Bab I: Menginstall ExtJs 6

Bab II: Membuat Aplikasi ExtJs 6

Bab III: Membuat Class ExtJs

Bab IV: Penggunaan *xtype* dan *Ext.create*

Bab V: Metode MVC

Bab VI: Parameter Layout

Bab VII: Komponen Bidang

Bab VIII: Komponen Input

Bab IX: Grid dan TreeGrid

Bab X: Membuat Web Desktop

Bab I

Menginstall ExtJs 6

Untuk menginstall Extjs perlu dipersiapkan dua komponen utama yaitu sebagai berikut:

1. ExtJs SDK

Extjs SDK yang akan digunakan adalah ExtJs versi 6. Boleh menggunakan versi 6.5.3 yaitu versi trial yang hanya bisa digunakan 30 hari. Atau menggunakan versi GPL nya yaitu versi 6.2.0. Untuk mendownload versi 6.5.3 silahkan klik link berikut:

<https://www.sencha.com/products/extjs/evaluate/>

Isi semua kolom pada form download. Setelah itu link download akan diberikan melalui email.

Untuk mendownload versi 6.2.0 silahkan klik link berikut.

<https://www.sencha.com/legal/GPL/>

Isi semua kolom pada form download. Setelah itu link download akan diberikan melalui email.

Atau jika tidak ingin mendownloadnya. File ExtJs SDK disertakan didalam CD. Untuk pembahasan selanjutnya saya akan menggunakan ExtJs versi GPL yaitu versi 6.2.0.

1. Sencha Cmd

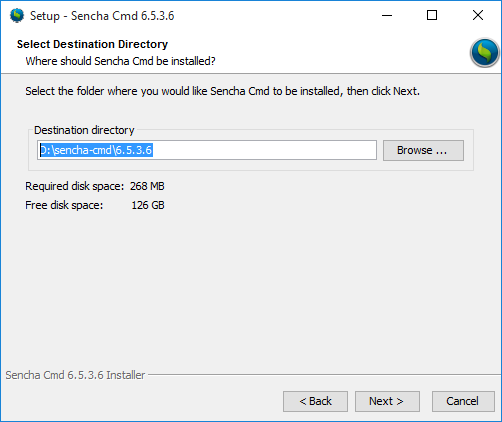
Adalah komponen untuk mengcompile dan menjalankan ExtJs SDK. Versi yang digunakan adalah versi 6.5.3. Untuk mendownload Sencha Cmd silahkan klik link berikut.

<https://www.sencha.com/products/extjs/cmd-download/>

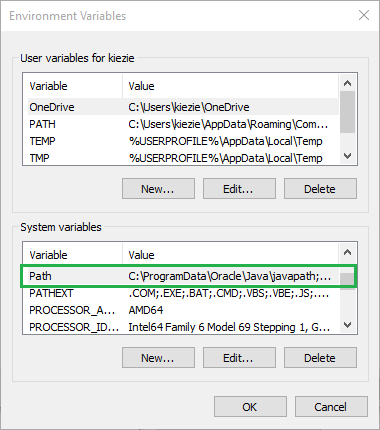
Atau jika tidak ingin mendownloadnya. File Sencha Cmd disertakan didalam CD.

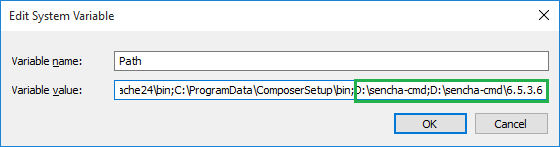
Setelah berhasil mendapatkan kedua komponen diatas selanjutnya adalah langkah-langkah pengistallannya sebagai berikut:

1. Extract File ExtJs SDK versi 6.2.0 pada folder yang diinginkan. Saya akan meletakkannya pada direktori ***D:\sencha-sdks***. Jika Folder sencha-sdks belum ada maka buat folder sencha-sdks pada direktori D:\
2. Mengistall Sencha Cmd. Ikuti langkah-langkah penginstallan hingga selesai. Saat penginstallan letakkan pada direktori ***D:\sencha-cmd\6.5.3.6*** seperti gambar berikut ini:

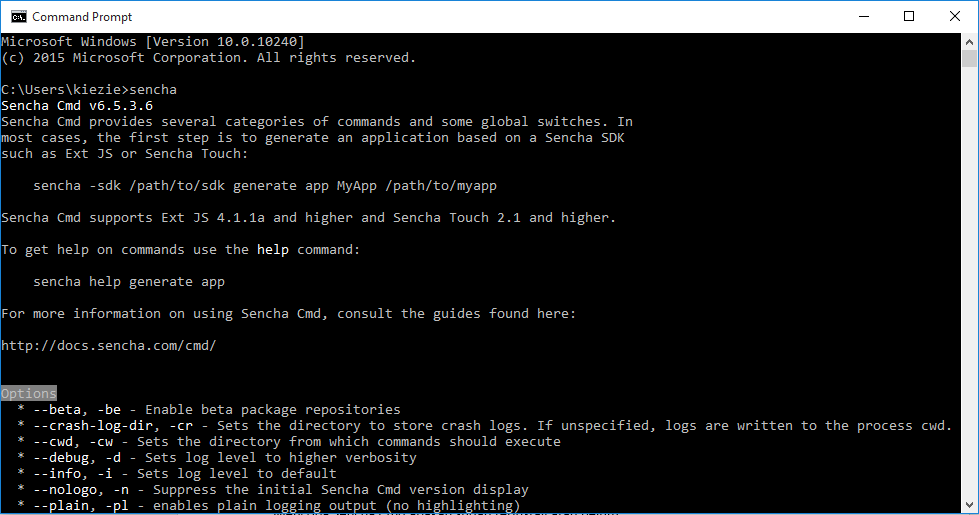


1. Menambahkan Sencha CMD ke variable environment. Saat proses installasi Sencha CMD akan ditambahkan kedalam variable environment. Tetapi jika tidak berhasil maka kita bisa menambahkannya sendiri secara manual dengan menamabahkan folder ***D:\sencha-cmd*** dan ***D:\sencha-cmd\6.5.3.6*** pada System variables yaitu variable ***PATH*** seperti gambar berikut.

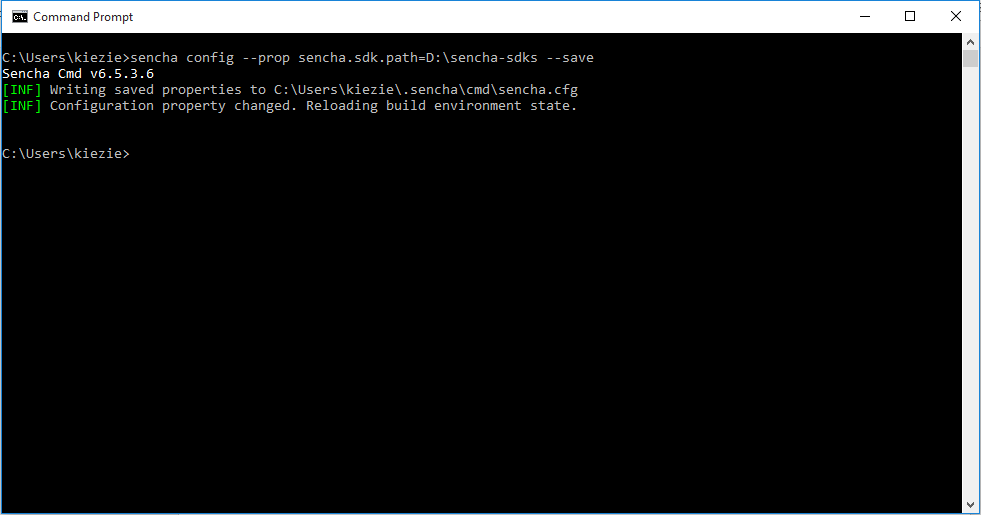




1. Setelah selesai setting variable environment selanjutnya membuka terminal command prompt. Ketik ***sencha***. Jika muncul seperti gambar berikut ini berarti Anda telah berhasil menginstall Sencha CMD.



Selanjutnya adalah melakukan konfigurasi Sencha CMD sesuai dengan letak folder ExtJs SDK yang tadi diextract di ***D:\sencha-sdks***.

*sencha config --prop sencha.sdk.path=D:\sencha-sdks –save*

Sampai disini kita sudah berhasil mengkonfigurasikan sencha Cmd dengan ExtJS SDK 6.2.0. Selanjutnya adalah membuat inisialisasi aplikasi dengan Sencha Cmd.

Buatlah sebuah direktori tempat aplikasi yang akan diciptakan oleh sencha cmd. Untuk membuat direktori gunakan perintah ***md***. Dan Saya akan membuatnya di direktori ***D:\www\myapp***.

