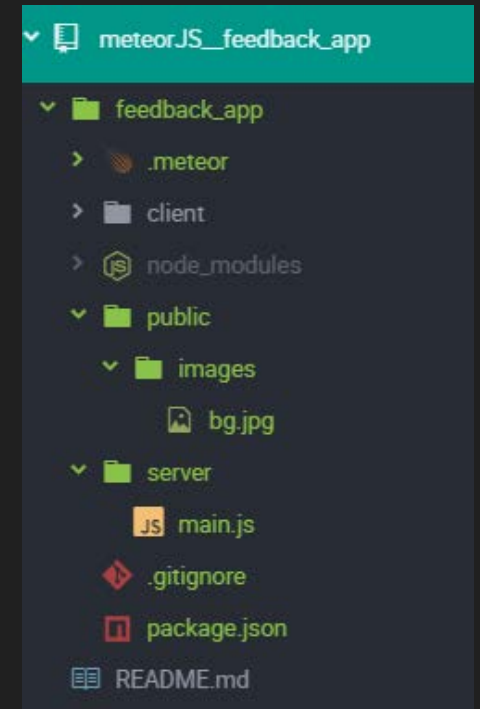
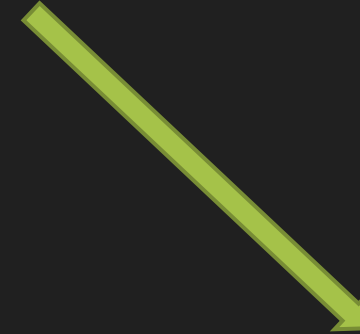
- 1.HTML template files are **always** loaded before everything else
- 2.Files beginning with main. are loaded **last**
- 3.Files inside **any** lib/ directory are loaded next
- 4.Files with deeper paths are loaded next
- 5.Files are then loaded in alphabetical order of the entire path

Standards

<https://guide.meteor.com/structure.html>

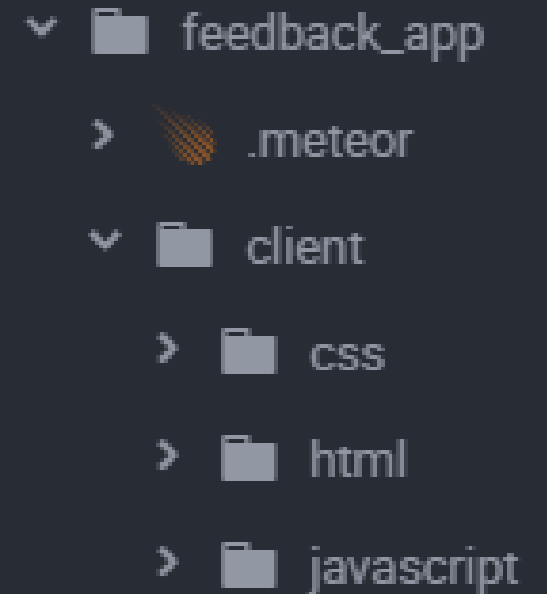
Frontend

1. Create a **folder called public** and a **subfolder named images**
2. The folder named public is a **special meteorJS folder** that allows files access to the public.
3. The images folder will contain all our images, you can categorize subfolders inside at your choice.
4. **Remove all files** in the client folder, we will create our own.



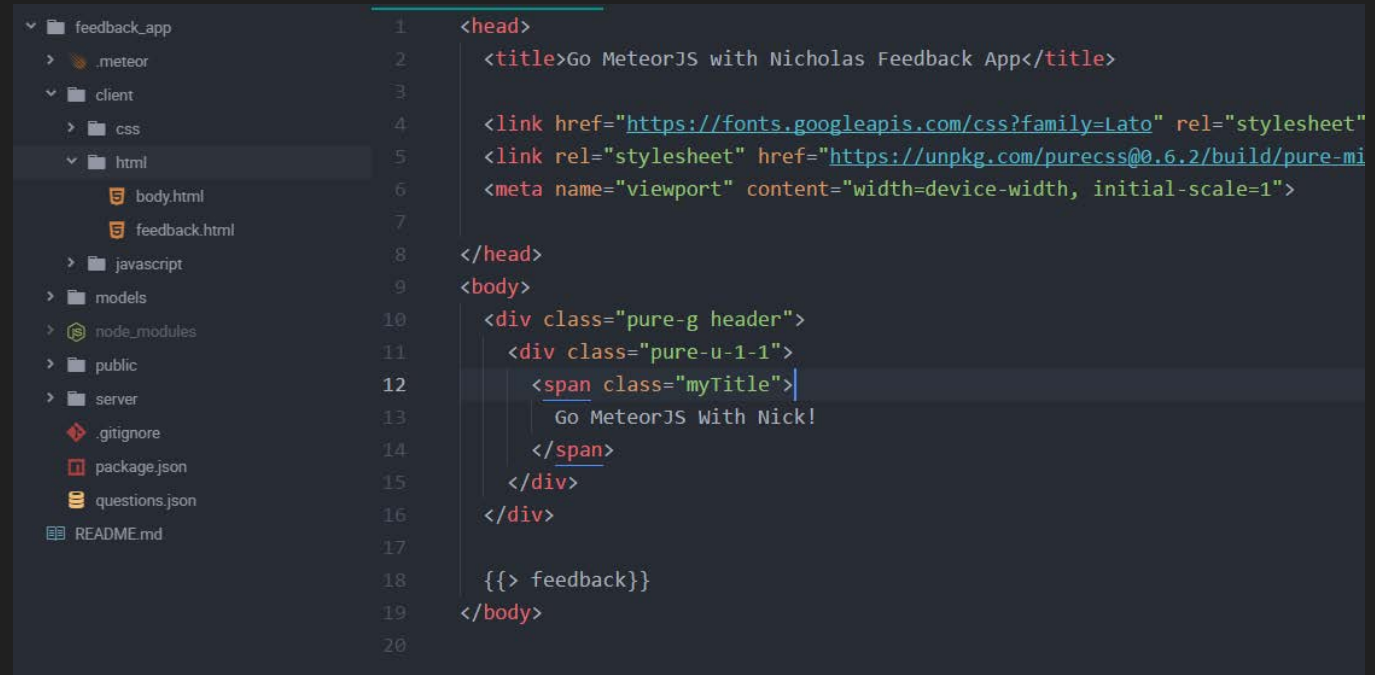
Frontend

1. We will create three folders in client named: **css, html and javascript**
2. We will place the respective files accordingly to the folders to keep it structured.
3. This will help us to keep track of files when the project scales larger in time.



