



Proposal Tugas
Pembahasan Framework Meteor JS dan
Implementasinya untuk Pembuatan Aplikasi Chat Realtime

Oleh

Albert Prima Laia	151112891
Heru Kurniawan	151240156
Sio Jurnal Pipin	151111241

Sekolah Tinggi Manajemen Informatika dan Komputer Mikroskil
Medan
2018

Bab I

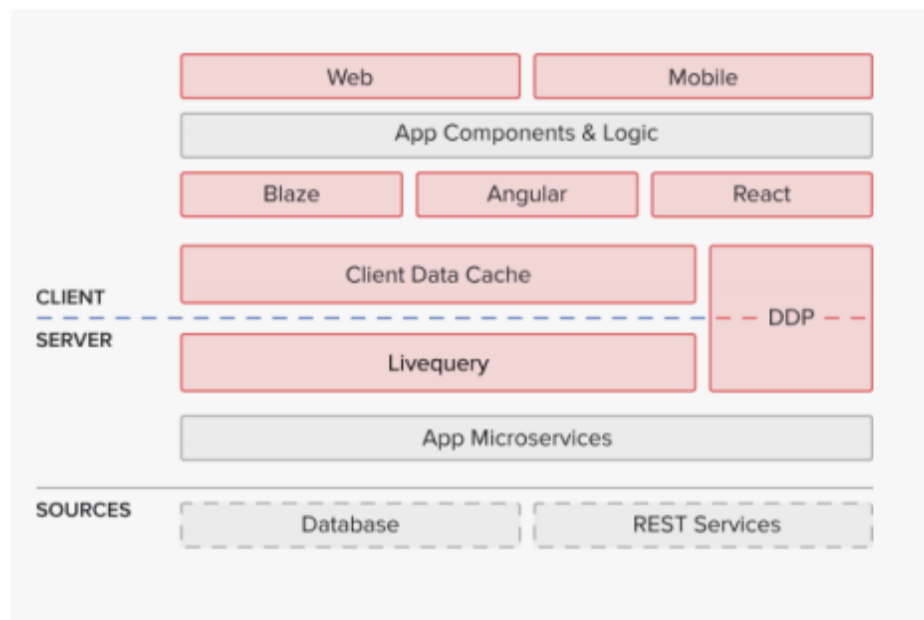
Pengantar

1. Framework Meteor JS

Framework Meteor Js menggunakan Javascript untuk membangun laman web yang modern dan real-time untuk penggunaan platform desktop maupun seluler. Meteor Js merupakan salah satu framework JS open-source yang paling terkenal karena framework ini berada pada ruang lingkup pengembangan aplikasi web modern yang sederhana dengan menggunakan 1 bahasa yaitu Javascript. Framework ini menawarkan cara yang inovatif untuk pengembangan aplikasi web yang luas, kaya dan interaktif.

Framework Meteor JS dapat menjadi pilihan framework terbaik karena berbagai alasan antara lain : Aplikasi meteor ditulis dalam bahasa JS, jadi tidak perlu belajar bahasa pemrograman khusus server lain. Hal Ini menyediakan cara mudah untuk menginstal pengembangan aplikasi yang meliputi web server dan server database. Ini termasuk cara cerdas untuk menggunakan sistem manajemen kemasan yang dapat dengan mudah dimasukkan ke aplikasi. Meteor menyediakan sumber data yang sangat reaktif, dengan model data terdistribusi rapi, yang berarti aplikasi akan merasa reaktif terhadap banyak pengguna dan Meteor dapat menghasilkan aplikasi iOS dan Android asli dari basis kode JS yang sama. Meteor meruntuhkan penghalang antara klien dan server dengan menerapkan model isomorfik yang berarti basis kode yang sama di sisi server dan sisi klien.

Arsitektur Meteor yang lengkap dapat dilihat pada gambar di bawah ini. Gambar tersebut menunjukkan bagaimana sisi-klien dan sisi-server berkomunikasi. Meteor awalnya digunakan untuk menggunakan **Blaze** hanya sebagai teknologi yang menghandel tampilan. Namun, pada 2016, React dan AngularJS masuk sebagai teknologi untuk menghandel bagian tampilan dari aplikasi web yang kita bangun. Saat ini, Meteor Js menggunakan **Blaze**, **React** dan **AngularJS** sebagai kerangka tampilan dan **Node.js** dan **MongoDB** di berada pada sisi back-end. Lapisan atas gambar 1 menunjukkan bahwa Meteor dapat digunakan untuk membangun aplikasi web atau mobile. Di bawah lapisan atas, ada tiga teknologi front-end Meteor seperti Blaze, AngularJS dan React. Di bawah teknologi tampilan / View, ada **Cache Data Klien**. Setiap pengguna yang terhubung ke aplikasi Meteor akan memiliki salinan data sisi server sebagai Cache Data Klien yang juga dikenal sebagai **Minimongo**. Kemudian, Di tengah gambar di bawah, ada **Distributed Data Protocol (DDP)**. Meteor menggunakan DDP untuk berkomunikasi dengan klien. Seperti yang digambarkan pada gambar di bawah, DDP adalah garis batas antara sisi klien dan sisi server.



Gambar 1

Meteor menyediakan tiga teknologi tampilan lapisan alternatif seperti yang digambarkan pada gambar di atas, seperti Blaze, React atau AngularJS dalam proyek Meteor. Di antara pilihan view / tampilan Meteor JS menyediakan alternatif berupa, Blaze dan React view.

2. Teknologi Front-end pada Framework Meteor JS

a. Blaze

Blaze adalah bagian bawaan dari Meteor Js dan "built-in reactive rendering library" yang menggunakan template HTML. Alih-alih menggunakan kombinasi template tradisional dan jQuery, Blaze menghilangkan kebutuhan untuk semua logika pembaruan dalam aplikasi yang mempehatikan perubahan data dan memanipulasi **DOM** dengan membiasakan arahan template dan mengintegrasikan dengan reaktivitas transparan Tracker dan kursor database Minimongo sehingga DOM memperbarui secara otomatis ketika ada perubahan. Berbeda dengan React, Blaze menekankan template HyperText Markup Language (HTML) daripada kelas komponen JS. Template lebih mudah didekati daripada kode JS dan lebih mudah dibaca, ditulis, dan bergaya dengan Cascading Style Sheet (CSS). Alih-alih menggunakan Tracker, React bergantung pada kombinasi panggilan setState eksplisit dan model data yang berbeda untuk mencapai rendering yang efisien.