*>D: [enter]*

*>md www [enter]*

*>cd www [enter]*

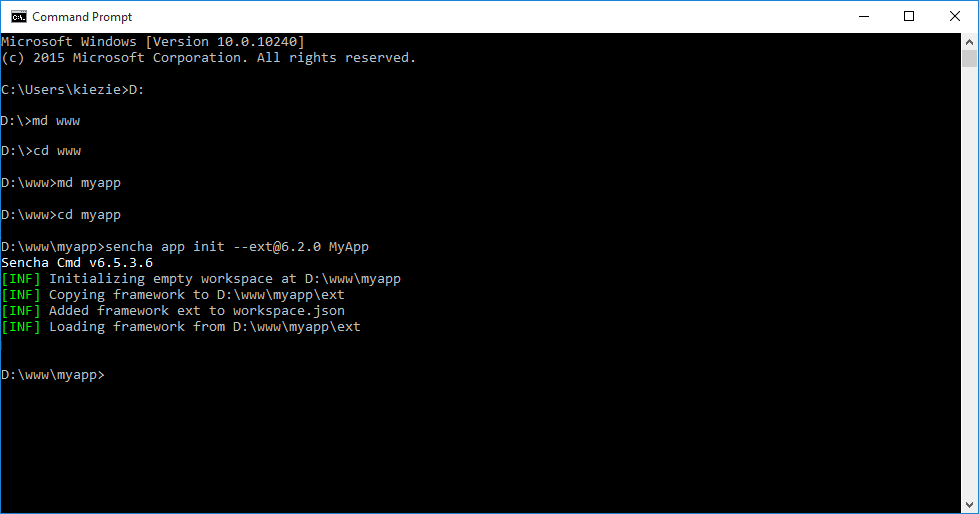
*>md myapp [enter]*

Kemudian masuk kedalam direktori ***myapp*** dengan perintah ***cd***.

*>cd myapp [enter]*

Pada direktori ***D:\www\myapp*** buatlah perintah:

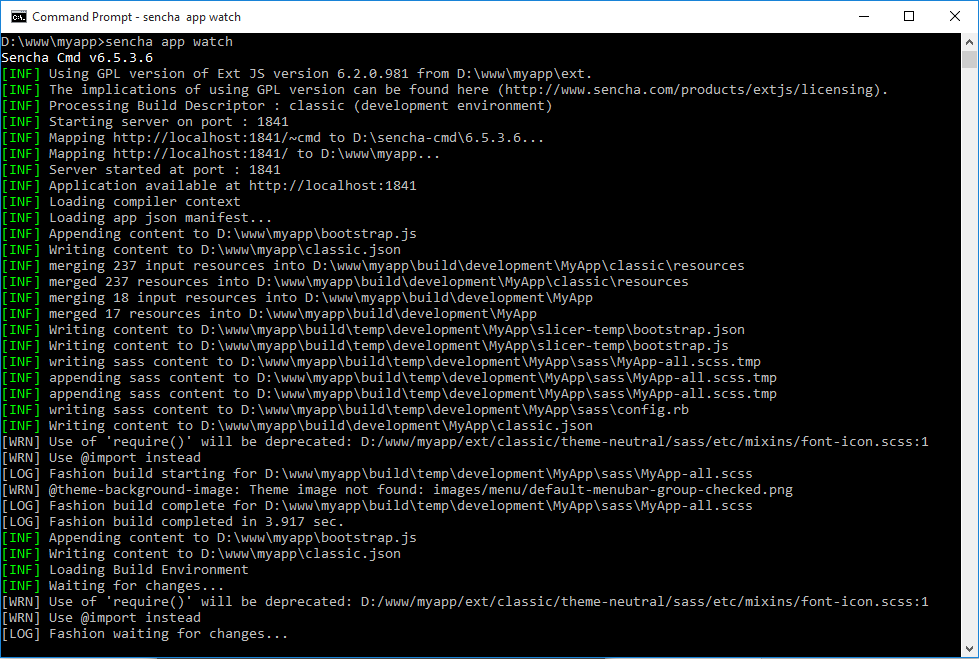
*sencha app init --ext@6.2.0 MyApp*



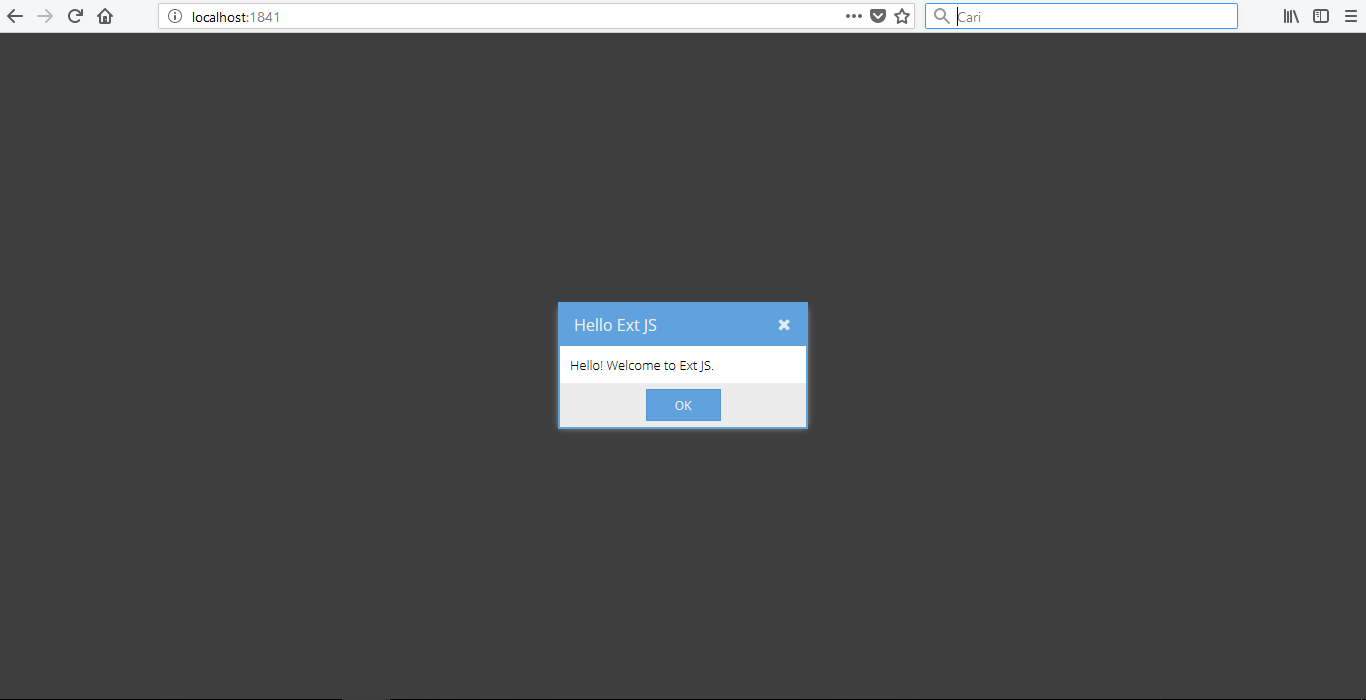
Perintah diatas adalah membuat sebuah aplikasi bernama ***MyApp*** dengan template ***modern*** dan menggunakan SDK ExtJs 6.2.0 yang diletakkan di folder ***D:\www\myapp***.

Untuk melihat hasil sudah dibuat di folder ***D:\www\myapp*** yaitu dengan menjalankan perintah berikut dengan posisi direktori beradi di ***D:\www\myapp***.

*sencha app watch*

****

Jika tampilan layar seperti itu maka aplikasi bisa dilihat pada browser dengan memanggil alamat address [***http://localhost:1841***](http://localhost:1841) seperti gambar berikut ini.



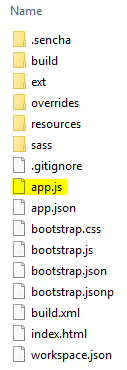
Atau jika pada komputer sudah terinstall webserver seperti contoh webserver *apache* maka bisa langsung dipanggil melalui *localhost* dengan alamat tempat aplikasi tersebut diatas. Sebagai contoh komputer saya sudah terinstall apache yang mana *localhost* nya berada di ***D:/www*** dan aplikasi yang dibuat tadi ada di ***D:/www/myapp***. Maka bisa saya panggil melalui [***http://localhost/myapp***](http://localhost/myapp).

Bab II

Membuat Aplikasi ExtJs 6

Setelah berhasil menginstall ExtJs dan menjalankan aplikasi awal yang dibuat melalui Sencha Cmd selanjutnya kita akan memodifikasi atau membuat aplikasi sendiri dari aplikasi yang sudah dibuat pada bab Sebelumnya.