Frontend

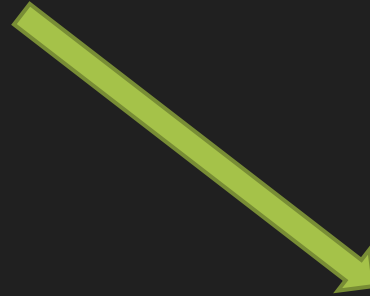
1. Inside html folder, we create **body.html**
2. Meteor is **smart enough** to identify only one file that has `<head>` and `<body>` tags as its **entry point**.
3. `body.html` will be our main placement of other scripts and css from external sources.
4. It will also be our layout structure.



```
1 <head>
2   <title>Go MeteorJS with Nicholas Feedback App</title>
3
4   <link href="https://fonts.googleapis.com/css?family=lato" rel="stylesheet"
5   <link rel="stylesheet" href="https://unpkg.com/purecss@0.6.2/build/pure-mi
6   <meta name="viewport" content="width=device-width, initial-scale=1">
7
8 </head>
9 <body>
10   <div class="pure-g header">
11     <div class="pure-u-1-1">
12       <span class="myTitle">
13         Go MeteorJS With Nick!
14       </span>
15     </div>
16   </div>
17
18   {{> feedback}}
19 </body>
20
```

Frontend

This will be the final feedback frontend design



What we will be learning from designing this in MeteorJS?

1. Wiring Meteor client JS with Meteor Blaze HTML
2. Creating Meteor Templates
3. Structuring Meteor Templates
4. Blaze HTML syntax: loop, if and else, output of data
5. Meteor Helpers, reactive data driven, events.
6. Meteor Forms.

Go MeteorJS With Nick!

Feedback App

How would you rate Nicholas Ooi as a teacher for this Orbital Tutorial?

- ☐ 1
☐ 2
☐ 3
☐ 4
☐ 5

How do you rate MeteorJS usefulness?

- ☐ 1
☐ 2
☐ 3
☐ 4
☐ 5

What is your gender?

- ☐ male
☐ female

Is there any comments that you would like to give?

Submit Feedback

Frontend - client/html/body.html

1. Add these external stuff at the head in body.html.
 2. This is just an example to show you that you can import any external sources!
 3. You can also add other meta tags for SEO keywords, viewport etc.
-
1. We will use the below that includes Font Lato, Pure CSS Framework.
 2. Add bootstrap, semantic-UI at your choice but do some research before adding!

```
<head>
  <title>Go MeteorJS with Nicholas Feedback App</title>

  <link href="https://fonts.googleapis.com/css?family=Lato" rel="stylesheet">
  <link rel="stylesheet" href="https://unpkg.com/purecss@0.6.2/build/pure-min.css"
  integrity="sha384-UQiGfs9ICog+LwheBSRCt1o5cbyKIHbwjWscjemyBMT9YCUMZffs6UqUTd0hObXD"
  crossorigin="anonymous">
  <meta name="viewport" content="width=device-width, initial-scale=1">

</head>
```

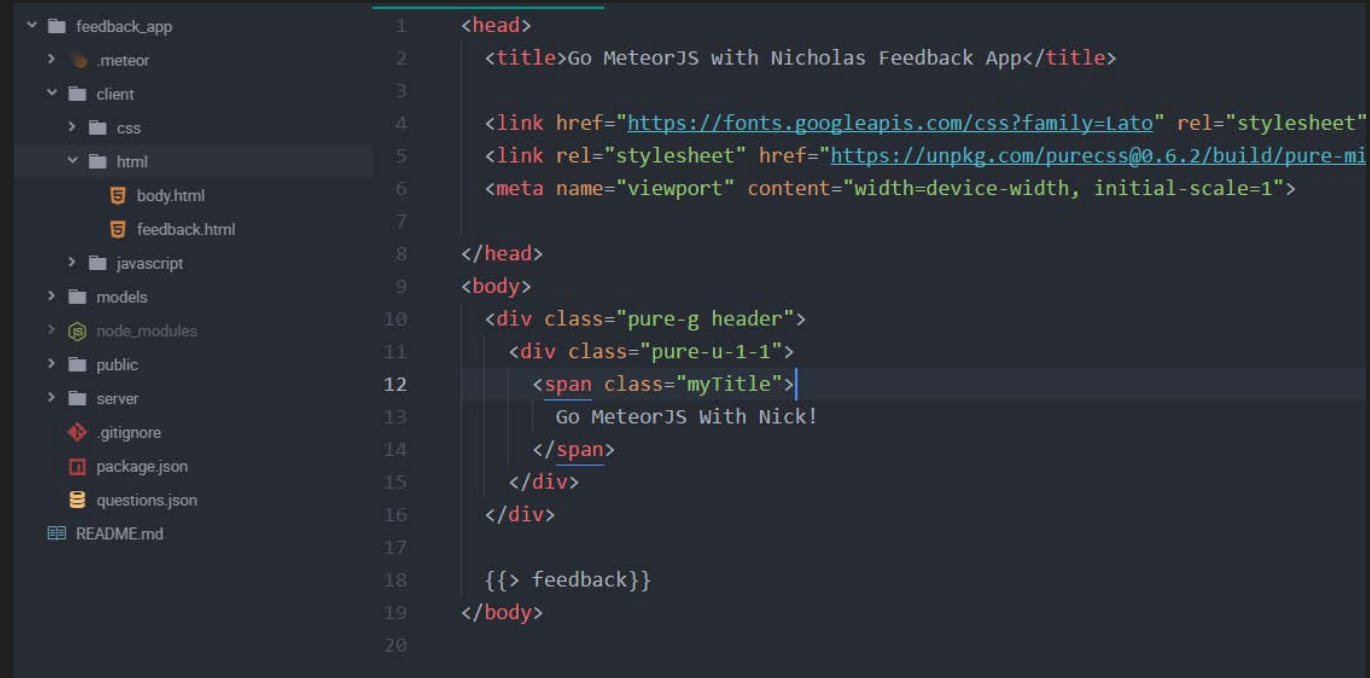
Frontend - client/html/body.html

1. Inside html folder, open **body.html**

Type:

```
<body>
  <div class="pure-g header">
    <div class="pure-u-1-1">
      <span class="myTitle">
        Go MeteorJS With Nick!
      </span>
    </div>
  </div>

  {{> feedback}}
</body>
```



1. `{{> feedback}}` this is to output a template, we will look into understanding template in the later slide.
2. Ignore pure-u-1-1 etc. those are pure CSS framework styling.