Bahasa template Blaze terinspirasi oleh **Handlebars** dan ditulis dalam **Spacebars**, yang merupakan varian dari **Handlebars**. Template Blaze terdiri dari tag template dan atribut nama. Di Meteor JS, template digunakan untuk membuat koneksi antara antarmuka HTML dan kode JS.

Untuk membuat template, diperlukan tag template, atribut nama template, dan elemen HTML di dalam tag template.

```
1 <template name="leaderboard">
2   <h4>player #1 - point 5</h4>
3   <h4>player #2 - point 4</h4>
4 </template>
```

Contoh Blaze Template

Blaze menggunakan Spacebars untuk menangani sebuah event dan inject data ke dalam antarmuka HTML dari template. **Spacebars** terdiri dari HTML diselingi dengan tag template, yang dibatasi oleh dua kurung kurawal. **Spacebars** menggunakan helper template **Spacebars** dan event untuk memanipulasi data dan untuk menangani event. Seperti yang diilustrasikan pada di bawah, pembantu Spacebars dapat berisi variabel, data, dan fungsi. Spacebars ini memiliki variabel array bernama pemain.

```
1 Template.leaderboard.helpers({
2   players: [
3     { name:"player #1", score: 5 },
4     { name:"player #2", score: 4 }
5   ]
6 });
```

Spacebar Helper Function

Data array dalam daftar 2 memiliki dua data, nama pemain dan skor pemain masing-masing. Untuk memasukkan data pemain ke dalam template leaderboard, Spacebar menggunakan metode iterasi `{{#each players}} {{name}} - {{score}} {{/each}}`. Mekanisme ini mirip dengan `forEach` in JS atau `map` di ECMA Script 6 (ES). Kalau React Js menggunakan `map` untuk looping data dan masukkan ke dalam elemen JSX atau komponen.

```
1 <template name="leaderboard">
2   {{#each players}}
3     <h4>{{name}} - {{score}}</h4>
4   {{/each}}
5 </template>
```

Contoh iterasi data secara **Spacebar** ke dalam template Blaze

b. React Js

React adalah library front-end dari javascript. React JS adalah salah satu teknologi front-end paling populer untuk mendesain aplikasi web. React JS menggunakan JSX, yang merupakan preprocessor yang menambahkan sintaks XML ke JS. Kita dapat menggunakan React tanpa JSX tetapi dengan menggunakan JSX dapat membuat React lebih elegan. React JS membuat

membangun Antarmuka Pengguna interaktif (UI) sederhana. Merancang sebuah tampilan sederhana untuk setiap kondisi dalam aplikasi, React JS akan secara efisien memperbarui dan membuat komponen yang tepat ketika data berubah. React JS membangun komponen yang dienkapsulasi yang mengelola statusnya sendiri, lalu menyusunnya untuk membuat UI yang rumit. Penulisan komponen logika pada React JS ditulis dalam JS tidak dalam sebuah template. Oleh karena itu, mudah untuk mengirim data melalui aplikasi dan menjaga kondisi dari DOM. Sintaks berikut: elemen **const** = `<h1> Hello World </ h1>` adalah contoh JSX yang bukan merupakan deklarasi string HTML atau JS. Setiap elemen dan komponen JSX akan ditampilkan di elemen div root. Root div biasanya ditulis seperti ini: `<div id = "root"> </ div>`. Sehingga memungkinkan untuk menanamkan ekspresi JS di JSX dengan membungkusnya dengan kurung kurawal. Fungsi JS `fullName ()` dalam daftar 5 disematkan ke elemen JavaScript XML (JSX).

```
1  function fullName(player){ // function to format fullname
2      return player.firstName + ' ' + player.lastName;
3  }
4  const player = { // Simple JavaScript Object
5      firstName:"Firstname",
6      lastName : "Lastname"
7  };
8
9  //JSX element
10 const element = (<h1> Hello, {fullName(player)} </h1>);
11
12 //Rendering JSX element to the root
13 ReactDOM.render(element, document.getElementById('root'));
14
15 const player = {
16     name:"Player #1",
17     score:5
18 };
```

Contoh rendering JSX dalam sebuah div root

React JS menggunakan komponen JSX alih-alih elemen JSX menjadi produktif. Komponen membuat Source Code terpecah menjadi potongan independen yang dapat digunakan kembali. Komponen seperti fungsi JS yang bertanggung jawab untuk tugas tertentu. Sebagaimana ditunjukkan dalam gambar di bawah, komponen kelas React dapat diekspor dan digunakan di mana saja dalam file proyek.

```

1  export default class Player extends React.Component{
2      render(){
3          return(
4              <div>
5                  <h1> Player name</h1>
6                  <h1> Player score</h1>
7              </div>
8          );
9      }
10 };

```

Contoh komponen React JS

Komponen React dalam gambar di atas adalah komponen yang dapat digunakan kembali yang dapat digunakan seperti ini, *elemen const = <Player name = "name" />* dalam sebuah proyek. Atribut name dalam *<Player name = "name" />* adalah properti yang dapat berupa data statis atau data yang dapat diubah secara dinamis. **Player** sekarang adalah komponen yang dapat digunakan kembali yang dapat digunakan di mana saja dalam proyek. Dalam contoh ini, komponen hanya memiliki data statis. Untuk menempatkan data dinamis ke UI, Bereaksi menggunakan **metode setState** memainkan peran besar. Inti utama dari setiap komponen React adalah statusnya, sebuah objek yang menentukan bagaimana komponen merender dan berperilaku. React JS memiliki *state objects* yang memungkinkan membuat komponen yang dinamis dan interaktif. Jika komponen menyatakan mereka mengubah UI sesuai dengan negara. Kode yang diilustrasikan dalam daftar 7 adalah contoh komponen dengan komponen negara. State dapat berupa nilai tunggal atau sekumpulan nilai dalam suatu objek atau array.

```

1  export default class Player extends React.Component{
2      constructor(props){
3          super(props);
4          this.state = {score:0};
5      }
6      render(){
7          return(
8              <div>
9                  <h1> Player name</h1>
10                 <h1> Player score</h1>
11             </div>
12         );
13     }
14 };