Pada folder ***D:/www/myapp*** tempat aplikasi dibuat terdapat file ***app.js*.** Adalah file permulaan untuk menjalanlan sebuah aplikasi ExtJS.

Yang berisi:

Ext.application({

name: 'MyApp',

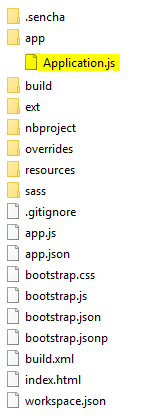
launch: function () {

Ext.Msg.alert('Hello Ext JS', 'Hello! Welcome to Ext JS.');

}

});

Buatlah sebuah folder ***app*** pada di dalam folder aplikasi ***D:/www/myapp*** kemudian buat file bernama ***Application.js*** yang berisi:



Ext.define('MyApp.Application', {

extend: 'Ext.app.Application',

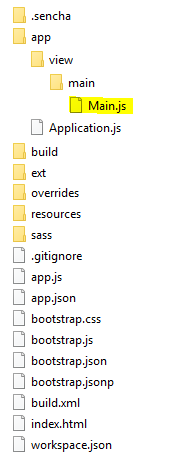
mainView: 'MyApp.view.main.Main'

});

Ini adalah class yang ber-*extend* sebagai *Ext.app.Application* sebagai aplikasi dasar dengan menambahkan parameter *mainView* untuk memerintahkan aplikasi agar menampilkan class *MyApp.view.main.Main* terlebih dahulu.

Selanjutnya adalah membuat class *MyApp.view.main.Main*.

Buatlah file ***Main.js*** pada folder ***D:/www/myapp/app/view/main*** dengan isi sebagai berikut:

Ext.define('MyApp.view.main.Main', {

extend: 'Ext.tab.Panel',

controller: 'main',

items: [{

title: 'Home',

html: '<h1 class="main-banner">Hello World!</h1>',

items: [{

xtype: 'button',

text: 'Go',

handler: 'onGo'

}]

}, {

title: 'Notifications'

}, {

title: 'Settings'

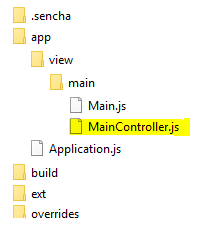
}]

});

*Class* dengan *extend* sebagai *Ext.tab.Panel* akan menampilkan sebuah tampilan berupa Tab Panel. Tab Panel adalah sebuah komponen wadah khusus yang mengandung komponen lainnya. Komponen anak biasanya disebut *items* ditambahkan ke wadah tersebut dengan memberikan config atau parameter *items*. Di atas, Tab Panel sebagai wadah dasar dan menambahkan bilah tab untuk memungkinkan pengguna beralih antara item anak-anaknya.

Menambahkan sebuah *controller* untuk class diatas adalah berfungsi untuk memberikan *event* terhadap komponen yang diinginkan. Kita akan membuat controller sebagai *event* dari komponen tombol ***GO*** diatas.

Buatlah file ***MainController.js*** pada folder ***D:/www/myapp/app/view/main*** dengan isi sebagai berikut:



Ext.define('MyApp.view.main.MainController', {

extend: 'Ext.app.ViewController',

alias: 'controller.main',

onGo: function () {

Ext.Msg.alert('Go', 'From main view controller');

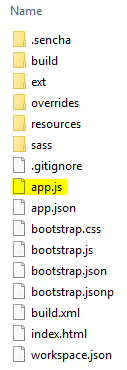
}

});

*Class* diatas memerintahkan kepada tombol ***GO*** untuk menampilkan sebuah *message dialog*. Yaitu dengan menambahkan parameter fungsi ***onGo*** pada *class* *controller* yang disesuai dengan komponen tombol pada file ***Main.js*** dengan config ***handler: 'onGo'***.

Agar *class* ini dapat berjalan maka pada file ***Main.js*** harus menambahkan config ***controller: ‘main’***. Sesuai dengan penulias alias pada class controllernya yaitu pada file ***MainController.js***.

Selanjutnya adalah merubah file ***app.js*** pada folde ***D:/www/myapp*** seperti dibawah ini.

Ext.application({

name: 'MyApp',

extend: 'MyApp.Application',

requires: [

'MyApp.\*'

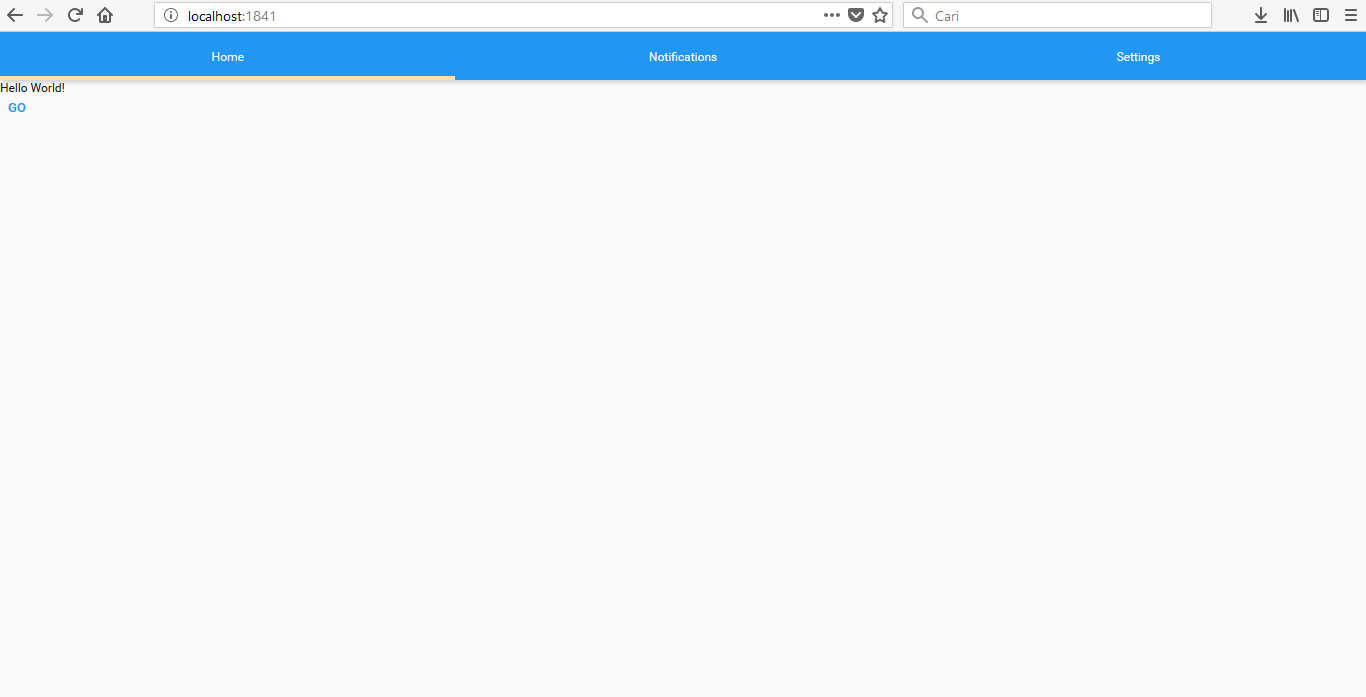
]

});

Yaitu memeritahkan agar menjankan class Application.js yang ada di folder ***D:/www/myapp***. Parameter ***extend: ‘MyApp.Application’***pada *class* diatas adalah menyatakan bahwa class ***app.js*** adalah sebuah class ***Application.js***. Sehingga saat ***app.js*** dijalankan secara otomatis akan menjalankan class ***Application.js***.

Parameter *requires* berfungsi memberitahukan kepada Sencha CMD untuk menambahkan semua *class* yang ada di ***D:/www/myapp/app***.

Untuk melihat hasil aplikasi yang sudah dibuat pada browser dengan mengetik url [***http://localhost:1841***](http://localhost:1841) pada browser maka akan tampil seperti gambar berikut ini.



Perhatian:

Setelah melakukan perubahan class. Jika perintah *sencha app watch* tidak dalam keadaan aktif. Maka harus diaktifkan atau dijalankan dahulu. Atau dalam melakukan perubahan *class* sebaiknya sudah terlebih dahulu menjalankan *sencha app watch*. Dengan demikian sencha cmd akan secara otomatis mengatur komponen-komponen aplikasi yang diperlukan.

Bab III

Membuat Class ExtJs

Di dalam pemrograman ExtJs Aplikasi dibuat menggunakan *class* yang ditulis kedalam suatu file. Untuk membuat (mendefinisikan) *class* pada ExtJs penulisannya adalah sebagai berikut:

*Ext.define([Nama Class], [Config]);*

penulisan diatas adalah sebuah format penulisan class dalam ExtJs dengan masing-masing penjelasan parameter sebagai berikut:

parameter pertama adalah *[Nama Class]* mendefinisikan sebuah nama *class* dan merupakan tipe *string*.

parameter kedua adalah *[Config]* mendefinisikan anggota *class* dan merupakan tipe *object* dengan semua properti dan metode yang dibutuhkan pada *class* tersebut.