Frontend

1. Create a new file named **feedback.html** in html folder.
2. Create a new file named **feedback.js** in javascript folder.

In **feedback.html** file

Add the following code:

```
<template name="feedback">  
</template>
```

1. Think template as placeholder/container/invisible storage of html codes within it!
2. Template will be used to tie or wire or link up with the javascript later, **which you can control only within that template scope or space or within it only!**

Frontend

In **feedback.html** file,
Add the following code within the feedback <template> tag

```
<template name="feedback">

<div class="pure-g center">
  <div class="pure-u-1-3">
  </div>

  <div class="pure-u-1-3 feedbackBody">
    <span>
      {{myTitle}}
    </span>
  </div>

  <div class="pure-u-1-3">
  </div>
</div>

...
...
</template>
```

1. The pure-g center, pure etc. those are CSS classes for styling, you can ignore them for now.
2. Focus on **{{myTitle}}** - this is how you call a function or attribute from javascript in Meteor to return a result.

```

1  <template name="feedback">
2    <div class="pure-g center">
3      <div class="pure-u-1-3">
4      </div>
5      <div class="pure-u-1-3 feedbackBody">
6        <span>
7          {{myTitle}}
8        </span>
9        <div class="questionBody">
10         <form id="feedbackForm" class="pure-form">
11           <ul>
12             {{#each questions}}
13               <li>
14                 <h4>
15                   {{text}}
16                 </h4>
17                 {{#if checkType type}}
18                 <ul>
19                   {{#each choices}}
20                     <li>

```


Frontend

Add the following code in `client/javascript/feedback.js`

```
import { Meteor } from 'meteor/meteor';

Template.feedback.helpers({
  myTitle: function() {
    return "Feedback App";
  },

  checkType: function(type) {
    return type == "mcq";
  }
});
```

Template.feedback.helpers

This is where **you place your functions/methods** to be called
Only within the feedback template.

myTitle is a function that returns a string.

checkType is a function that takes in a parameter to check
and return you a Boolean.

Frontend

Add the following code in `client/javascript/feedback.js`

```
Template.feedback.onCreated(function() {  
  
});
```

Template.feedback.onCreated

This is a callback handler that will be triggered when the feedback Template is created from a request such as `{{ > feedback }}`. This handler will then be triggered and codes within it will execute, usually this is the entry point of the template.

Frontend

Add the following code in `client/javascript/feedback.js`

```
Template.feedback.events({  
  
});
```

Template.feedback.events

This is where you place event driven codes such as click, submit of form, onChange input when someone types, etc.

Frontend - client/css/feedback.css

In css folder create **feedback.css** and add the following code.

```
body {
  background: url("/images/bg.jpg") no-repeat;
  background-size: cover;
  color: #fff;
  font-family: "Lato";
}

.header {
  text-align: center;
  margin-top: 25px;
}

.center {
  text-align: center;
}

.myTitle {
  font-size: 48px;
  font-weight: bold;
}

.feedbackBody {
  color: #333;
  background: #fff;
  border: 1px solid #eee;
  box-shadow: 0px 7px 25px 3px #aaa;
  padding-top: 20px;
  padding-bottom: 200px;
}
```

- Take note, bg.jpg is an image file placed in public/images folder
- To access it via CSS, you could simply type /images/bg.jpg
- To put it in SRC for img -> ****

This example is to show you how you can store and access images!

Realize we **did not link CSS** in our head or anywhere else! MeteorJS again Compiles and include everything as one file automatically.

Frontend

Deploy meteor and test result.

1. Open terminal
2. Chang Directory or CD to your meteor project folder

3. Type meteor

4. Launch your browser type <http://localhost:3000>
5. Meteor uses port 3000 by default, please unblock it or keep it available.
6. you might see an error! This is to show you that <div> at feedback.html is </div> and meteor helps you to catch it!

```
]]]]]

=> Started proxy.
=> Started MongoDB.
=> Started your app.

=> App running at: http://localhost:3000/
    Type Control-C twice to stop.
```

Frontend

You should see the result at the right

“Feedback App” is the String we called From our **myTitle** function called in feedback.js.

The background image you can replace and renamed it as bg.jpg and place Into public/images folder.



Frontend - feedback.html

```
<!-- start of feedbackBody -->
<div class="pure-u-1-3 feedbackBody">
  <span>
    {{myTitle}}
  </span>
  <div class="questionBody">
    <form id="feedbackForm" class="pure-form">
      <ul>
        {{#each questions}}
        <li>
          <h4>
            {{text}}
          </h4>
          {{#if checkType type}}
          <ul>
            {{#each choices}}
            <li>
              <label class="pure-radio">
                <input name="{{../_id._str}}" type="radio" value="{{text}}" />
                {{text}}
              </label>
            </li>
            {{/each}}
          </ul>
          {{else}}
          <textarea name="{{_id._str}}"></textarea>
          {{/if}}
        </li>
        {{/each}}
      </ul>
      <input class="pure-button" type="submit" value="Submit Feedback" />
    </form>
  </div>
</div>
<!-- end of feedbackBody -->
```

Add the following code into `client/html/feedback.html`

1. This will create our layout for a list of questions and choices.
2. Also to check if the question is either MCQ or freetext.
3. **If its freetext**, it will show a huge textbox (textarea)
4. Otherwise show a list of choices that is related to the question.

Frontend - feedback.html

```
<!-- start of feedbackBody -->
<div class="pure-u-1-3 feedbackBody">
  <span>
    {{myTitle}}
  </span>
  <div class="questionBody">
    <form id="feedbackForm" class="pure-form">
      <ul>
        {{#each questions}}
        <li>
          <h4>
            {{text}}
          </h4>
          {{#if checkType type}}
          <ul>
            {{#each choices}}
            <li>
              <label class="pure-radio">
                <input name="{{../_id._str}}" type="radio" value="{{text}}" />
                {{text}}
              </label>
            </li>
            {{/each}}
          </ul>
          {{else}}
          <textarea name="{{_id._str}}"></textarea>
          {{/if}}
        </li>
        {{/each}}
      </ul>
      <input class="pure-button" type="submit" value="Submit Feedback" />
    </form>
  </div>
</div>
<!-- end of feedbackBody -->
```

#each is a foreach loop or for loop that iterates from a collection.

