* **[Template]**

Pre-Conditions:

1. xxx

Main Success Scenario:

1. Xxx

Alternatives:

1. Xxx

* **Add Document to Collection**

Main Success Scenario:

1. User insert method on a collection.

* **Add Package from Atmosphere**

Main Success Scenario:

1. Navigate to <https://atmospherejs.com/>
2. Choose package to add to project.
3. Click on package in the browser
4. Copy generated command from the browser

Ex. Meteor add twbs:boostrap

1. Paste the command into a command window and run.

* **Allow User to write to a collection via client code**

Main Success Scenario:

1. Use allow method.

* **Call function when template render event is triggered**

Note:

1. Template rendered means template is inserted into the DOM.

Main Success Scenario:

1. Highlight all “pre” elements in “codeSample” template.

Template.codeSample.onRendered(function () {

hljs.highlightBlock(this.findAll('pre'));

});

* **Control Collection Publish**

Main Success Scenario:

1. Use “meteor remove autopublish” command and/or
2. Meteor.publish and Meteor.subscribe.

* **Control if Code run on Server or Client**

Main Success Scenario:

1. Meteor.isClient is a boolean to check if running on client.
2. Meteor.isServer is a boolean to check if running on server.

* **Create Template Event Handler**

Note:

1. More info [here](http://docs.meteor.com/#/basic/Template-onRendered).

Main Success Scenario:

1. Handle click event for elements in the “my-button” class.

Template.example.events({

"click .my-button": function (event, template) {

alert("My button was clicked!");

},

});

1. Create template

<template name="example">

{{#with myHelper}}

<button class="my-button">My button</button>

<form>

<input type="text" name="myInput" />

<input type="submit" value="Submit Form" />

</form>

{{/with}}

</template>

* **Create Template**

Note:

1. A view is defined in a template.
2. Template is a snippet of HTML that can include dynamic data.
3. Can interact with templates from JS code to insert data or listen for events.
4. Templates are defined in .html files.
5. Templates cannot be defined in server, public, or private directories.
6. Syntax is from the Spacebars language.

Main Success Scenario:

1. Welcome someone to your page.

<!-- add code to the <body> of the page -->

<body>

<h1>Hello!</h1>

{{> welcomePage}}

</body>

<!-- define a template called welcomePage -->

<template name="welcomePage">

<p>Welcome to my website!</p>

</template>

* **Create Template Helper**

Note:

1. More info [here](http://docs.meteor.com/#/basic/Template-helpers).

Pre-Conditions:

1. Template helper goes in .JS file.
2. Have template

<template name="nametag">

<p>My name is {{name}}.</p>

</template>

Main Success Scenario:

1. Create template helper that provides a name to the nametag template.

Template.nametag.helpers({

name: "Ben Bitdiddle"

});

* **Define Documents to retrieve from a collection**

Main Success Scenario:

1. Use find or findOne methods on a collection.

* **Find Official Meteor JS site**

Main Success Scenario:

1. Go to <http://www.meteor.com>

* **Get Help for Meteor JS**

Main Success Scenario:

1. [Look for Meteor on Stack Overflow.](http://stackoverflow.com/questions/tagged/meteor)
2. [Look at Meteor Forums.](https://forums.meteor.com/)

* **Install Meteor JS on Windows OS**

Main Success Scenario:

1. Download official installer [here](https://install.meteor.com/windows).

* **Limit File Structure**

Main Success Scenario:

1. Files in “client” folder only served to the client.
2. Files in the “server” folder only served to the server.
3. “public” folder is good image storage.
4. Files in the “private” folder only accessible by server code.
5. HTML templates not in the folders above are compiled and sent to the client.
6. CSS files not in the folders above are sent to the client.
7. JavaScript files not in the folders above are loaded on the client and server.

* **Remove Document from a Collection**

Main Success Scenario:

1. Call remove method on collection.

* **Run Code at startup**

Main Success Scenario:

1. Run code when client or server starts up.

Meteor.startup()

* **Store data in Meteor**

Note:

1. Data is stored in collections.
2. JavaScript objects stored in collections are called documents.
3. Collection with a name is a persistent collection.
4. Persistent collection is stored on the server, but can be published to client.
5. Save collection in global JavaScript variable.
6. Client and Server code can access same collection in a global JavaScript variable.
7. Nameless collections is only local and not synchronized between client and server.
8. Meteor automatically publishes every doc in collection to each connected client.

Main Success Scenario:

1. Create a collection and store in a global variable.

Comments = new Mongo.Collection("comments");

* **Update Document in a Collection**

Main Success Scenario:

1. Use Update method on a collection.

* **Use Command Line Tool**

Main Success Scenario:

1. Get help.

meteor help

meteor help <command>

1. Make a new subdirectory

Meteor create <name>

1. Serve the current app at <http://localhost:3000>

Meteor run

1. Run project with Node inspector.

Meteor debug

1. Bundle your app and deploy it to a site.

Meteor deploy <site>

1. Update Meteor platform version that is installed.

Meteor update

1. Add a meteor package.

Meteor add <package name>

1. Remove a meteor package.

Meteor remove <package name>

1. Open a MongoDb shell for collections stored in the DB.

Meteor mongo

1. Remove all local data to reset current project to a refresh state

Meteor reset

Alternatives:

1. Xxx

* **Use Free Meteor Host**

Pre-Conditions:

1. App name is specified in <your app>.
2. The app name cannot be used by someone else.

Main Success Scenario:

1. Deploy app to <your app>.meteor.com.

Meteor deploy <your app>.meteor.com

* **Where Meteor Runs**

Main Success Scenario:

1. On Client or
2. On the server via Node.js container or
3. both