Contoh *class*:

Ext.define(

'MyApp.Application',

{

extend: 'Ext.app.Application',

mainView: 'MyApp.view.main.Main'

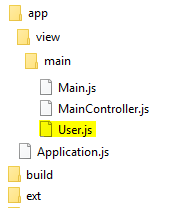
}

);

Yang berlabel warna kuning adalah *[Nama Class]* dan yang berlabel warna hijau adalah *[Config]* yang berupa *object*.

Berikut ini adalah contoh *class* berbentuk *form window* untuk mengisi sebuah data:

1. Buat file dengan nama ***User.js*** pada folder ***D:/www/myapp/app/view/main***.
2. Pada file ***User.js*** buat class bernama ***‘MyApp.view.main.User’***.



Ext.define('MyApp.view.main.User', {

...

});

Penulisan nama class harus sesuai dengan nama file dan letak folder yang ditempatkan. ***‘MyApp.view.main.User’*** berarti file nya bernama ***User.js***. Karena Aplikasi bernama ***MyApp*** secara default Extjs Menempatkan nama ***MyApp*** pada Folder ***D:/www/myapp/app***. Dan diteruskan folder selanjutnya yaitu ***view*** dan ***main***. Jadi jika file ***User.js*** yang berada pada folder ***D:/www/myapp/app/view/main/User.js***. Maka penulisan nama classnya adalah ***‘MyApp.view.main.User’***. Lihat penyesuain warna antara nama class dan letak filenya.

1. Mengisi parameter *[config]* berupa *object*. Akan kita tambahkan 2 parameter yaitu *extend* dan *alias*. Atribut pertama yaitu parameter *extend*. *Extend* adalah parameter yang menjelaskan bahwa *class* tersebut adalah suatu komponen tertentu. Karena saya akan membuat sebuah *window* maka *extend*-nya adalah *‘Ext.window.Window’*. Sehingga class menjadi seperti berikut. Dan parameter kedua adalah *alias* adalah parameter untuk menggantikan nama class menjadi lebih sederhana. Misalnya class ini kita beri nama *alias*-nya adalah *user* penulisannya menjadi *‘widget.user’*. Sehingga penulisan classnya menjadi.

Ext.define('MyApp.view.main.User', {

extend: 'Ext.window.Window',

alias: 'widget.user'

});

1. Selanjutnya menambahkan parameter yang lain seperti judul, lebar, tinggi, *layout* dan pemberian *border* pada window.

Ext.define('MyApp.view.main.User', {

extend: 'Ext.window.Window',

alias: 'widget.user',

title: 'User',

width: 400,

height: 400,

layout: 'fit',

border: false

});

Sampai disini Kita sudah berhasil membuat komponen *window* sebagai komponen induk. Jika Kita ingin menampilkan *class* ***User.js*** maka lakukan perubahan pada file ***app.js*** yang ada pada folder ***D:/www/myapp***. Seperti berikut:

Ext.application({

name: 'MyApp',

requires: [

'MyApp.\*'

],

launch: function() {

Ext.create('MyApp.view.main.User').show();

}

});

Untuk memanggil atau menggunakan sebuah *class* bisa menggunakan perintah sebagai berikut.

*Ext.create([Nama Class]);*

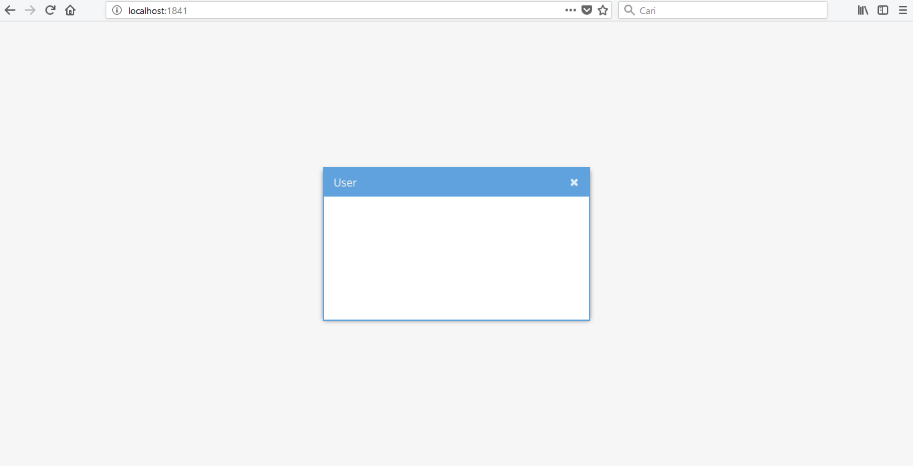
Pada fungsi *launch* file ***app.js*** diatas melakukan pemanggilan *class* ***'MyApp.view.main.User'*** dengan menuliskan:

*Ext.create('MyApp.view.main.User')*

Setelah itu karena sebagai komponen *window* dan ingin ditampilkan maka dilanjutkan dengan perintah *show()*. Maka perintahnya menjadi.

*Ext.create('MyApp.view.main.User').show();*

Untuk melihat hasilnya yaitu dengan mengetik url [***http://localhost:1841***](http://localhost:1841) pada maka akan tampil seperti gambar berikut ini.



1. Setelah berhasil membuat komponen *window* sebagai komponen induk. Selanjutnya menambahkan komponen anak menggunakan parameter *items*. Komponen anak yang akan ditambahkan adalah komponen *form*. Sehingga *class* ***User.js*** menjadi seperti berikut ini:

Ext.define('MyApp.view.main.User', {

extend: 'Ext.window.Window',

alias: 'widget.user',

title: 'User',

width: 400,

height: 400,

layout: 'fit',

border: false,

items: [{

xtype: 'form',

layout: 'anchor',

border: true,

bodyPadding: 10

}]

});

Atau untuk komponen *form* bisa ditulis juga seperti berikut.

Ext.define('MyApp.view.main.User', {

extend: 'Ext.window.Window',

alias: 'widget.user',

title: 'User',

width: 400,

height: 400,

layout: 'fit',

border: false,

items: [Ext.create('Ext.form.Panel', {

layout: 'anchor',

border: true,

bodyPadding: 10

})]

});

1. Menambahkan komponen *input* sebagai komponen anak dari komponen *form* dengan menggunakan parameter *items*. Komponen Input yang akan ditambahkan antara lain:

6.a. Text Field

{

xtype: 'textfield',

name: 'nama',

fieldLabel: 'Nama',

anchor: '100%'

}

6.b. Combo Box

{

xtype: 'combobox',

name: 'pendidikan',

fieldLabel: 'Pendidikan',

store: Ext.create('Ext.data.Store', {

fields: ['kode'],

data: [

{'kode': 'S1'},

{'kode': 'S2'},

{'kode': 'S3'}

]

}),

valueField: 'kode',

displayField: 'kode',

queryMode: 'local',

typeAhead: true,

anchor: '100%'

}

6.c. Check Box

{

xtype: 'checkbox',

name: 'status',

boxLabel: 'Aktif',

checked: true,

fieldLabel: 'Status',

anchor: '100%'

}

6.d. Radio Button

{

xtype: 'fieldcontainer',

fieldLabel: 'Jenis Kelamin',

anchor: '100%',

defaultType: 'radiofield',

defaults: {flex: 1},

style: 'border-width: 0px',

layout: 'hbox',

items: [{

boxLabel: 'Laki-laki',

name: 'jeniskelamin',

inputValue: 'L'

}, {

boxLabel: 'Perempuan',

name: 'jeniskelamin',

inputValue: 'P'

}]

}

6.e. Text Area

{

xtype: 'textarea',

name: 'alamat',

fieldLabel: 'Alamat',

anchor: '100%',

height: 40

}

Setelah ditambahkan komponen *input* pada komponen *form* maka class ***User.js*** menjadi:

Ext.define('MyApp.view.main.User', {

extend: 'Ext.window.Window',

alias: 'widget.user',

title: 'User',

width: 400,

height: 400,

layout: 'fit',

border: false,

items: [Ext.create('Ext.form.Panel', {

layout: 'anchor',

border: true,

bodyPadding: 10,

items: [{

xtype: 'textfield',

name: 'nama',

fieldLabel: 'Nama',

anchor: '100%'

}, {

xtype: 'combobox',

name: 'pendidikan',

fieldLabel: 'Pendidikan',

store: Ext.create('Ext.data.Store', {

fields: ['kode'],

data: [

{'kode': 'S1'},

{'kode': 'S2'},

{'kode': 'S3'}

]

}),

valueField: 'kode',

displayField: 'kode',

queryMode: 'local',

typeAhead: true,

anchor: '100%'

}, {

xtype: 'checkbox',

name: 'status',

boxLabel: 'Aktif',

checked: true,

fieldLabel: 'Status',

anchor: '100%'

}, {

xtype: 'fieldcontainer',

fieldLabel: 'Jenis Kelamin',

anchor: '100%',

defaultType: 'radiofield',

defaults: {flex: 1},

style: 'border-width: 0px',

layout: 'hbox',

items: [{

boxLabel: 'Laki-laki',

name: 'jeniskelamin',

inputValue: 'L'

}, {

boxLabel: 'Perempuan',

name: 'jeniskelamin',

inputValue: 'P'

}]

}, {

xtype: 'textarea',

name: 'alamat',

fieldLabel: 'Alamat',

anchor: '100%',

height: 40

}]

})]

});

1. Menambahkan Tombol pada Komponen Induk *Window*.

Ext.define('MyApp.view.main.User', {

extend: 'Ext.window.Window',

alias: 'widget.user',

title: 'User',

width: 400,

height: 400,

layout: 'fit',

border: false,

items: [Ext.create('Ext.form.Panel', {

layout: 'anchor',

border: true,

bodyPadding: 10,

items: [

...