#if is an if-else statement used for conditional checking.

#if checkType type

checkType is a javascript function that you will write later when tying up the html and javascript file together!

what the if statement does is:

The 1st question in the list of questions collection: please pass the **value in the type attribute** of the 1st question into the function **checkType** and return a Boolean.

_id refers to the ID of a row retrieved from mongoDB.

../_id is to get the id accessed from the parent loop because **_id** does not exist in the choices collection loop nor we want that **_id** of choices collection.

Frontend

```
.questionBody ul
{
  text-align: left;
  margin: 0px;
  padding: 15px;
}

.questionBody ul ul
{
  padding: 10px;
  padding-top: 0px;
}

.questionBody li
{
  list-style: none;
}

.questionBody textarea
{
  width: 100%;
  height: 100px;
}
```

Create a stylesheet - client/css/**question.css**
and add the following code

Backend

What we will be doing for Backend?

1. We will create a list of questions as a MongoDB collection.
1. We will add some data in MongoDB in the questions collection.
1. We will use **Meteor Call** to execute/call a function via the client with the server methods. Usually insert, update, delete of questions etc..
1. We will use **subscribe and publish** to call “getter” functions such as getQuestions from client and server.

Backend

1. We will remove two insecure packages.
 2. In terminal, type [meteor remove insecure](#)
 3. In terminal, type [meteor remove autopublish](#)
-
1. We do not want our client(user) to access our database via JS.
 2. Those two packages were added to allow client modification of the mongoDB database, this is **insecure!**

Backend - MongoDB

mongoDB is a noSQL database.

Tables are known as documents or collections.

mongoDB has a series of operations to control the data in a collection.

<https://gist.github.com/aponxi/4380516> - cheatsheet

Backend - MongoDB

It **does not** have any concept of relationships or data integrity capabilities like the ones found in a relational database management system (e.g. MySQL, Microsoft SQL, etc.)

Backend - models/questions.js

Create a **models** directory in the project root – feedback_app/**models/**

Create a javascript file in the model directory - feedback_app/models/**questions.js**

Add the following code in **questions.js**

```
import { Mongo } from 'meteor/mongo';  
  
export const Questions = new Mongo.Collection('questions');
```

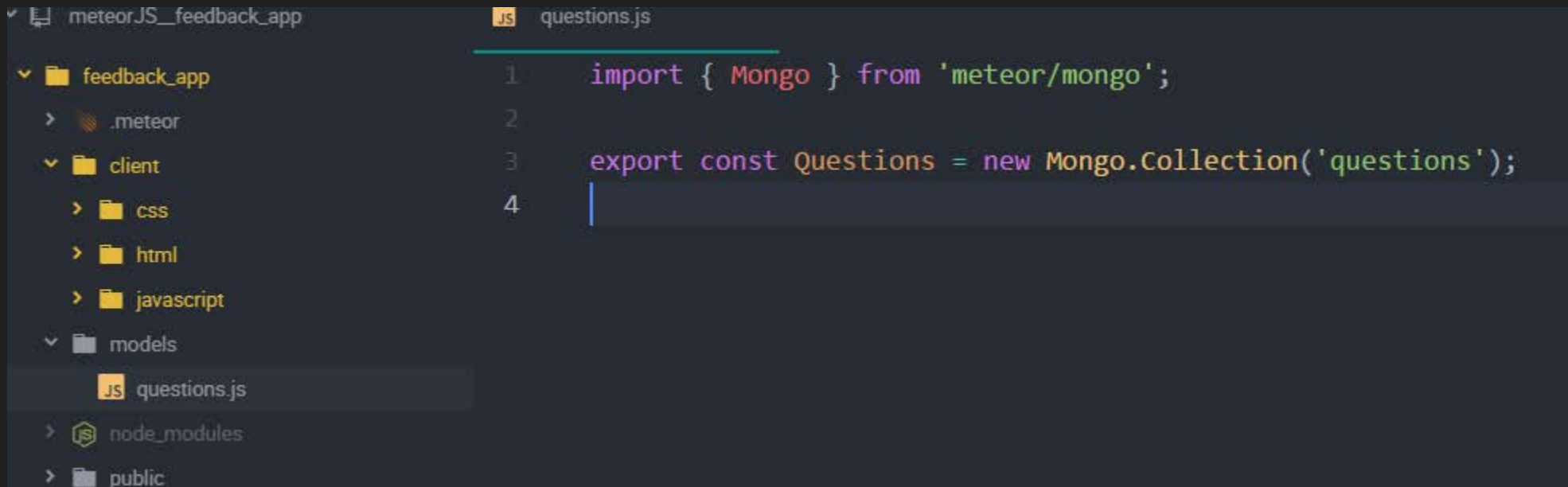
Import { Mongo} from 'meteor/mongo';

This is how you import mongoDB features to access the database or manipulate the data.

export const Questions = new Mongo.Collection('questions');

This exposes the mongoDB collection so that it can be accessed across the entire MeteorJS application.

Backend - MongoDB



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the following structure:

- meteorJS_feedback_app
 - feedback_app
 - .meteor
 - client
 - css
 - html
 - javascript
 - models
 - questions.js
 - node_modules
 - public

The code editor shows the content of the `questions.js` file:

```
1 import { Mongo } from 'meteor/mongo';
2
3 export const Questions = new Mongo.Collection('questions');
4
```

Backend - server/main.js

Open **main.js** in server folder.

Add the following code:

```
Meteor.startup(() => {  
  Meteor.methods({  
  });  
  
  Meteor.publish('getQuestions', function() {  
  });  
});
```

Meteor.startup()

This function will be triggered when meteor runs.