```

State dalam komponen React JS

States dan **Props** memainkan peran besar dalam mengubah data secara dinamik. Komponen JSX biasanya dirender menjadi UI. React Js menggunakan fungsi built-in untuk memprioritaskan rendering dan untuk menghancurkan komponen yang diberikan. Sangat penting untuk membebaskan sumber daya yang digunakan oleh komponen dengan menghancurkan komponen yang diberikan. React Js menggunakan fungsi bawaan untuk memantau siklus hidup komponen. *Dua fungsi built-in* yang umum digunakan adalah *componentDidMount()* dan *componentWillUnmount()*. *componentDidMount()* hook berjalan setelah output komponen telah diberikan ke Document Object Model (DOM). Hook life-cycle *componentWillUnmount()* akan menghancurkan komponen yang diberikan. Bagian selanjutnya akan membahas sisi server Meteor.

3. Teknologi Front-end pada Framework Meteor JS

a. Node.js

Node.js adalah ruang lingkup **run-time** JS lintas-platform yang open-source untuk mengembangkan aplikasi yang berbeda platform. Sudah ada sejak 2009 dan membawa perubahan besar di dunia pemrograman web. Node.js memperkenalkan manajer paket yang disebut NPM pada Januari 2010. Node Package Management (NPM) membuat penerbitan dan pembagian kode sumber JS lebih mudah dan dirancang untuk menyederhanakan instalasi, memperbarui dan uninstallasi pustaka. Data tidak boleh hanya diminta dari klien, untuk menjadikan web reaktif, data juga harus didorong dari server ke klien dan Node.js bagus dalam aplikasi web real-time dengan menggunakan teknologi push melalui **web-socket**. Lingkungan run-time Node.js mengartikan JS dengan memanfaatkan sepenuhnya *mesin Google V8 JS*. **Mesin V8 JS** memungkinkan Node.js untuk menciptakan lingkungan run-time yang mengirim JS dari server ke klien dengan sangat cepat. V8 menerjemahkan JS ke dalam kode mesin asli, alih-alih bekerja untuk menafsirkannya sebagai kode byte, hal ini memberikan Node.js kecepatan. Kecepatan responsif, dikombinasikan dengan pemrograman asynchronous membuat Node.js sangat responsif dan diinginkan dalam membangun aplikasi yang dinamis dan reaktif. Meteor memanfaatkan teknologi Node.js dengan menggunakannya di sisi back-end untuk menyampaikan konten kepada klien.

a. MongoDB

MongoDB adalah basis data open source yang menyediakan kinerja tinggi, ketersediaan tinggi, dan penskalaan otomatis. MongoDB adalah sistem basis data NoSQL yang digunakan dalam banyak aplikasi yang menawarkan struktur yang sangat fleksibel yang dapat berkisar dari basis data dasar hingga kompleks. MongoDB tidak memerlukan skema yang ditentukan sebelumnya untuk memulai dan dapat ditingkatkan setiap saat. MongoDB menggunakan sintaks jenis JSON disebut *Binary JSON (BSON)*. Berbeda dengan database relasional terkenal (RDB), SQL, Mo

ngoDB tidak menggunakan tabel, baris dan kolom. Sebaliknya menggunakan pengumpulan, dokumen dan lapangan. Seperti tabel di bawah menunjukkan beberapa istilah MongoDB dan SQL berbeda satu sama lain.

MongoDB	SQL
Database	Database
Collection	Table
Index	Index
Document	Row
Field	Column
Embed Documents and Linking	Join
Primary Key	Primary
Aggregation	Group by

Perbandingan MongoDB dan SQL

4. Kelebihan menggunakan Meteor JS

a. Blaze

BLAZE Blaze ini adalah salah satu JavaScript library yang membuat suatu template menjadi live-updating.

b. DPP

DPP atau singkatan dari Distributed Data Protocol merupakan protocol untuk mengambil data dari database dari server, dan menerima live-update ketika data diubah.

c. LIVEQUERY

Meteor Livequery masih merupakan keluarga dari livedatabase connectors. Konektor ini memungkinkan untuk melakukan “live query” pada databas.

Bab II

Implementasi

1. Mulai Project dengan Meteor JS

a. Instalasi Meteor Js Dan Mongodb

- Pertama install **Chocolatey** yang dapat dilihat melalui laman <https://chocolatey.org/install> , Kemudian install Meteor JS dengan menggunakan terminal / CMD pada windows dan mengetik “*choco install meteor*”.

Administrator: Command Prompt - choco install meteor

```
C:\Windows\system32>choco install meteor
Chocolatey v0.10.11
Installing the following packages:
meteor
By installing you accept licenses for the packages.
Progress: Downloading chocolatey-core.extension 1.3.3... 100%
Progress: Downloading meteor 0.0.2... 100%

chocolatey-core.extension v1.3.3 [Approved]
chocolatey-core.extension package files install completed. Performing other installation steps.
Installed/updated chocolatey-core extensions.
The install of chocolatey-core.extension was successful.
Software installed to 'C:\ProgramData\chocolatey\extensions\chocolatey-core'

meteor v0.0.2 [Approved]
meteor package files install completed. Performing other installation steps.
The package meteor wants to run 'chocolateyinstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable -n allowGlobalConfirmation
Do you want to run the script?([Y]es/[N]o/[P]rint): y

Downloading meteor 64 bit
from 'https://packages.meteor.com/bootstrap-link?arch=os.windows.x86_64'
Progress: 0% - Saving 1.23 MB of 177.2 MB
```

- Untuk membuat file project, pada terminal masukkan perintah “*meteor create nama_aplikasi*”.
- Untuk pindah ke folder project, ketikkan perintah “*cd nama_aplikasi*”
- Untuk memulai menggunakan project ketikkan perintah “*meteor*”
- Kemudian, buka browser web dan buka <http://localhost:3000> untuk melihat aplikasi yang berjalan. Berikut tampilan awal dari Meteor JS.