]

})],

buttons: [{

text: 'Simpan',

action: 'simpan'

}, {

text: 'Batal',

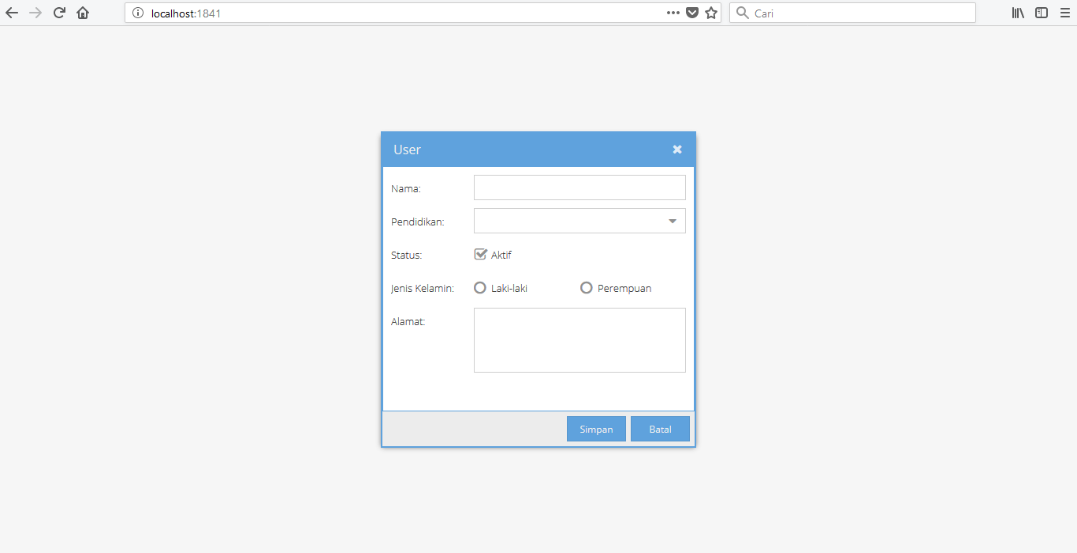
scope: this,

handler: this.close

}]

});

Dan jika aplikasi dijalankan maka *class* **User.js** akan menampilkan seperti gambar dibawah ini:



Bab IV

Penggunaan *xtype* dan *Ext.create*

Sebagai mana yang telah kita ketahui penggunaan *xtype* dan *Ext.create* adalah untuk memanggil atau menggunakan class yang sudah dibuat. Akan tetapi walaupun fungsinya sama akan tetapi ada perbedaan dalam menggunakannya.

1. Penggunaan *xtype*

*xtype* adalah sebuah parameter untuk menggunakan *class* tertentu dan pemakaiannya hanya sebagai config pada komponen anak yaitu didalam parameter *items*. pemanggilan class dengan *xtype* harus menggunakan nama alias dari *class* yang akan digunakan. Sebagai contoh berikut ini:

Ext.define('MyApp.view.main.User', {

extend: 'Ext.window.Window',

alias: 'widget.user',

layout: 'fit',

items: [{

xtype: 'login',

layout: 'anchor',

items: [

...

]

}]

});

Penting untuk menggunakan *xtype,* *class* yang akan digunakan tidak dapat langsung menuliskan *alias*-nya. Akan tetapi selain komponen-komponen yang sudah tersedia di ExtJs, *class* lain yang dibuat sendiri harus disertakan dahulu pada aplikasi dengan perintah *requires*. Seperti yang dituliskan pada file ***app.js*** dibawah ini:

Ext.application({

name: 'MyApp',

requires: [

'MyApp.\*'

],

launch: function(profile) {

Ext.create('MyApp.view.main.User').show();

}

});

Kode diatas yang diblok warna kuning adalah untuk menyertakan semua *class*. Untuk menyertakan *class* tertentu dapat menuliskannya seperti berikut ini.

Ext.application({

name: 'MyApp',

requires: [

'MyApp.view.main.Login'

],

launch: function(profile) {

Ext.create('MyApp.view.main.User').show();

}

});

1. Penggunaan *Ext.create*

*Ext.create* adalah suatu perintah untuk menciptakan sebuah komponen dari suatu *class*. Karena berbentuk perintah sehingga *Ext.create* dapat dijadikan suatu *variabel*. Dengan terciptanya suatu *variabel* sehingga dapat dengan *flexibel* untuk digunakan dalam keperluan tertentu. Sebagai contoh berikut ini:

var formLogin = Ext.create('MyApp.view.main.Login', {

layout: 'anchor',

items: [

...

]

});

Contoh diatas adalah sebuak komponen yang disimpan didalam sebuah variabel *formLogin*. Dan variabel itu dapat digunakan didalam kode berikut:

Ext.define('MyApp.view.main.User', {

extend: 'Ext.window.Window',

alias: 'widget.user',

title: 'User',

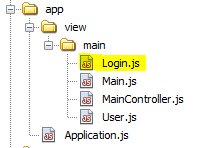
layout: 'fit',

items: [formLogin]

});

Perbedaan dengan parameter *xtype* diatas adalah. *Class* yang dibuat sendiri dan akan digunakan tidak harus disertakan pada file ***app.js***.

Contoh sebuah file ***Login.js*** yang berada di folder ***D:/www/myapp/app/view/main/*** adalah sebuah *class* yang merupakan sebuah form login.



Ext.define('MyApp.view.main.Login', {

extend: 'Ext.form.Panel',

alias: 'widget.login',

layout: 'anchor',

border: true,

bodyPadding: 10,

items: [{

xtype: 'textfield',

fieldLabel: 'Id User',

name: 'username',

anchor: '100%'

}, {

xtype: 'textfield',

inputType: 'password',

fieldLabel: 'Password',

name: 'password',

anchor: '100%'

}]

});

Dengan menggunakan parameter *xtype* untuk menggunakan *class login* diatas, file ***app.js*** pada folder ***D:/www/myapp*** dapat dituliskan sebagai berikut.

Ext.application({

name: 'MyApp',

requires: [

'MyApp.\*'

],

launch: function() {

Ext.create('Ext.window.Window', {

title: 'User Login',

width: 400,

height: 300,

border: false,

layout: 'fit',

items:[{

xtype: 'login'

}],

buttons: [{

text: 'Login',

action: 'login'

}, {

text: 'Cancel',

action: 'cancel'

}]

}).show();

}

});

Dan jika ingin menggunakan perintah *Ext.create*dapatdituliskan sebagai berikut.