Meteor.methods()

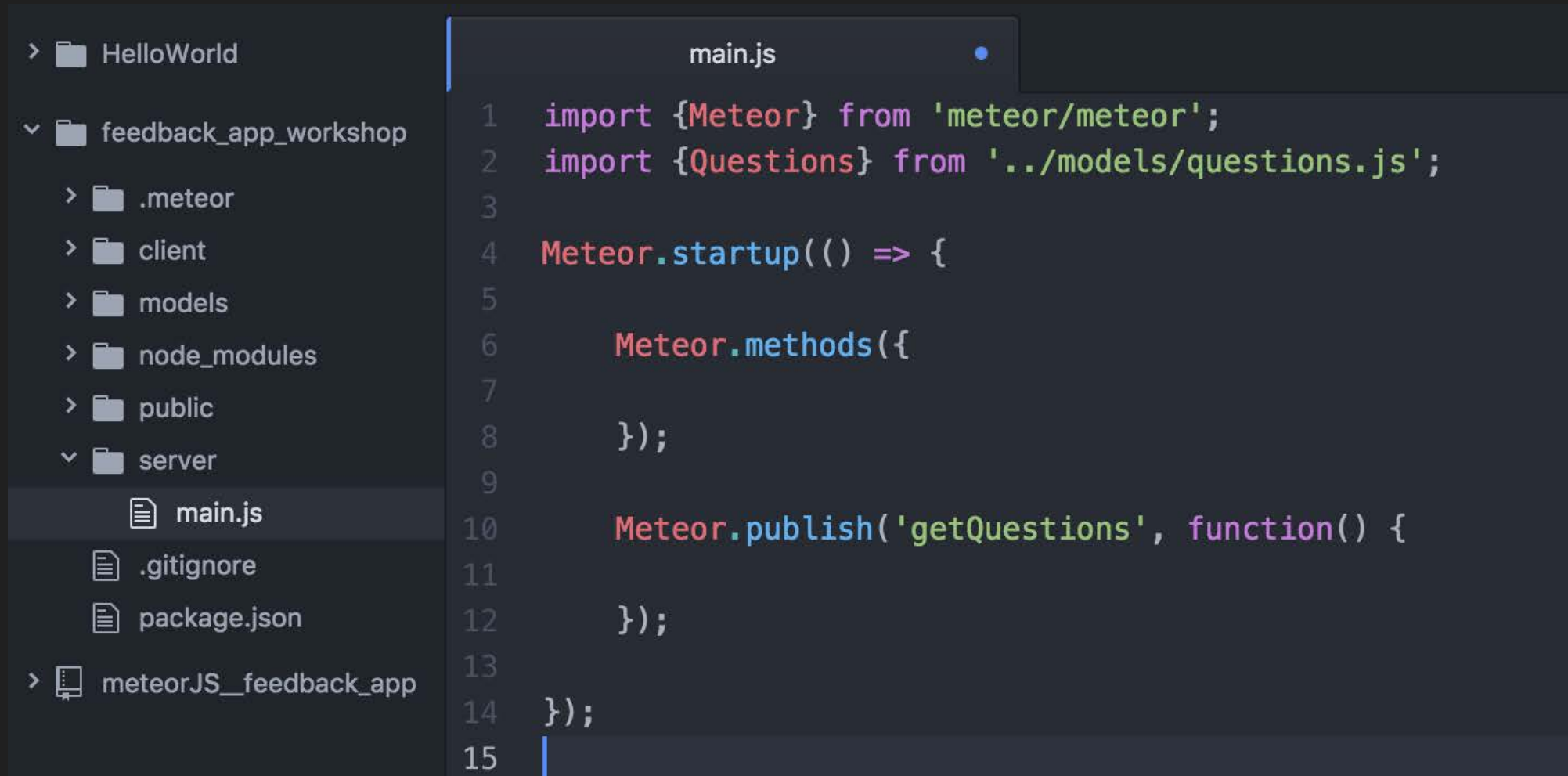
This is where you define all your functions to be executed from the server.

Meteor.publish()

This is where you define a name to a function to retrieve data.
Usually publish is used for retrieval of data into a collection. To achieve the **real-time effect or reactivity**. When you change data in MongoDB, it will push it to all the clients!

*Note: Files in the server folder **cannot be accessed** by importing it into the client.*

Backend



```
main.js
1  import {Meteor} from 'meteor/meteor';
2  import {Questions} from '../models/questions.js';
3
4  Meteor.startup(() => {
5
6      Meteor.methods({
7
8      });
9
10     Meteor.publish('getQuestions', function() {
11
12     });
13
14 });
15
```

Backend - server/main.js

In server/main.js

Add the following code in `Meteor.methods`:


```
'addAnswers':function(formData) {  
  try {  
  
    for(let i = 0, len = formData.length; i < len ; i++) {  
      let answer = formData[i].value;  
      const qid = new  
Meteor.Collection.ObjectID(formData[i].name);  
  
      Questions.update({_id: qid}, {$push: { result: answer }});  
    }  
  
    return true;  
  
  } catch (e) {  
    return new Meteor.Error(e);  
  }  
},
```


addAnswers












This is a function we create so we can add answers to a question, we loop the formData. We will convert the formData[i].name to an objectID, which we retrieved from the client side at [client/javascript/feedback.js](#)

A try and catch error, if anything goes wrong, throw a meteor error.

`Questions.update` is a mongoDB collection update operation. We find the QuestionID in the collection and push an answer to its result array.

>  HelloWorld

▼  feedback_app_workshop

- >  .meteor
- >  client
- >  models
- >  node_modules
- >  public
- ▼  server
 -  main.js
 -  .DS_Store
 -  .gitignore
 -  package.json
- >  meteorJS_feedback_app

```
main.js
1  import { Meteor } from 'meteor/meteor';
2  import { Questions } from '../models/questions.js';
3
4  Meteor.startup(() => {
5
6    Meteor.methods({
7      'addAnswers':function(formData) {
8        try {
9          for(let i = 0, len = formData.length; i < len ; i++)
10           {
11             let answer = formData[i].value;
12             const qid = new Meteor.Collection.ObjectID(formData[i].name);
13             Questions.update({_id: qid}, {$push: { result: answer }});
14           }
15           return true;
16         } catch (e) {
17           return new Meteor.Error(e);
18         }
19       },
20     });
21
22
23     Meteor.publish('getQuestions', function() {
24
25     });
26
27
28   });
```


Backend - server/main.js


In server/main.js












Add the following code in `Meteor.publish`:

```
Meteor.publish('getQuestions', function() {  
  return Questions.find({});  
});
```

In `Meteor.publish`, we create a `getQuestions` method. we want to **allow** all data to be retrieved for any client that subscribe it. This is where you control what data to be pushed to the client. **The ultimate data controller.**