Welcome to Meteor!

[Click Me](#)

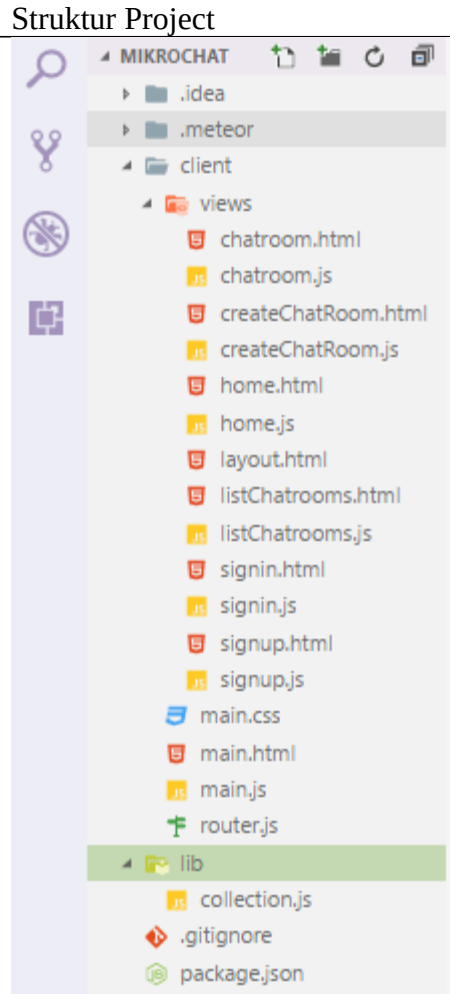
You've pressed the button 20 times.

Learn Meteor!

- [Do the Tutorial](#)
- [Follow the Guide](#)
- [Read the Docs](#)
- [Discussions](#)

b. Membuat aplikasi Chat secara Realtime dengan Meteor JS

Disini kami akan mencoba menggunakan Meteor JS untuk membuat aplikasi Chat yang memungkinkan pengguna untuk berkomunikasi secara langsung. Struktur file dari project ini adalah sebagai berikut :

Struktur Project	Penjelasan File Project
	<ul style="list-style-type: none">• chatroom.html : Tampilan dari halaman chatroom• Chatroom.js : Javascript handler untuk tampilan chatroom.html• createChatRoom.html : Tampilan dialog pop up untuk penambahan chatroom• createChatRoom.js : Javascript handler untuk tampilan CreateChatRoom.html• home.html : Tampilan dari halaman home• home.js : Javascript handler untuk tampilan home.html• layout.html : Verifikasi apakah user sudah login, jika sudah login diarahkan ke tampilan home jika belum ke tampilan sign in.• listChatrooms.html : Tampilan dialog pop up untuk daftar chatroom yang ada• listChatrooms.js : Javascript handler untuk tampilan listChatrooms.html• signin.html : Tampilan dari halaman sign in• signin.js : Javascript handler untuk tampilan signin.html• signup.html : Tampilan dari halaman sign up atau registrasi• signup.js : Javascript handler untuk tampilan signup.html• main.css : Css tambahan yang ingin kita gunakan• main.html : Tampilan atau template utama dari aplikasi chat yang akan kita buat.• main.js : Javascript handler utama untuk semua template dan tampilan yang ada.• router.js : Javascript handle routing• Collection.js : Definsi dokumen database yang ada di project kita selain dokumen user dan juga pendefinisian untuk skema dari dokumen tersebut.

1. Membuat tampilan Sign In

signin.html

```
<template name="signin">
  <div class="top-content">
    <div class="inner-bg">
      <div class="container">
        <div class="row">
          <div class="col-sm-8 col-sm-offset-2 text">
            <h1><strong>Chat</strong> App</h1>
            <div class="description">
              <p>
                Aplikasi Chatting dengan Meteor Js dan MongoDB
              </p>
            </div>
          </div>
        </div>
        <div class="row">
          <div class="col-sm-6 col-sm-offset-3 form-box">
            <div class="form-top">
              <div class="form-top-left">
                <h3>Masuk</h3>
                <p>Masukkan username dan password anda</p>
              </div>
              <div class="form-top-right">
                <i class="fa fa-key"></i>
              </div>
            </div>
            <div class="form-bottom">
              <form role="form" action="" method="post" class="login-form">
                <div class="form-group">
                  <label class="sr-only" for="form-username">Username</label>
                  <input type="text" name="form-username" placeholder="Username..." class="form-username form-control" id="form-username">
                </div>
                <div class="form-group">
                  <label class="sr-only" for="form-password">Password</label>
                  <input type="password" name="form-password" placeholder="Password..." class="form-password form-control" id="form-password">
                </div>
                <button type="submit" class="btn">Sign in!</button>
                <h6>Don't have account?</h6>
                <a href="{{pathFor 'signup'}}">Create account</a>
              </form>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</template>
```

signin.js

```
Template.signin.events({
  'submit form':function (e, tmpl) {
    e.preventDefault();
    var usernameVar = tmpl.find('#form-username').value;
    var passwordVar = tmpl.find('#form-password').value;
    Meteor.loginWithPassword(usernameVar, passwordVar, function (e) {
      if(e){
        alert("Login Error \n" + e);
      }else{
        Router.go('/');
      }
    });
  }
});
```

Yang perlu diperhatikan :

```
1 Template.signin.events({
2   'submit form': function
```

```
1 <template name="signin">
2   <div class="container">
```

File signin harus sesuai dengan nama template yang kita buat di html. Di file javascript di atas saya mendefinisikan event-event yang ada di file signin.html. Tetapi untuk event di atas cuma terdapat satu event. Yaitu event submit untuk tag form yang ada di html file signin.html.