Ext.application({

name: 'MyApp',

launch: function() {

var formLogin = Ext.create('MyApp.view.main.Login');

var winLogin = Ext.create('Ext.window.Window', {

title: 'User Login',

width: 400,

height: 300,

border: false,

layout: 'fit',

items:[formLogin],

buttons: [{

text: 'Login',

action: 'login'

}, {

text: 'Cancel',

action: 'cancel'

}]

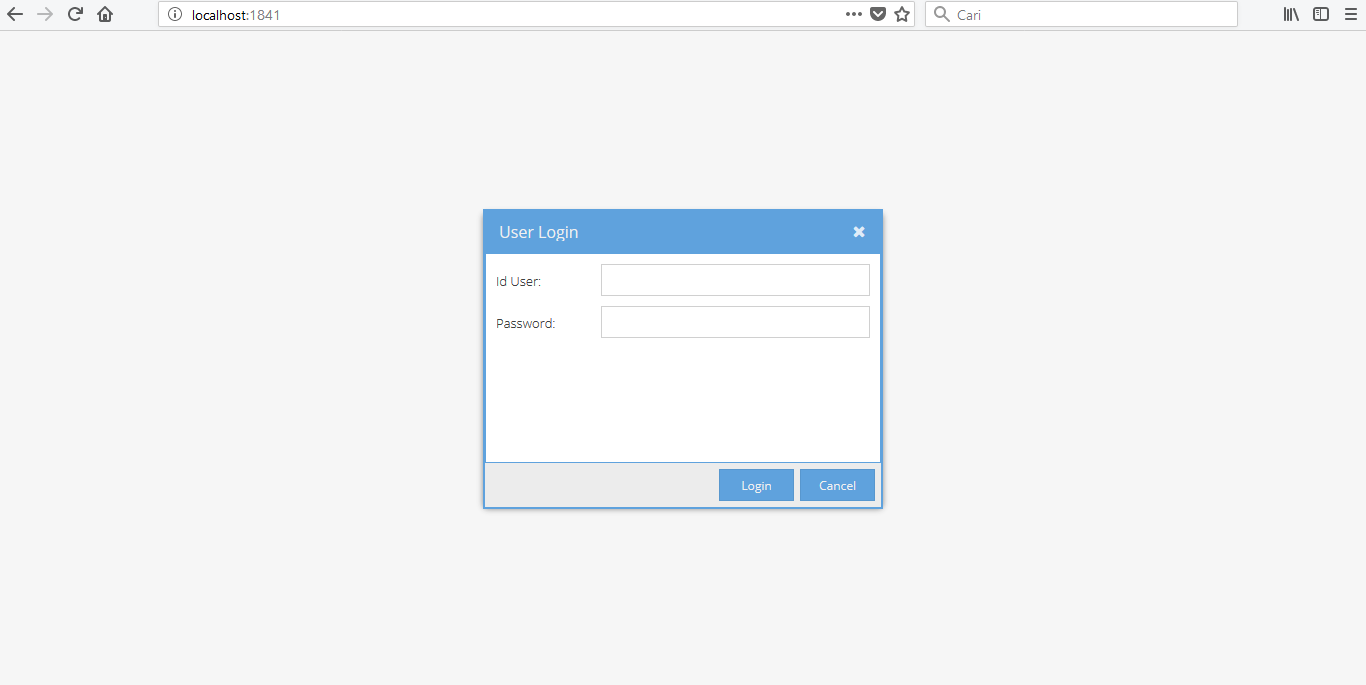
});

winLogin.show();

}

});

Hasilnya akan tampak seperti gambar berikut ini.



Bab V

Metode MVC

MVC atau Model View Controller adalah sebuah metode untuk membuat sebuah aplikasi dengan memisahkan data (Model) dari tampilan (View) dan cara bagaimana memprosesnya (Controller). Dalam implementasinya kebanyakan framework dalam aplikasi website adalah berbasis arsitektur MVC.

Penerapan MVC ini juga bisa digunakan di ExtJs. Berikut ini adalah penjelasan bagaiman membuat aplikasi menggunakan ExtJs dengan metode MVC.

*Model* berfungsi sebagai interaksi langsung  antara aplikasi dengan data. Model menggambarkan format umum untuk data yang digunakan dalam aplikasi. Didalam *Model* terdapat *field-field* data yang perlu ditampilkan dalam betuk nama dan tipe *field*.

Contoh *Model*:

Ext.define('UserModel', {

extend: 'Ext.data.Model',

fields: [

{name: 'nama', type: 'string'},

{name: 'umur', type: 'int'},

{name: 'telepon', type: 'string'},

{name: 'status', type: 'boolean'}

]

});

*Store* sebagai pendukung dari *Model* yang digunakan untuk mengambil data dari suatu *server* yang dibutuhkan *Model*.

Contoh *Store*:

Ext.create('Ext.data.Store', {

model: 'User',

autoLoad: true,

proxy: {

type: 'ajax',

url: '/users.json',

reader: {

type: 'json',

root: 'users'

}

}

});

*View* adalah *interface* atau tampilan sebuah aplikasi yang mewakili data ke pengguna. Beberapa tampilan mungkin menampilkan data yang sama dengan cara yang berbeda misalnya grafik atau *grid*.

Contoh *View*:

Ext.create('Ext.grid.Panel', {

title: 'Users',

store: Ext.create('Ext.data.Store', {

model: 'User',

autoLoad: true,

proxy: {

type: 'ajax',

url: '/users.json',

reader: {

type: 'json',

root: 'users'

}

}

}),

columns: [

{ text: 'Nama', dataIndex: 'nama' },

{ text: 'Umur', dataIndex: 'umur', flex: 1 },

{ text: 'Telepon', dataIndex: 'telepon' }

],

height: 200,

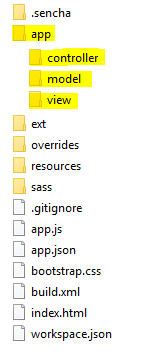
width: 400,

renderTo: Ext.getBody()

});

*Controller* adalah bagian utama dari aplikasi MVC. *Controller* mengatur sebuah *events* dalam perintah aplikasi dan delegasi antara *Model* dan *View*.

Dalam pembuatan aplikasi MVC umumnya struktur penempatan folder dan file adalah sebagai berikut:

Folder *controller* akan berisi file-file *class* *controller*.

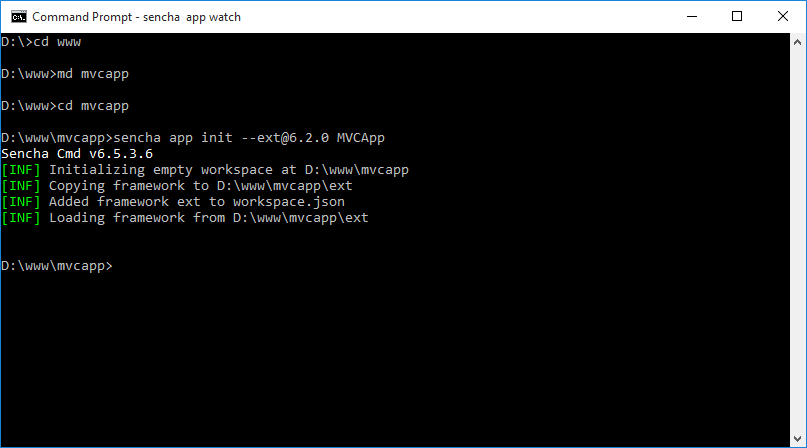
Folder *model* akan berisi file-file *class* *model*.

Folder *view* akan berisi file-file *class* *view*.

Berikut ini adalah contoh langkah-langkah membuat aplikasi dengan metode MVC.

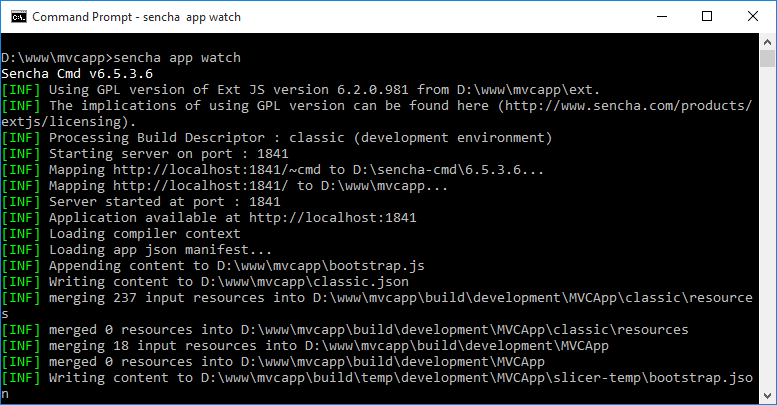
1. Buatlah sebuah folder dengan nama ***mvcapp*** pada direktori yang diinginkan. Sebagai contoh saya buat di dalam direktori ***D:/www***. Kemudian masuk kedalam folder ***D:/www/mvcapp***.
2. Setelah berada di folder ***D:/www/mvcapp*** lakukan inisialisasi aplikasi menggunakan sencha CMD dengan nama aplikasi ***MVCApp*** dengan perintah berikut ini.

*sencha app init --ext@6.2.0 MVCApp*



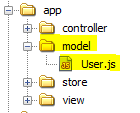
1. Setelah berhasil inisialisasi aplikasi ***MVCApp***. Kemudian jalankan aplikasi tersebut dengan perintah berikut ini.

*sencha app watch*



1. Setelah aplikasi dapat diakses melalui alamat <http://localhost:1841> selanjutnya adalah melakukan perubahan pada aplikasi tersebut.

Buatlah sebuah *Model* dengan nama file ***User.js*** pada folder ***D:\www\mvcapp\app\model*** yang isinya sebagai berikut.