>  HelloWorld

▼  feedback_app_workshop

- >  .meteor
- >  client
- >  models
- >  node_modules
- >  public
- ▼  server
 -  main.js
 -  .DS_Store
 -  .gitignore
 -  package.json
- >  meteorJS__feedback_app

main.js

```
1  import { Meteor } from 'meteor/meteor';
2  import { Questions } from '../models/questions.js';
3
4  Meteor.startup(() => {
5
6    Meteor.methods({
7      'addAnswers':function(formData) {
8        try {
9          for(let i = 0, len = formData.length; i < len ; i++)
10           {
11             let answer = formData[i].value;
12             const qid = new Meteor.Collection.ObjectID(formData[i].name);
13             Questions.update({_id: qid}, {$push: { result: answer }});
14           }
15           return true;
16         } catch (e) {
17           return new Meteor.Error(e);
18         }
19       },
20     });
21
22
23     Meteor.publish('getQuestions', function() {
24       return Questions.find({});
25     });
26
27
28   });
```

Frontend - client/javascript/feedback.js

Add the following import to get access to the Questions model

Add the following function in `Template.feedback.helpers()`

```
import { Meteor } from 'meteor/meteor';
import { Questions } from '/models/questions.js';

Template.feedback.helpers({
  myTitle:function() {
    return "Feedback App";
  },
  checkType:function(type)
  {
    return type == "mcq";
  },
  questions:function() {
    return Questions.find({});
  },
});

Template.feedback.onCreated(function() {
  ...
});

Template.feedback.events({
  ...
});
```

`questions:function(){ ... }`

Access the Questions data collections obtained from the “getQuestions” publication.

Note: data coming from the publisher are controlled.
For this example, entire data is given accessed.

Frontend - client/javascript/feedback.js

Add the following function call in `Template.feedback.onCreated()`

```
import { Meteor } from 'meteor/meteor';
import { Questions } from '/models/questions.js';

Template.feedback.helpers({
  ...
});

Template.feedback.onCreated(function() {
  Meteor.subscribe('getQuestions');
});

Template.feedback.events({
  ...
});
```

`Meteor.subscribe("getQuestions");`

Subscribes to the "getQuestions" publication from server.

Frontend - client/javascript/feedback.js

Add the following function call in `Template.feedback.onCreated()`

```
import { Meteor } from 'meteor/meteor';
import { Questions } from '/models/questions.js';

Template.feedback.helpers({
  ...
});

Template.feedback.onCreated(function() {
  ...
});

Template.feedback.events({
  "submit #feedbackForm": function(event, instance) {
    event.preventDefault();
    const formData = $("#feedbackForm").serializeArray();
    Meteor.call("addAnswers", formData, function(error, result) {
      if(result) {
        alert("Thank you for your feedback!");
        $('#feedbackForm').trigger('reset');
      }
    });
  },
});
```

submit #feedbackForm: function(event, instance) { ... }

1. Find the ID feedbackForm in the feedback Template.
2. When you submit the form, it triggers and execute the code.
3. **Event and instance**, the event is the entire form that is selected.
4. The instance is the form within the template itself.
5. Use event to access its data and so forth.
6. **serializeArray()** is a method provided by jQuery to retrieve all form data from a form., which we pass to the server to process it.

Frontend - client/javascript/feedback.js

```
import {Meteor} from 'meteor/meteor';
import {Questions} from '/models/questions.js';
...

Template.feedback.onCreated(function() {
  Meteor.subscribe('getQuestions');
});

Template.feedback.events({
  ...
  Meteor.call("addAnswers",formData,function(error,result){
    if(result) {
      alert("Thank you for your feedback!");
      $('#feedbackForm').trigger('reset');
    }
  });
},
});
```

Meteor.subscribe()

This is used to subscribe to a publication (i.e. published function)

Meteor.call()

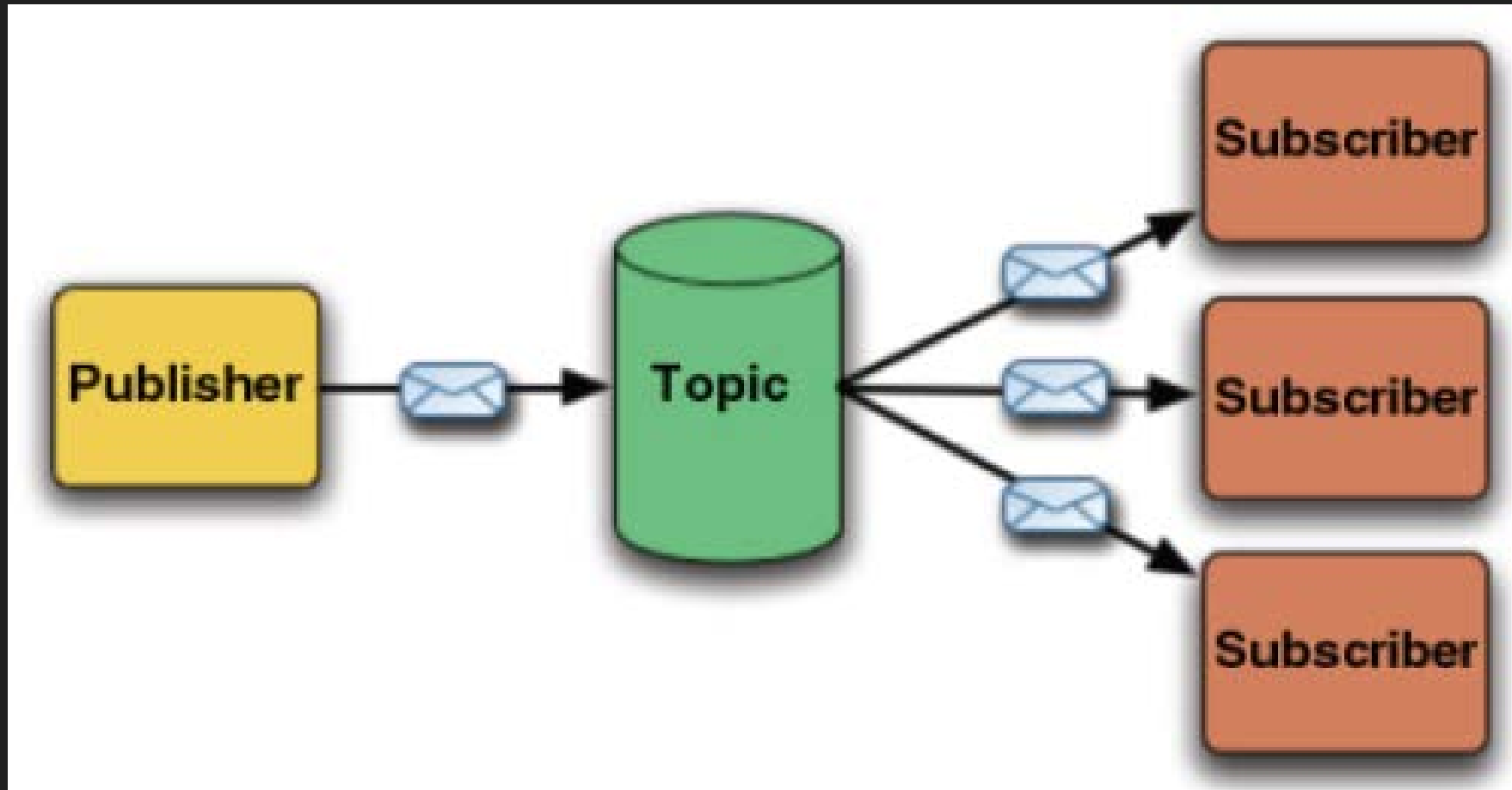
This is where you call a function defined in Meteor.methods.