Meter.loginWithPassword merupakan method atau fungsi dari *package accounts-password* yang berfungsi untuk mengecek data inputan user apakah sudah terdaftar sebagai user atau belum. Pendefinisian event seperti di atas akan juga terjadi di template-template lain yang kita buat.

2. Membuat tampilan Sign Up

signup.html

```
<template name="signup">
  <div class="top-content">
    <div class="inner-bg">
      <div class="container">
        <div class="row">
          <div class="col-sm-8 col-sm-offset-2 text">
            <h1><strong>Chat</strong> App</h1>
            <div class="description">
              <p>
                Aplikasi Chatting dengan Meteor Js dan MongoDB
              </p>
            </div>
          </div>
        </div>
        <div class="row">
          <div class="col-sm-6 col-sm-offset-3 form-box">
            <div class="form-top">
              <div class="form-top-left">
                <h3>Daftar</h3>
                <p>Daftar dan chat teman anda</p>
              </div>
              <div class="form-top-right">
                <i class="fa fa-key"></i>
              </div>
            </div>
            <div class="form-bottom">
              <form role="form" method="post" class="login-form">
                <div class="form-group">
                  <label class="sr-only" for="form-email">Email</label>
                  <input type="text" name="form-email" placeholder="Email..." class="form-username form-control" id="form-email">
                </div>
                <div class="form-group">
                  <label class="sr-only" for="form-username">Username</label>
                  <input type="text" name="form-username" placeholder="Username..." class="form-username form-control" id="form-username">
                </div>
                <div class="form-group">
                  <label class="sr-only" for="form-password">Password</label>
                  <input type="password" name="form-password" placeholder="Password..." class="form-password form-control" id="form-password">
                </div>
                <div class="form-group">
                  <label class="sr-only" for="form-conf-password">Confirm Password</label>
```

```

        <input type="password" name="form-conf-password" placeholder="Confirm
Password..." class="form-password form-control" id="form-conf-password">
    </div>
    <button type="submit" class="btn">Sign Up</button>
</form>
</div>
</div>
</div>
</div>
</div>
</template>

```

signup.js

```

Template.signup.events({
  'submit form': function(e, tmpl){
    e.preventDefault();
    var emailVar = tmpl.find('#form-email').value;
    var usernameVar = tmpl.find('#form-username').value;
    var passwordVar = tmpl.find('#form-password').value;
    var confPasswordVar = tmpl.find('#form-conf-password').value;
    if(passwordVar !== confPasswordVar || emailVar == 0 ||
      usernameVar == 0 || passwordVar == 0 || confPasswordVar == 0) {
      console.log("email : "+emailVar+"\nusername : "+usernameVar+"\npassword : 
"+passwordVar);
      window.alert("Isikan data dengan benar");
    }
    Accounts.createUser({
      email: emailVar,
      username: usernameVar,
      password: passwordVar
    }, function (err) {
      if(err){
        window.alert("Error \n"+err);
        console.log(err);
      }
      else{
        console.log("success");
        window.alert("Sukses");
        Router.go('/home');
      }
    });
  }
});

```

Yang perlu diperhatikan :

```
    window.alert("Isikan data dengan :");
  }
  Accounts.createUser({
    email: emailVar,
    username: usernameVar,
    password: passwordVar
  }, function (err) {
    if(err){
      window.alert("Error \n"+err);
      console.log(err);
    }
    else{
      console.log("success");
      window.alert("Sukses");
      Router.go('/home');
    }
  });
});
```

Accounts.createUser merupakan method untuk melakukan penambahan user ke database MongoDB. Method ini merupakan bawaan dari package accounts-password.

3. Membuat tampilan Home

```
<template name="home">
  {{#if currentUser}}
  <br>
  <div class="container bootstrap snippet">
    <div class="row">
      {{#if showCreateDialog}}
      {{> createChatRoom}}
      {{/if}}
      {{#if showListDialog}}
      {{> listChatrooms}}
      {{/if}}
    </div>
    <div class="portlet portlet-default">
      <div class="portlet-heading">
        <div class="portlet-title">
          <ul class="nav nav-pills" style="float: right">
            <li role="presentation"><a href="#" id="createChatRoom">Create Chatroom</a></li>
            <li role="presentation"><a href="#" id="listChatRoom">List Chatroom</a></li>
          </ul>
          <h2>Aplikasi Chatting</h2>
          <h3>{{ currentUser.username }}</h3>
          <a href="#" id="logout">Log Out</a>
        </div>
        <div class="clearfix"></div>
      </div>
      <div id="chat" class="panel-collapse collapse in">
        <div>
          <div id="always-scoll" class="portlet-body chat-widget" style="overflow-y: auto; overflow-x:
hidden; width: auto; height: 420px">
            <div class="row">
              <div class="col-lg-12">
                <p class="text-center text-muted small" style="float: right"></p>
              </div>
            </div>
            {{#each chat}}
            <div class="row">
              <div class="col-lg-12">
                <div class="media">
                  <div class="media-body">
                    <h4 class="media-heading">{{username}}
                    <span class="small pull-right">{{formatDate timestamp}}</span>
                  </h4>
                  <p>
                    {{text}}
                  </p>
                  {{#if canDelete username}}
                  <a href="#" class="delete"><h6>Delete</h6></a>
                </if>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</template>
```

```

        </div>
      </div>
    </div>
  </div>
  <hr>
  {{each}}
</div>
</div>
<div class="portlet-footer">
  <form role="form add-chat">
    <div class="form-group">
      <textarea class="form-control" placeholder="Enter message..." id="text-
msg"></textarea>
    </div>
    <div class="form-group">
      <button type="button" class="btn btn-default pull-right add-chat">Send</button>
    </div>
  </div>
</form>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
{{else}}
  {{ tolayout }}
{{/if}}
</template>