Ext.define('MVCApp.model.User', {

extend : 'Ext.data.Model',

fields : [{

name : 'nama',

type : 'string'

}, {

name : 'email',

type : 'string'

}, {

name : 'telepon',

type : 'string'

}, {

name : 'id',

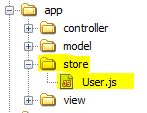
type : 'string'

}]

});

Kode diatas adalah menyiapkan sebuah data *model* dengan filed *nama*, *email*, *telepon*, dan *id*.

1. Membuat *store* data untuk menarik data sesuai dengan filed pada *model* diatas. Kita buat file store dengan nama ***User.js*** pada folder ***D:\www\mvcapp\app\store*** dengan isi file seperti berikut.



Ext.define('MVCApp.store.User', {

extend: 'Ext.data.Store',

storeId: 'User',

model: 'MVCApp.model.User',

autoLoad: true,

proxy: {

type: 'ajax',

url: 'data/User.json',

reader: {

type: 'json',

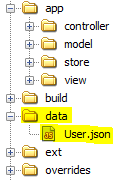
root: 'topics'

}

}

});

Class diatas berfungsi untuk melakukan pengambilan data melalui akses url *'data/User.json'* yang berlabel warna hijau diatas yaitu pada ***D:\www\mvcapp\data\User.json***. Dengan format data berupa *JSON* dengan mengambil data yang ber-parameter *'topics'* dari hasil respon akses url *data/User.json*. Seperti hasil berikut ini dari url *'data/User.json'*.



{

"topics": [{

"nama" : "Lisa",

"email" : "lisa@simpsons.com",

"telepon" : "555-111-1224"

},

{

"nama" : "Bart",

"email" : "bart@simpsons.com",

"telepon" : "555-222-1234"

},

{

"nama" : "Homer",

"email" : "homer@simpsons.com",

"telepon" : "555-222-1244"

},

{

"nama" : "Marge",

"email" : "marge@simpsons.com",

"telepon" : "555-222-1254"

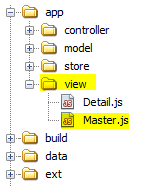
}]

}

Url *'data/User.json'* adalah sebuah penarikan data yang dalam contoh ini adalah *statis*. Untuk pengembangan dapat diubah menjadi data yang *dinamis* dengan menggabungkan logika program dan database seperti contoh menggunakan PHP dan MySql. Dengan melakukan *query database* sehingga data dapat ditampilkan secara *dinamis*.

1. Selanjutnya adalah membuat *class View* pada folder ***D:\www\mvcapp\app\view*** sebagai *interface* atau antarmuka kepada pengguna. Ada 2 *class view* yang dibuat yaitu ***Master.js*** dan ***Detail.js***.

Class pada file ***Master.js*** adalah sebuah *view* dengan komponen *grid* untuk menampilkan *list data* yang dikumpulkan oleh *class store* ***User.js***. Berikut ini adalah isi dari file ***Master.js***.

Ext.define('MVCApp.view.Master', {

extend: 'Ext.grid.Panel',

alias: 'widget.mvc-MasterView',

title: 'Master Panel',

store: 'User',

columns: [{

text: 'Nama',

dataIndex: 'nama',

flex: 1,

align: 'left'

}, {

text: 'Email',

dataIndex: 'email',

flex: 1,

align: 'left'

}, {

text: 'Telepon',

dataIndex: 'telepon',

flex: 1,

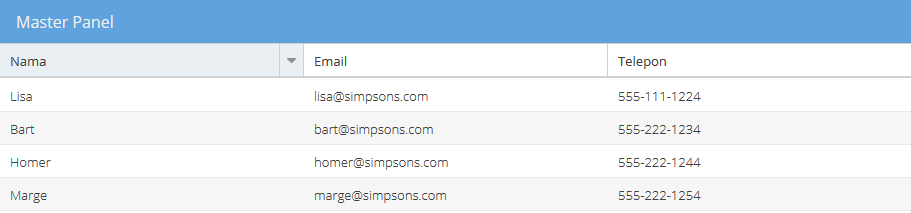
align: 'left'

}]

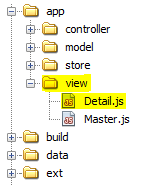
});

Parameter *store: 'User'* pada kode diatas adalah menyatakan bahawa komponen grid diatas mengambil sumber data dari *class store* *'MVCApp.store.User'*. Adapun penulisannya hanya *store: 'User'* karena pada class *'MVCApp.store.User'* terdapat parameter *storeId: 'User'*. Maka dapat ditulis *storeId* nya saja.

Parameter *columns* pada kode diatas adalah mengatur komponen *grid* agar menampilkan list data sesuai dengan hasil dari *class store*. Dengan parameter *dataIndex* yang berlabel hijau adalah agar data yang ditampilkan sesuai dengan *field* pada *class model* yaitu ***D:\www\mvcapp\app\model\User.js***. Sehingga tampilannya menjadi seperti gambar berikut ini.



Selanjutnya adalah class ***Detail.js*** yaitu class berupa komponen form yang akan menampilkan setiap baris data pada *class* ***Master.js*** ketika salah satu baris dipilih. Berikut ini adalah isi dari class ***Detail.js***.



Ext.define('MVCApp.view.Detail', {

extend: 'Ext.form.Panel',

alias: 'widget.mvc-DetailView',

title: 'Detail Panel',

frame: true,

padding: 10,

items: [{

xtype: 'textfield',

name: 'nama',

fieldLabel : 'Nama'

}, {

xtype: 'textfield',

name: 'email',

fieldLabel: 'Email'

}, {

xtype: 'textfield',

name: 'telepon',

fieldLabel: 'Telepon'

}, {

xtype: 'hiddenfield',

name: 'id'

}, {

xtype: 'button',

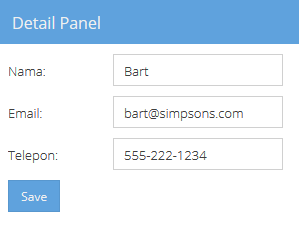
text: 'Save',

itemId : 'SaveRecord'

}]

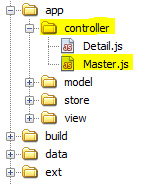
});

Pada parameter *items* yang berlabel hijau merupakan komponen anak dari komponen *form* diatas adalah komponen masukan berupa *textfield* dan *hiddenfield* dimana terdapat parameter *name* yang berlabel warna kuning dengan nilai yang sama dengan *field* pada *class model* yaitu***D:\www\mvcapp\app\model\User.js***. Agar data yang tampil sama dengan data yang dipilih pada *class* ***Master.js***. Berikut ini adalah tampilan dari ***Detail.js*** ketika salah satu baris pada *grid* dipilih.



1. Membuat class *Controller* pada folder ***D:\www\mvcapp\app\controller*** yaitu untuk mengatur suatu *event* atau *listeners* dari komponen yang ada pada *class* ***Master.js*** dan ***Detail.js*** di *class View*. Ada 2 class pada *Controller* yaitu *class* *Controller* ***Master.js*** yaitu mengatur *event* komponen yang ada pada *View* ***Master.js*** dan *class* *Controller* ***Detail.js*** untuk mengatur *event* komponen yang ada pada *View* ***Detail.js***.

Pada *Controller* ***Master.js*** komponen yang memiliki event adalah komponen grid yaitu saat baris dipilih. Dan berikut ini adalah isi dari *Controller* ***Master.js***.



Ext.define('MVCApp.controller.Master', {

extend : 'Ext.app.Controller',

init: function() {

this.control({

'mvc-MasterView': {

select: this.onGridSelect

}

});

},

onGridSelect: function(grid, rec, index, eOpts) {

var detailView =

Ext.ComponentQuery.query(

'mvc-DetailView')[0];

detailView.loadRecord(rec );

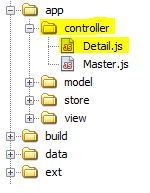
}

});

Pada kode *class* *Controller* ***Master.js*** di atas. Yang berlabel kuning yaitu *'mvc-MasterView'* adalah komponen grid yang akan dikasih sebuah *event* atau *listeners*. Kemudian yang berlabel hijau yaitu *'select'* adalah *event* atau *listeners* dari grid tersebut. Dan yang berlabel merah adalah sebuah fungsi *'onGridSelect'* yang ada dibawahnya untuk mengerjakan perintah dari *event* atau *listeners* *'select'* tersebut. Artinya ketika grid mengalami sebuah event *'select'* maka lakukan perintah yang ada pada fungsi *'onGridSelect'*.

Sementara perintah yang ada pada fungsi *'onGridSelect'* adalah perintah untuk menampilkan data yang dipilih kedalam sebuah komponen *form* yang ada pada *View* ***Detail.js***.