Frontend - client/javascript/feedback.js

Publish/Subscribe VS function calls? What is the difference?

1. Pub/Sub are usually for **retrieval of data**. It provides reactivity effect, which when you change data in a collection, the server will push data to all subscribed clients.
1. Call/methods is a **traditional request and response**. You call a method, it respond, there is no reactivity or sort of socket connection between it. Usually **used for operations such as insert, update, and delete data**.

Publish/Subscribe model (PubSub)



Credits - <http://randomdotnext.com/content/images/2015/07/pubsub.png>

Frontend - client/javascript/feedback.js

Meteor.call(methodName, data, callback)

it has three parameters

1st – the name of the method in Meteor.methods

2nd – any data you want to pass over

3rd – the callback function, when Meteor.methods respond.

It will respond two variables – error and result.

Error is undefined if no error is returned.

Result is undefined if no data is returned.

```
import { Meteor } from 'meteor/meteor';
import { Questions } from '/models/questions.js';

Template.feedback.helpers({
  myTitle: function() {
    return "Feedback App";
  },
  questions: function() {
    return Questions.find({});
  },
  checkType: function(type) {
    {
      return type == "mcq";
    }
  }
});

Template.feedback.onCreated(function() {
  Meteor.subscribe('getQuestions');
});

Template.feedback.events({
  "submit #feedbackForm": function(event, instance) {
    event.preventDefault();
    const formData = $("#feedbackForm").serializeArray();
    Meteor.call("addAnswers", formData, function(error, result) {
      if(result)
      {
        alert("Thank you for your feedback!");
        $('#feedbackForm').trigger('reset');
      }
    });
  },
});
```


Backend - MongoDB

When you run the application.
It does not show you any data
so we will need to add data in the mongoDB.

Step 1

Open terminal, cd to where your meteor project is.

Type: **meteor mongo**

```
2017-04-11T18:21:24.512+0800 I CONTROL [initandlisten] ** NOTE: This is a 32-bit MongoDB binary running on a 64-bit operating
2017-04-11T18:21:24.512+0800 I CONTROL [initandlisten] **      system. Switch to a 64-bit build of MongoDB to
2017-04-11T18:21:24.512+0800 I CONTROL [initandlisten] **      support larger databases.
2017-04-11T18:21:24.512+0800 I CONTROL [initandlisten]
2017-04-11T18:21:24.512+0800 I CONTROL [initandlisten]
2017-04-11T18:21:24.512+0800 I CONTROL [initandlisten] ** NOTE: This is a 32 bit MongoDB binary.
2017-04-11T18:21:24.512+0800 I CONTROL [initandlisten] **      32 bit builds are limited to less than 2GB of data (or less with --journal).
2017-04-11T18:21:24.512+0800 I CONTROL [initandlisten] **      Note that journaling defaults to off for 32 bit and is currently off.
2017-04-11T18:21:24.512+0800 I CONTROL [initandlisten] **      See http://dochub.mongodb.org/core/32bit
2017-04-11T18:21:24.512+0800 I CONTROL [initandlisten]
meteor:PRIMARY> 
```

Backend - MongoDB

Step 2

Type in the following command string in mongo console

Basically, a `db.questions.insert()` allow us to manually insert data into the database with any string written in JSON format.

```
db.questions.insert({
  "text": "What is your gender?",
  "type": "mcq",
  "choices": [
    {
      "text": "male"
    }, {
      "text": "female"
    }
  ],
  "result": []
})
```

Backend - MongoDB

Step 3 - Result

```
...      "choices": [{
...          "text": "male"
...      },
...      {
...          "text": "female"
...      }
...  ],
...  "result": []
...  })
WriteResult({ "nInserted" : 1 })
meteor:PRIMARY> |
```

Backend - MongoDB

Extras

`db.questions.remove({})`

This removes all data in the questions collection.

```
[meteor:PRIMARY> db.questions.insert({          "text": "What is your gender?",          "type": "mcq",
,          "choices": [{          "text": "male"          }, {
    "text": "female"          },          "result": []          })
WriteResult({ "nInserted" : 1 })
[meteor:PRIMARY> db.questions.remove({})
WriteResult({ "nRemoved" : 1 })
meteor:PRIMARY> █
```

Deploy

Deploy meteor and test result.

1. Open terminal
2. Chang Directory or CD to your meteor project folder

3. Type meteor

4. Launch your browser type <http://localhost:3000>
5. Meteor uses port 3000 by default, please unblock it or keep it available.
6. You can also deploy it on iOS and Android.

<https://www.meteor.com/tutorials/blaze/running-on-mobile>

```
|||||  
  
=> Started proxy.  
=> Started MongoDB.  
=> Started your app.  
  
=> App running at: http://localhost:3000/  
    Type Control-C twice to stop.
```

Extra

To import a JSON file into mongoDB

Install mongoDB first. Meteor mongoDB is just an instance within its framework, it does not have the functionality - **mongoimport**
For windows, set ENV variable path of installed of mongoDB for mongoimport to be called in terminal.