```

home.js

```

Template.home.events({
  'click #logout':function (e) {
    e.preventDefault();
    Meteor.logout();
    Router.go('/');
  },
  'click #createChatRoom':function (e) {
    e.preventDefault();
    Session.set('showCreateDialog', true);
  },
  'click #listChatRoom':function (e) {
    e.preventDefault();
    Session.set('showListDialog', true);
  },
  'click .add-chat':function (e, tmpl) {
    e.preventDefault();
    var text = tmpl.find('#text-msg').value;
    addChat(text);
    var objDiv = document.getElementById("always-scoll");
    objDiv.scrollTop = objDiv.scrollHeight;
    tmpl.find('#text-msg').value = '';
  },
  'click .delete':function (e, tmpl) {
    e.preventDefault();
    deleteChat(this._id);
  }
});

Template.home.helpers({
  tolayout:function () {
    Router.go('/');
  },
  showCreateDialog:function () {
    return Session.get('showCreateDialog');
  }
});

```

```

},
showListDialog:function () {
  return Session.get('showListDialog');
},
chat:function () {
  return Messages.find({
    chatRoomId: { $exists : false }
  }, { sort: { timestamp: 1 }});
},
formatDate: function(timestamp) {
  var date = new Date(timestamp);
  return date.toLocaleString();
},
canDelete:function (username) {
  if(username == Meteor.user().username)
    return true;
  else
    return false;
}
});

var addChat = function (text) {
  var id = Messages.insert({
    username: Meteor.user().username,
    text: text,
    timestamp: Date.now()
  });
  if(id==0){
    alert("insert error");
  }
};

var deleteChat = function (_id) {
  Messages.remove(_id);
};

Template.home.onRendered(function () {
  var objDiv = document.getElementById("always-scoll");
  objDiv.scrollTop = objDiv.scrollHeight;
});

```

Yang perlu diperhatikan :

```

<template name="home">
  {{#if currentUser}}
  <br>

```

Tulisan `{{#if currentUser}}` juga merupakan bawaan dari package accounts-password. Dimana `currentUser` merupakan pengecekan apakah halaman ini di akses oleh user yang terdaftar. Ketika ada if pembuka maka pasti ada if penutup bahkan juga ada else.

```

</div>

{{else}}
  {{ toLayout }}
{{/if}}

```

Else dan If penutup berada di bawah. Dimana jikalau user belum terdaftar maka akan di panggil `{{ toLayout }}`. Meteor Js ini menggunakan handlebar.js sebagai templatengnya.

Jadi penulisan dengan `{{ }}` merupakan penulisan handlebar.js. Dan pemrosesannya berada di sisi javascript. Dan kita bisa lihat di file `home.js` terdapat seperti ini :

```
Template.home.helpers({
  toLayout: function () {
    Router.go('/');
  },
  showCreateDialog: function () {
    return Session.get('showCreateDialog');
  },
  showListDialog: function () {
    return Session.get('showListDialog');
  }
});
```

`ToLayout` di atas merupakan fungsi atau method yang dipanggil ketika kita mendefinisikan `{{ toLayout }}` di file `home.html`. Dan bisa kita lihat di dalam method atau helper tersebut dia memanggil `Router.go('/')` ;

`Router.go('/')` merupakan fungsi bawaan dari `iron:router`. Dimana dia akan redirect ke url root dengan memanggil `(')`. Dan di bawahnya terdapat helper `showCreateDialog: function(){`
..... }

Yang mana ini mengembalikan value yang ada di session dengan key `showCreateDialog`.

```
{{/if}}
{{#if showListDialog}}
  {{> listChatrooms}}
{{/if}}
<div>
  <div class="portlet portlet-default">
    <div class="portlet-heading">
```

Dapat kita lihat potongan kodingan di atas terdapat handlebar dengan penulisan `{{> listChatRooms}}`.

Penulisan tersebut merupakan pemanggilan template dengan nama `listChatRooms`. Jadi kalo digantikan dengan template maka akan tertulis seperti di bawah ini.

```
<template name="listChatRooms">
..... </template>
```

Helper atau method dengan mengirimkan parameter

```
</p>
{{#if canDelete username}}
<a href="#"
```

```
right">{{formatDate timestamp}}</span>
```

Dapat di lihat gambar di atas merupakan pemanggilan helper dengan parameter yang satu `canDelete` dengan parameter `username` Dan `formatDate` dengan parameter `timestamp`

Untuk mendapatkan parameter tersebut pada file `home.js` terdapat koding seperti ini

```

},
formatDate: function(timestamp) {
    var date = new Date(timestamp);
    return date.toLocaleString();
},
canDelete: function (username) {
    if (username === Meteor.user().username)
        return true;
    else
        return false;
}

```

4. Interaksi dengan Database

```

var addChat = function (text) {
    var id = Messages.insert({
        username: Meteor.user().username,
        text: text,
        timestamp: Date.now()
    });
    if(id==0){
        alert("insert error");
    }
};

var deleteChat = function (_id) {
    Messages.remove(_id);
};

```

Pada file home.js terdapat kodingan seperti di atas. Koding di atas merupakan salah satu cara pendefinisian fungsi atau method di javascript. Dimana pada method addChat dia membutuhkan satu parameter yaitu parameter text yang diinput user. Kemudian dilakukan insert data ke dalam dokumen Messages yang ada di database.

Dan di method deleteChat juga membutuhkan satu parameter, yaitu id dari dokumen Messages yang ingin kita hapus. Deklarasi Messages akan kita definisikan file collection.js di bawah direktori lib. Retrieving data dari Database.

```

},
chat: function () {
    return Messages.find({
        chatRoomId: { $exists: false }
    }, { sort: { timestamp: 1 }});
}

```

Return data dari method chat merupakan data yang di dapat dari database. Bisa di lihat data tersebut di dapat dari dokumen Messages. `{{username}}` merupakan value field yang ada di dokumen Messages. Begitu juga dengan timestamp yang di tulis seperti ini `{{ formatDate timestamp }}`. Itu berarti timestamp di jadikan parameter untuk method formatDate. Dimana timestamp merupakan value yang ada di dokumen Messages.

