Meteor.js Cheat Sheet

A comprehensive cheat sheet for Meteor.js, covering core concepts, commands, and best practices for efficient development.

Core Concepts

Key Principles

Data on the Wire: Only data is sent, not HTML. This minimizes bandwidth and maximizes responsiveness.

Latency Compensation: Simulate server results on the client for a smoother user experience.

Full Stack Reactivity: Changes in the database are automatically reflected in the UI.

One Language: Use JavaScript (or TypeScript) for both the client and the server.

Directory Structure

`client/` : Client-side JavaScript, HTML, and CSS files.

`server/` : Server-side JavaScript files.

`imports/` : Reusable modules shared between client and server.

`public/` : Static assets like images and fonts.

Data Flow

Client interacts with UI -> Client-side methods -> Server-side methods (if needed) -> MongoDB -> Server publishes data -> Client subscribes to data -> UI updates reactively.

Commands & Syntax

Project Setup

`meteor create <app-name>`	Create a new Meteor project.
`meteor run`	Run the Meteor application.
`meteor add <package-name>`	Add a Meteor package to the project.
`meteor remove <package-name>`	Remove a Meteor package from the project.

Collections

`new Mongo.Collection('collectionName');`	Create a new MongoDB collection.
`Collection.insert({ ... });`	Insert a document into the collection.
`Collection.update({ _id: '...' }, { $set: { ... } });`	Update a document in the collection.
`Collection.remove({ _id: '...' });`	Remove a document from the collection.
`Collection.find({ ... }).fetch();`	Find documents and return them as an array.

Methods

`Meteor.methods({ 'methodName': function(arg1, arg2) { ... } });`	Define a Meteor method.
`Meteor.call('methodName', arg1, arg2, (error, result) => { ... });`	Call a Meteor method from the client.

Publish and Subscribe

Publishing Data

`Meteor.publish('publicationName', function(arg1, arg2) { return Collection.find({ ... }, { fields: { ... } }); });`

Publishes a set of documents from a collection.
Use `this.userId` to only publish data relevant to the current user.
Return `Collection.find()` or an array of cursors to publish.

Example: Publish all todos for the current user.

```
Meteor.publish('todos', function () {
  return Todos.find({ userId: this.userId });
});
```

Subscribing to Data

`Meteor.subscribe('publicationName', arg1, arg2, { onReady: function() { ... }, onError: function(error) { ... } });`

Subscribes to a publication from the client.
`onReady` callback is called when the subscription is ready.
`onError` callback is called if an error occurs during the subscription.

Example: Subscribe to the 'todos' publication.

```
Meteor.subscribe('todos', { onReady: () =>
console.log('Todos ready!'), onError: (error)
=> console.error(error) });
```

Controlling Data Access

Use `allow` and `deny` rules on collections to control data access.

```
Collection.allow({
  insert: function (userId, doc) { return
true; },
  update: function (userId, doc, fields,
modifier) { return true; },
  remove: function (userId, doc) { return
true; }
});
```

Note: In production, use more restrictive rules to prevent unauthorized access.

Templates and UI

Blaze Templates

Define templates using HTML with Handlebars-like syntax.

```
<template name="myTemplate">
  <h1>Hello, {{name}}!</h1>
  {{#each items}}
    <p>{{value}}</p>
  {{/each}}
</template>
```

Use template helpers to provide data to templates.

```
Template.myTemplate.helpers({
  name: function() {
    return 'World';
  },
  items: function() {
    return [{ value: 'Item 1' }, { value:
'Item 2' }];
  }
});
```

Handle events in templates.

```
<template name="myTemplate">
  <button>Click Me</button>
</template>


Template.myTemplate.events({
  'click button': function(event, template) {
    console.log('Button clicked!');
  }
});
```

Using React or Vue

Meteor can be integrated with React or Vue for more complex UIs.
Use packages like `meteorhacks:npm` to manage npm dependencies.
Follow the official documentation for integrating React or Vue with Meteor.

Example: Using React with Meteor:

```
meteor add react-meteor-data
```

Then, create React components and use `useTracker` to reactively fetch data from Meteor collections.

Session Variables

Note: Session variables are deprecated. Use ReactiveVar or ReactiveDict instead.

`Session.set('key', 'value');` - Set a session variable.
`Session.get('key');` - Get a session variable.