Create a **questions.json** file

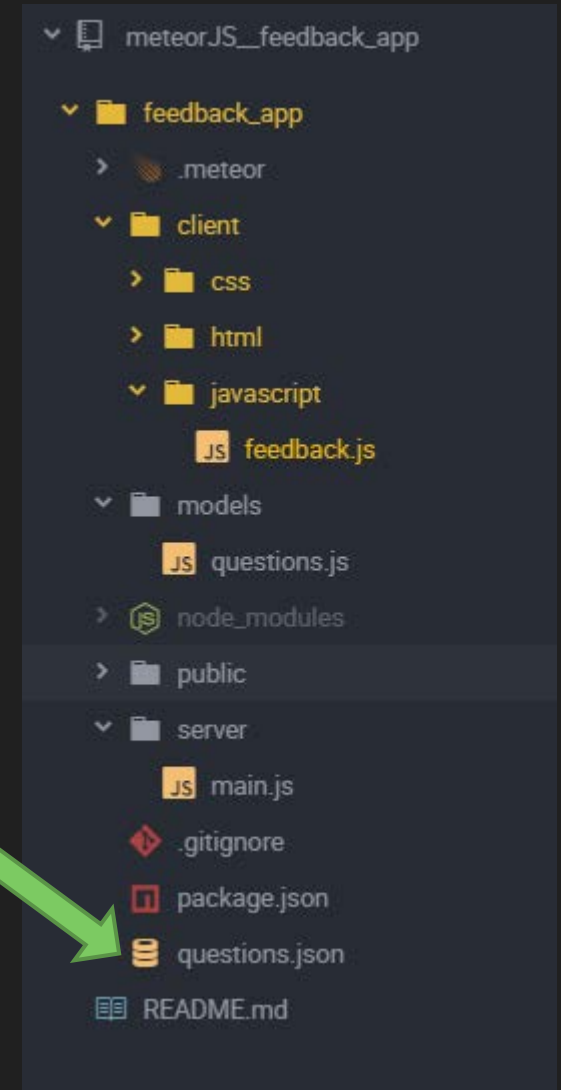
Inside questions.json create an array of json.

or copy from https://github.com/nicholas-ooi/meteorJS_feedback_app/blob/master/feedback_app/questions.json

In terminal

```
mongoimport -h localhost:3001 --db meteor --collection=questions  
questions.json --jsonArray
```

if you encounter mongoimport not found in command line.
go next slide



Extra

Windows:

if you encounter mongoimport not found in command line.

Install mongoDB <https://www.mongodb.com/download-center?jmp=nav>
for windows, add env variable to where mongoddb is installed.

Click environment variables.

double click PATH

click NEW

type C:/ where you installed mongoDB \mongodb\bin

restart command line

try again.

On Mac:

Go to terminal and type cd~/

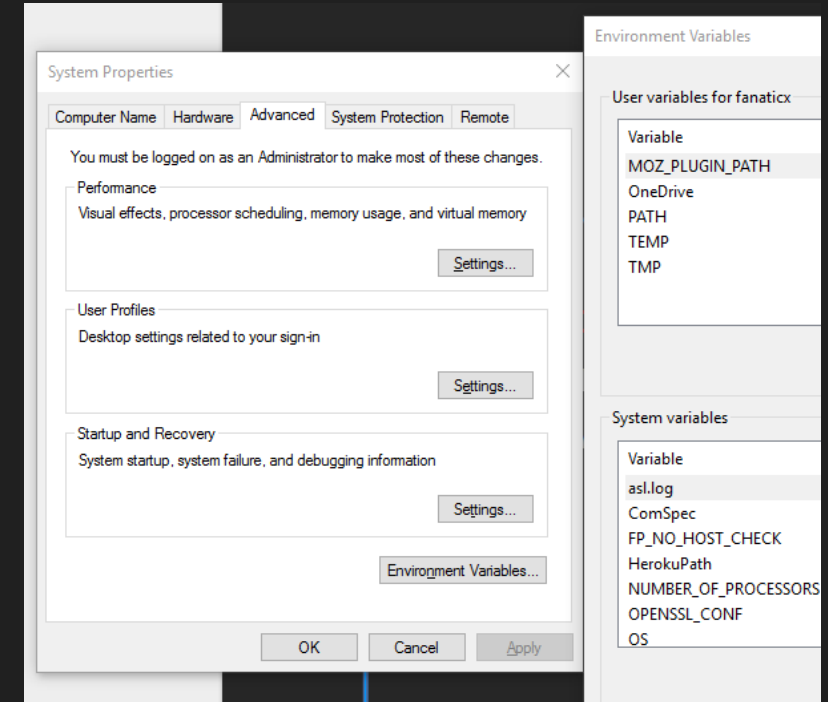
Then type "open -e .bash_profile"

A file should open on the terminal automatically. Paste in the following:

#Setting PATH for MongoDB

export PATH=\${PATH}:/<path where you installed

MongoDB>/mongodb-osx-x86_64-3.6.4/bin



```
C:\server\bin\mongodb\bin
```

Try out

Try it out

You will receive an alert if Meteor.call is successfully
Otherwise an error is thrown.

Go meteor mongo console

Type `db.questions.find({})`

You will see the list of json data retrieved.

Go MeteorJS With Nick!

Feedback App

What is your gender?

☒ male
☐ female

Is there any comments that you would like to give?

Very good!

How would you rate Nicholas Ooi as a ...?

☐ 1
☐ 2
☐ 3
☐ 4
☒ 5

How do you rate MeteorJS usefulness?

☐ 1
☐ 2
☐ 3
☐ 4
☒ 5

Submit Feedback

Thank you for your feedback!

OK

More Materials

- **Build a Whatsapp clone**

<https://blog.meteor.com/build-a-whatsapp-clone-with-meteor-and-ionic-meteor-platform-version-f2f4a45af1a8>

- **MeteorJS Angular**

<https://angular-meteor.com/>

- **MeteorJS Cookbook / Cheatsheet**

<https://github.com/clinical-meteor/software-development-kit/blob/master/table-of-contents.md>

Conclusion & Summary

1. You have learnt:

2. Create a meteorJS project.
3. How meteorJS structure works.
4. How meteorJS frontend templates works.
5. meteorJS backend for mongoDB works.
6. meteorJS is deployed.
7. build a secure, real-time feedback web app.

Submit a feedback about us at- <https://bit.ly/2KXuTlk>

You can download the entire source code at
https://github.com/nicholas-ooi/meteorJS_feedback_app