Setelah membuat *Controller* ***Master.js*** selanjutnya membuat *Controller* ***Detail.js***. Pada *Controller* ***Detail.js*** komponen yang memiliki event adalah komponen tombol *Save* saat diklik. Dan berikut ini adalah isi dari *Controller* ***Detail.js***.

Ext.define('MVCApp.controller.Detail', {

extend: 'Ext.app.Controller',

init: function() {

this.control({

'mvc-DetailView > button#SaveRecord' : {

click : this.onSaveButtonClick

}

});

},

onSaveButtonClick : function(btn) {

var detailView = btn.up('mvc-DetailView');

var masterView =

btn.up('#mvc-MainView').

down('mvc-MasterView');

var data = detailView.getValues();

var store = masterView.getStore();

var record = store.getById(data.id);

if (!record) {

Ext.Msg.alert('Kesalahan',

'Maaf, Tidak ada data yang dipilih.');

return;

}

detailView.updateRecord();

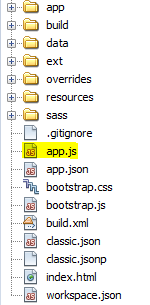
}

});

Pada kode *class* *Controller* ***Detail.js*** di atas. Yang berlabel kuning yaitu *'mvc-DetailView > button#SaveRecord'* adalah komponen *button* dengan '*itemId: SaveRecord'* yang berada pada komponen *form* *'mvc-DetailView'* melakukan *event* atau *listeners* berupa *click* yang berlabel hijau dengan melakukan perintah pada fungsi *'onSaveButtonClick'* yang berlabel merah.

Fungsi dari *'onSaveButtonClick'* dari kode diatas adalah melakukan perubahan data pada baris data yang dipilih. Tetapi jika data belum dipilih maka akan menampilkan sebuah pesan kesalahan.

1. Selanjutnya adalah merubah file ***app.js*** yang berada pada folder ***D:\www\mvcapp*** seperti kode dibawah ini.

Ext.application({

name: 'MVCApp',

models: [

'User'

],

views: [

'Master',

'Detail'

],

controllers: [

'Master',

'Detail'

],

stores: [

'User'

],

launch: function () {

Ext.create('Ext.container.Viewport', {

itemId: 'mvc-MainView',

layout: {

type: 'hbox',

align: 'stretch'

},

items: [{

xtype: 'mvc-MasterView',

flex: 2

}, {

xtype: 'mvc-DetailView',

flex: 1

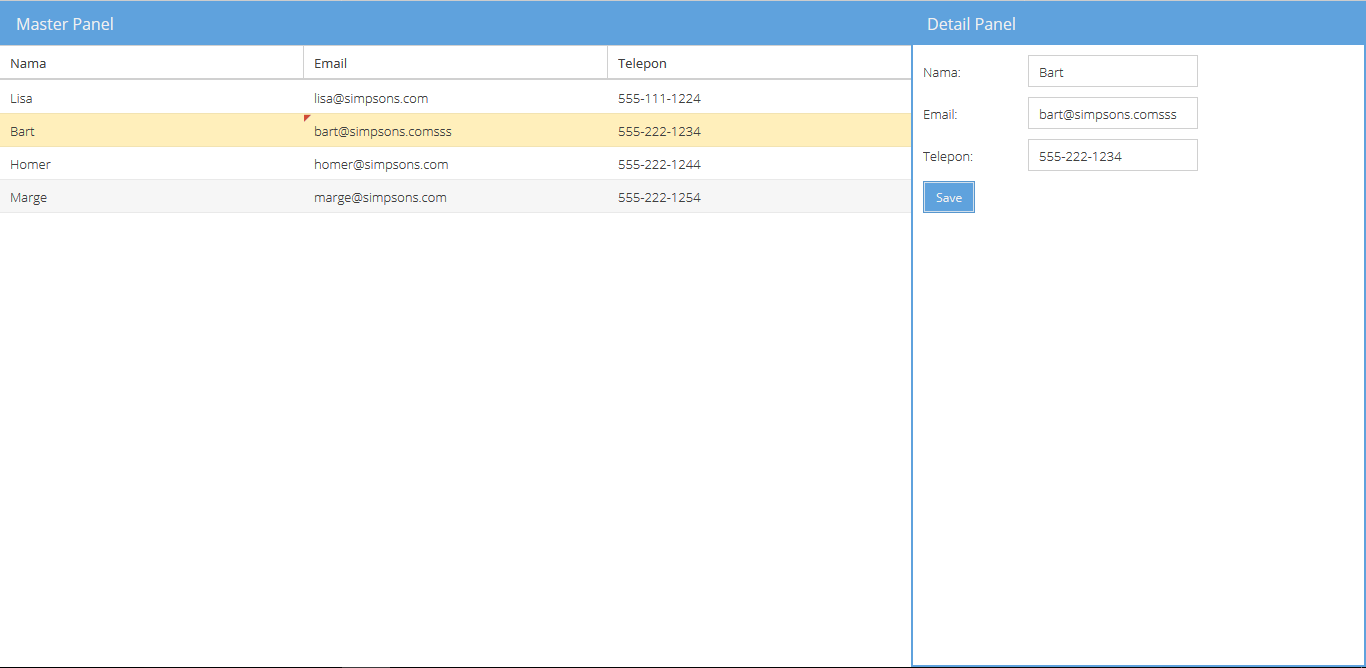
}]

});

}

}) ;

Pada kode diatas yang berlabel warna kuning yaitu parameter *models*, *views*, *controllers*, dan *stores* adalah menambahkan *class-class* yang ada di *Model*, *View*, *Controller*, dan *Store* kedalam aplikasi. Sementara fungsi *launch* berisi menciptakan sebuah komponen *Viewport* sebagai komponen utama untuk ditampilkan. Pemberian parameter *itemId: 'mvc-MainView'* adalah sebagai sebagai identitas komponen. Dan sebagai komponen anaknya adalah *Class View* ***Master.js*** dan *Class View* ***Detail.js*** yang disusun secara horizontal dalam 1 layar. Dan berikut ini adalah hasilnya.



Bab VI

Parameter Layout

Layout adalah suatu parameter untuk mengatur posisi komponen-komponen yang diletakan pada sebuah bidang. Dalam hal ini komponen bidang dapat berupa ***Form***, ***Panel***, ***Container***, ***Viewport***, dan beberapa komponen bidang lainnya. Mengenai komponen bidang akan dibahas pada bab selanjutnya.

Dibawah ini adalah gambar peletakan komponen pada bidang ***Viewport*** menggunakan layout border dengan masing-masing region ***west***, ***north***, ***east***, ***south***, dan ***center***.

A screenshot of a social media post

Description automatically generated

Untuk lebih jelas mengenai ***layout*** berikut ini adalah macam-macam bentuk layout yang dapat diterapkan pada sebuah bidang.

Macam-macam layout adalah sebagai berikut:

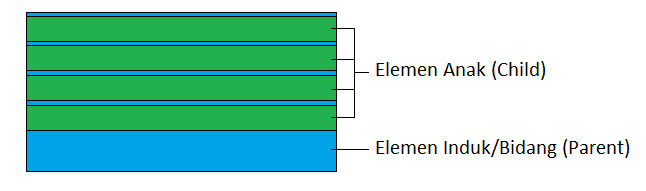
1. Anchor (Jangkar)
2. Vbox (Vertikal Kotak)
3. Hbox (Horizontal Kotak)
4. Border (Sekat)
5. Fit (Penuh)
6. Accordeon Vertikal (Seperti alat musik accordeon yang bergerak vertikal)
7. Accordeon Horizontal (Seperti alat musik accordeon yang bergerak Horizontal)

**VI.1. Layout Anchor**

Layout anchor atau yang disebut juga jangkar digunakan untuk meletakkan elemen pada elemen bidang induk dengan cara menarik penuh lebar elemen tersebut sesuai lebar elemen bidang induk.

Kelebarannya bisa di tentukan dengan nilai persen menggunakan parameter ***anchor***. Jika bernilai 100% maka leber elemen akan sama dengan lebar dari elemen bidang induk tersebut.

Berikut ini adalah ilustrasi gambar penggunaan layout Anchor.

[](http://kiezie.web.id/wp-content/uploads/2015/11/layout_anchor.png)

Biasanya layout ini digunakan untuk membuat sebuah ***submit form*** atau ***form entri***. Berikut ini adalah contoh penulisan kode menggunakan layout ***anchor***.

Ext.create(MyApp.view.utama', {

extend: 'Ext.form.Panel',

alias: 'widget.formlayoutanchor',

layout: 'anchor',

items: [{

xtype: 'textfield',

fieldLabel: 'Id User',

name: 'username',

anchor: '100%'

}, {

xtype: 'textfield',

fieldLabel: 'Password',

inputType: 'password',

name: 'password',

anchor: '50%'

}],

buttons: [{

text: 'Login',

handler: proseslogin

}]

});