INFOR Group

一、動機

林澤宇:自從我暑假學會寫Android之後，就一直很想為INFOR設計一個專屬的應用程式，原本是想讓他功能可以取代掉FB社團，但礙於現實因素，所以有些功能難以在時間內實現，可能考完學測再回來寫吧

陳威廷:我覺得我還需要吃更多的優格才會變瘦，而且一定要是有糖的

陳威廷:我覺得我還需要吃更多的優格才會變瘦，而且一定要是有糖的

陳威廷:我覺得我還需要吃更多的優格才會變瘦，而且一定要是有糖的

陳威廷:我覺得我還需要吃更多的優格才會變瘦，而且一定要是有糖的

陳威廷:我覺得我還需要吃更多的優格才會變瘦，而且一定要是有糖的

李秉諺:我們這組人真多

二、成品介紹

Infor Group是一個可以線上提問，並且可以很快獲得回答的程式，同時也是資訊社學長學弟交流感情的好地方，為了方便，我們同時設計了網頁版與手機版(Android only)

三、使用技術

App:Java

前端：vue.js

後端：Node.js MongoDB socket.io

四、技術介紹

**Android: Java**

本次使用的語言是Java，基本上是由XML檔、Java檔兩大部分組成一個標準的Android程式

**以下為一個Hello World範例:**

**XML檔:**

|  |
| --- |
| <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:orientation="vertical"  >  <TextView  android:id="@+id/textview"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  />  </LinearLayout> |

**Java檔:**

|  |
| --- |
| public class MainActivity extends AppCompativity {    @Override  protected void onCreate(Bundle saveInstanceState){  super.onCreate(saveInstanceState);  setContentView(R.layout.activity\_main);  TextView text = (TextView) findViewById(R.id.textview);  text.setText("JIZZ World!");  }  } |

接下來講解登入畫面，直接採用Android Studio內建的預設登入畫面，簡單說就只是跟後端確認輸入資料是否正確，符合的話即可登入

登入後的頁面是程式的主畫面，也是由Android Studio預設的Navigation Drawer Activity實作出來的，主要由Navigation + Drawer 組成的

整個頁面叫Drawer，而滑出來的目錄部分叫做Navigation，順帶一提目錄上面綠色的部分叫Header，我們來看一下這個頁面的XML:

|  |
| --- |
| <android.support.v4.widget.DrawerLayout  xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto"  xmlns:tools="http://schemas.android.com/tools"  android:id="@+id/drawer\_layout"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:fitsSystemWindows="true"  tools:openDrawer="start">  <include  layout="@layout/app\_bar\_menu"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent" />  <android.support.design.widget.NavigationView  android:id="@+id/nav\_view"  android:layout\_width="wrap\_content"  android:layout\_height="match\_parent"  android:layout\_gravity="start"  android:fitsSystemWindows="true"  app:headerLayout="@layout/nav\_header\_menu"  app:menu="@menu/activity\_menu\_drawer" />  </android.support.v4.widget.DrawerLayout> |

include裝的東西就是原本主畫面的Content，使用include標籤可以之前寫過的Layout裝進去並顯示出來

當想要切換到不同頁面的時候，主要的Content就會刷新成不同的頁面，這裡我使用的是Fragment來切換到不同的頁面，可以把Fragment想像成把Activity裡面的各個部分拆開成很多小部分，每個部分都是一個Fragment，我們來看一下程式碼:

|  |
| --- |
| <LinearLayout  xmlns:android="http://schemas.android.com/apk/res/android"  android:id="@+id/content\_menu"`  xmlns:app="http://schemas.android.com/apk/res-auto"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:orientation="vertical"  app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior"  tools:showIn="@layout/app\_bar\_menu">  <RelativeLayout  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:orientation="vertical"  >  <LinearLayout  android:id="@+id/search\_bar"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:orientation="horizontal">  <EditText  android:id="@+id/search\_question\_box"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:hint="Search Questions"/>  </LinearLayout>  <ListView  android:id="@+id/question\_list"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_below="@id/search\_bar"/>  </RelativeLayout>  </LinearLayout> |
| public class FragmentActivityQuestion extends Fragment {  @Override  public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {  View view = inflater.inflate(R.layout.fragment\_question,container,false);  return view;  }  } |

主要Activity寫法:

|  |
| --- |
| @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_menu);  Fragment f = new FragmentActivityQuestion();  getFragmentManager().beginTransaction().replace(R.id.content\_menu,f).commit();  }  . . .  . . .  @Override  public boolean onNavigationItemSelected(MenuItem item) {  // Handle navigation view item clicks here.  int id = item.getItemId();  switch(id){  case R.id.nav\_question:  fab.show();  Fragment fQuestion = new FragmentActivityQuestion();  getFragmentManager().beginTransaction().replace(R.id.content\_menu,fQuestion).commit();  if(Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.LOLLIPOP) {  fab.setImageDrawable(getResources().getDrawable(R.drawable.ic\_add\_black\_24dp, MenuActivity.this.getTheme()));  } else {  fab.setImageDrawable(getResources().getDrawable(R.drawable.ic\_add\_black\_24dp));  }  break;  }  DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer\_layout);  drawer.closeDrawer(GravityCompat.START);  return true;  } |

如此一來就可以順暢的切換到不同頁面了，接著我們切換到主頁面:

可以注意到右下角有一個Floating Action Button(以下簡稱Fab)，這是採用Google在2015 Google I/O大會上介紹的Android Design Support Library裡面的CoordinatorLayout，簡單說就是為了Fab而生的Layout

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <android.support.design.widget.CoordinatorLayout  xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">  <android.support.design.widget.FloatingActionButton  android:id="@+id/fab"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_gravity="bottom|end"  android:layout\_margin="@dimen/fab\_margin"  app:srcCompat="@drawable/ic\_add\_black\_24dp" />  </android.support.design.widget.CoordinatorLayout> |