5. Membuat Tampilan Dialog Pop Up Createchatroom

createChatRoom.html

```

<template name="createChatRoom">
  <div id="modal-id">
    <div class="modal-dialog">
      <div class="modal-content">
        <div class="modal-header">
          <button type="button" class="close exit" data-dismiss="modal" aria-
hidden="true">&times;</button>
          <h4 class="modal-title">Create Chatroom</h4>
        </div>
        <div class="modal-body">
          <input type="text" name="name" class="form-control" id="name" placeholder="Chatroom
name">
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-success add create">Create</button>
          <button type="button" class="btn cancel exit" data-dismiss="modal">Close</button>
        </div>
      </div><!-- /.modal-content -->
    </div><!-- /.modal-dialog -->
  </div><!-- /.modal -->
</template>

```

createChatRoom.js

```

Template.createChatRoom.events({
  'click .exit':function (e) {
    e.preventDefault();
    Session.set('showCreateDialog', false);
  },
  'click .create':function (e, tmpl) {
    e.preventDefault();
    var name = tmpl.find('#name').value;
    addChatroom(name);
    tmpl.find('#name').value = '';
    Session.set('showCreateDialog', false);
  }
});

var addChatroom = function (name) {
  var id = ChatRooms.insert({
    userId: Meteor.user().username,
    name: name,
    timestamp: Date.now()
  });
  if(id==0){
    alert("insert error");
  }else{
    alert("insert success");
  }
};

```

6. Membuat Tampilan Dialog Pop Up Listchatrooms

listChatrooms.html

```

<template name="listChatrooms">
  <div id="modal-id">
    <div class="modal-dialog">
      <div class="modal-content">
        <div class="modal-header">
          <button type="button" class="close exit" data-dismiss="modal" aria-
hidden="true">&times;</button>
          <h4 class="modal-title">List Available Chatrooms</h4>
        </div>
        <div class="modal-body">
          <div class="list-group">

```

```

        {{#each chatrooms}}
        <a href="{{pathFor route='chatroom' _id=_id chatroom_name=name}}" class="list-
group-item">{{name}}</a>
        {{/each}}
    </div>
</div>
<div class="modal-footer">
    <!--<button type="button" class="btn btn-success add">Join</button>-->
    <button type="button" class="btn cancel exit" data-dismiss="modal">Close</button>
</div>
</div><!-- /.modal-content -->
</div><!-- /.modal-dialog -->
</div><!-- /.modal -->
</template>

```

listChatrooms.js

```

Template.listChatrooms.events({
  'click .exit':function (e) {
    e.preventDefault();
    Session.set('showListDialog', false);
  }
});

Template.listChatrooms.helpers({
  chatrooms:function () {
    return ChatRooms.find({}, { sort: { timestamp: -1 }});
  }
});

```

7. Membuat Tampilan Chatroom

chatroom.html

```

<template name="chatroom">
  {{#if currentUser}}
  <div class="container bootstrap snippet">
    <div class="row">
      {{#if showCreateDialog}}
      {{> createChatRoom}}
      {{/if}}
      {{#if showListDialog}}
      {{> listChatrooms}}
      {{/if}}
    </div>
    <div class="portlet portlet-default">
      <div class="portlet-heading">
        <div class="portlet-title">
          <ul class="nav nav-pills" style="float: right">
            <li role="presentation"><a href="#" id="createChatRoom">Create Chatroom</a></li>
            <li role="presentation"><a href="#" id="listChatRoom">List Chatroom</a></li>
          </ul>
          <a href="{{pathFor route='home'}}" ><h4>Back to Home</h4></a>
          <h2>{{chatroomName}}</h2>
          <a href="#" id="logout">Log Out</a>
        </div>
        <div class="clearfix"></div>
      </div>
      <div id="chat" class="panel-collapse collapse in">
        <div>
          <div id="always-scoll" class="portlet-body chat-widget" style="overflow-y: auto; overflow-x:
hidden; width: auto; height: 420px">

```

```

<div class="row">
  <div class="col-lg-12">
    <p class="text-center text-muted small" style="float: right"></p>
  </div>
</div>

{{#each chat}}
<div class="row">
  <div class="col-lg-12">
    <div class="media">
      <div class="media-body">
        <h4 class="media-heading">{{username}}
        <span class="small pull-right">{{formatDate timestamp}}</span>
        </h4>
        <p>
          {{text}}
        </p>
        {{#if canDelete username}}
        <a href="#" class="delete"><h6>Delete</h6></a>
        {{/if}}
      </div>
    </div>
  </div>
</div>
<hr>
{{/each}}

</div>
</div>
<div class="portlet-footer">
  <form role="form add-chat">
    <div class="form-group">
      <textarea class="form-control" placeholder="Enter message..." id="text-
msg"></textarea>
    </div>
    <div class="form-group">
      <button type="button" class="btn btn-default pull-right add-chat ">Send</button>
      <div class="clearfix"></div>
    </div>
  </form>
</div>
</div>
</div>
</div>
</div>
{{else}}
{{ tolayout }}
{{/if}}
</template>

```

chatroom.js

```
Template.chatroom.helpers({
  toLayout:function () {
    Router.go('/');
  },
  showCreateDialog:function () {
    return Session.get('showCreateDialog');
  },
  showListDialog:function () {
    return Session.get('showListDialog');
  },
  formatDate: function(timestamp) {
    var date = new Date(timestamp);
    return date.toLocaleString();
  },
  canDelete:function (username) {
    if(username == Meteor.user().username)
      return true;
    else
      return false;
  }
});

Template.chatroom.events({
  'click #logout':function (e) {
    e.preventDefault();
    Meteor.logout();
    Router.go('/');
  },
  'click #createChatRoom':function (e) {
    e.preventDefault();
    Session.set('showCreateDialog', true);
  },
  'click #listChatRoom':function (e) {
    e.preventDefault();
    Session.set('showListDialog', true);
  },
  'click .add-chat':function (e, tmpl) {
    e.preventDefault();
    var text = tmpl.find('#text-msg').value;
    var _id = Router.current().params._id;
    addChatInChatroom(text, _id);
    var objDiv = document.getElementById("always-scoll");
    objDiv.scrollTop = objDiv.scrollHeight;
    tmpl.find('#text-msg').value = "";
  },
  'click .delete':function (e, tmpl) {
    e.preventDefault();
    deleteChat(this._id);
  }
});

var addChatInChatroom = function (text, chatroom_id) {
  var id = Messages.insert({
    username: Meteor.user().username,
    text: text,
    timestamp: Date.now(),
    chatRoomId: chatroom_id
  });
  if(id==0){
    alert("insert error");
  }
}
```

```

    }
  };

  var deleteChat = function (_id) {
    Messages.remove(_id);
  };

  Template.chatroom.onRendered(function () {
    var objDiv = document.getElementById("always-scoll");
    objDiv.scrollTop = objDiv.scrollHeight;
  });

```

8. Membuat Tampilan Layout

layout.html

```

<template name="layout">
  {{#if currentUser}}
    {{> home}}
  {{else}}
    {{> signin}}
  {{/if}}
</template>

```

main.css

```

#modal-id{
  position: fixed;
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
  z-index: 1050;
}

```

main.html

```

<head>
  <title>Chat App</title>
</head>
<body>
</body>
<template name="main">
  {{> yield}}
</template>

```

`{{ >yield }}` merupakan pendefinisian untuk tumpuan layout karena template main merupakan layout template dari aplikasi kita yang kita definisikan di dalam file router.js.

9. Inisialisai Dokumen database dan skemanya

collection.js

```

Messages = new Mongo.Collection('messages');
var Schemas = {};
Schemas.Messages = new SimpleSchema({
  text: {
    type: String
  },
  timestamp: {

```

```

    type: Number
  },
  username: {
    type: String
  },
  chatRoomId: {
    type: String,
    optional: true
  }
});
Messages.attachSchema(Schemas.Messages);

ChatRooms = new Mongo.Collection('chatRooms');
Schemas.ChatRooms = new SimpleSchema({
  name: {
    type: String
  },
  userId: {
    type: String
  },
  timestamp: {
    type: Number
  }
});
ChatRooms.attachSchema(Schemas.ChatRooms);

```

Code di atas merupakan inialisasi dokumen database kita. Terdapat 2 dokumen yaitu Messages dan ChatRooms. Sebenarnya terdapat satu dokumen lagi yaitu dokumen Users. Tapi dokumen Users sudah otomatis ada pada project Meteor kita tanpa kita definisikan. Yang dapat digunakan dengan package accounts-password.

10. Handling Routing Dengan Iron:Router

```

router.js
Router.configure({
  layoutTemplate: 'main'
});
Router.route('/', {
  name: 'layout'
});
Router.route('/signup', {
  name: 'signup'
});
Router.route('/home', {
  name: 'home'
});

Router.route('/chatroom/:_id/name/:chatroom_name', {
  name: 'chatroom',
  data: function() {
    return {
      chatroomName: this.params.chatroom_name,
      chat: Messages.find({
        chatRoomId: this.params._id
      },
      { sort:
        { timestamp: 1 }}
    )
  }
});

```


Yang perlu diperhatikan :

```
Router.route('/chatroom/:_id/name/:chatroom_name', {
  name: 'chatroom',
  data: function() {
    return {
      chatroomName: this.params.chatroom_name,
      chat: Messages.find({
        chatRoomId: this.params._id
      },
      { sort:
        { timestamp: 1 }}
    )
  };
});
```

Ini berarti ketika user mengakses halaman chatroom diperlukan 2 parameter. Yaitu parameter `_id` dan parameter `chatroom_name`. Dan pada template chatroom terdapat return 2 data yaitu `chatroomName` dan data `chat`. Perhatikan file `chatroom.html` terdapat kodingan seperti di bawah ini :

```
st Chatroom</a></li>
</ul>
<a href="{{pathFor route='home'}}" >
<h2>{{chatroomName}}</h2>
<a href="#" id="Logout">Log Out</a>
</div>
```

Perhatikan `{{chatroomName}}` merupakan data return yang kita definisikan di `router.js`

```

    {{#each chat}}
    <div class="row">
      <div class="col-lg-12">
        <div class="media">
          <div class="media-body">
            <h4 class="media-heading">{{username}}
            <span class="small
pull-right">{{formatDate timestamp}}</span>
          </h4>
          <p>
            {{text}}
          </p>
        </div>
      </div>
    </div>
  </div>
</div>
```

Begitu juga dengan `{{ #each chat }}` merupakan data return yang kita definisikan di `router.js` yang kita dapat data tersebut dari database dengan query berikut :

```
chatroomName: this.params.chatroom_name,
chat: Messages.find({
  chatRoomId: this.params._id
},
{ sort:
  { timestamp: 1 }}
)
```

11. Tampilan aplikasi

Untuk memulai aplikasi ketikkan **meteor** di terminal. Setelah proses build selesai. Akses browser dengan alamat *localhost:3000*. Maka akan tampil seperti di bawah ini.

Tampilan awal meminta user untuk login.

MikroskilChat

Aplikasi Chatting dengan Meteor Js dan MongoDB

Masuk

Masukkan username dan password anda

Sign in!

Don't have account?

[Create account](#)

Laman untuk pendaftaran.

MikroskilChat

Aplikasi Chatting dengan Meteor Js dan MongoDB

Daftar

Daftar dan chat teman anda

Sign Up

Laman Chatting.

Mikroskil Chatting

[Create Chatroom](#) [List Chatroom](#)

sio

[Log Out](#)

heru

hallo

6/29/2018, 11:31:53 AM

heru

ini sio

6/29/2018, 11:31:59 AM

Enter message...

Send

Jendela Buat Chat Room

Create Chatroom

×

Chatroom name

Create

Close

[Create Chatroom](#)

[List Chatroom](#)

6/29/2018, 11:31:53 AM

List Chat Room

List Available Chatrooms

×

Web Lanjutan

imk

Close

Chatting dalam chat room

[Back to Home](#)

[Create Chatroom](#)

[List Chatroom](#)

Web Lanjutan

[Log Out](#)

Sio

6/29/2018, 10:40:42 PM

Hallo ini adalah pesan

[Delete](#)

SIO

6/29/2018, 10:41:02 PM

Sebuah pesan

[Delete](#)

Enter message...

Send