我們重新看一次上面的那個介面，可以發現有3個問題，而列出那些問題的正是ListView + Adapter，Adapter可以把他想像成一個跟Data和View溝通的橋樑，以下為使用範例:

首先，在ListView裡的每一個Item都有一個固定的View，也就是Item的樣式有一個他自己的XML，我們可以寫出問題的XML檔形如:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <LinearLayout  xmlns:android="http://schemas.android.com/apk/res/android"  android:id="@+id/lcontainer"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:orientation="horizontal">  <ImageView  android:id="@+id/question\_asker"  android:layout\_width="60dp"  android:layout\_height="60dp"  android:adjustViewBounds="true"/>  <LinearLayout  android:layout\_width="wrap\_content"  android:layout\_height="match\_parent"  android:orientation="vertical">  <TextView  android:id="@+id/question\_title"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_weight="1"  android:textSize="20sp"/>  <TextView  android:id="@+id/question\_tags"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_weight="1"  android:textSize="15sp"  />  </LinearLayout>  </LinearLayout> |

再來我們要創建每個Item的Class，跟XML一樣創一個共同格式即可

|  |
| --- |
| public class Question{  public int asker;  public String qtitle;  public String qtag;  public Question(int asker,String qtitle,String qtag){  super();  this.asker = asker;  this.qtitle = qtitle;  this.qtag = qtag;  }  public int getAsker(){  return asker;  }  public String getQtitle(){  return qtitle;  }  public String getQtag(){  return qtag;  }  public void setAsker(int asker){  this.asker = asker;  }  public void setQtitle(String qtitle){  this.qtitle = qtitle;  }  public void setQtag(String qtag){  this.qtag = qtag;  }  } |

再來就是要寫Adapter了，由於我們需要同時實作搜尋功能，所以需要使用一個類別叫Filter，以下為程式碼:

|  |
| --- |
| public class Madapter extends BaseAdapter implements Filterable{  private ArrayList<Question> mOriginValue;  private ArrayList<Question> mDisplayValue;  LayoutInflater inflater;  private class ViewHolder{  LinearLayout llContainer;  ImageView asker;  TextView qtitle,qtag;  }  public Madapter(Context context,ArrayList<Question> mQuestion){  this.mOriginValue = mQuestion;  this.mDisplayValue = mQuestion;  inflater = LayoutInflater.from(context);  }  @Override  public int getCount(){  return mDisplayValue.size();  }  @Override  public Object getItem(int pos){  return pos;  }  @Override  public long getItemId(int pos){  return pos;  }  @Override  public View getView(final int pos,View convertView ,ViewGroup parent){  ViewHolder holder = null;  if(convertView == null){  holder = new ViewHolder();  convertView = inflater.inflate(R.layout.questions\_list\_item,null);  holder.llContainer = (LinearLayout) convertView.findViewById(R.id.lcontainer);  holder.asker = (ImageView) convertView.findViewById(R.id.question\_asker);  holder.qtitle = (TextView) convertView.findViewById(R.id.question\_title);  holder.qtag = (TextView) convertView.findViewById(R.id.question\_tags);  convertView.setTag(holder);  } else {  holder = (ViewHolder) convertView.getTag();  }  holder.asker.setBackgroundResource(mDisplayValue.get(pos).asker);  holder.qtitle.setText(mDisplayValue.get(pos).qtitle);  holder.qtag.setText(mDisplayValue.get(pos).qtag);  holder.llContainer.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View view) {  Toast.makeText(getActivity(),mDisplayValue.get(pos).qtitle,Toast.LENGTH\_SHORT).show();  }  });  return convertView;  }  @Override  public Filter getFilter(){  Filter filter = new Filter() {  @SuppressWarnings("unchecked")  @Override  protected void publishResults(CharSequence constraint, FilterResults results){  mDisplayValue = (ArrayList<Question>) results.values;  notifyDataSetChanged();  }  @Override  protected FilterResults performFiltering(CharSequence constraint){  FilterResults results = new FilterResults();  ArrayList<Question> FilterArrList = new ArrayList<Question>();  if(mOriginValue == null){  mOriginValue = new ArrayList<Question>(mDisplayValue);  }  if(constraint == null || constraint.length() == 0){  results.count = mOriginValue.size();  results.values = mOriginValue;  } else {  constraint = constraint.toString().toLowerCase();  for(int i = 0;i < mOriginValue.size();++i){  String data = mOriginValue.get(i).qtitle;  if(data.toLowerCase().contains(constraint.toString())){  FilterArrList.add(new Question(mOriginValue.get(i).asker  ,mOriginValue.get(i).qtitle,mOriginValue.get(i).qtag));  }  results.count = FilterArrList.size();  results.values = FilterArrList;  }  }  return results;  }  };  return filter;  }  } |

最後在Adapter中輸入資料，即可完成搜尋功能

連接後端的方式我們採用一個套件叫做okhttp，使用時須在Gradle加入:

|  |
| --- |
| implementation 'com.squareup.okhttp3:okhttp:3.9.1' |

他可以透過URL來拿到網頁上的資料，也就是GET，以下為一簡單的範例:

|  |
| --- |
| private OkHttpClient client = new OkHttpClient();  private void HandleRequest(){  client.newCall(request).enqueue(new Callback() {  @Override  public void onFailure(Call call,final IOException e) {  runOnUiThread(new Runnable() {  @Override  public void run() {  getText.setText(e.getMessage());  }  });  }  @Override  public void onResponse(Call call, Response response) throws IOException {  final String resStr = response.body().string();  runOnUiThread(new Runnable() {  @Override  public void run() {  getText.setText(resStr);  }  });  }  });  }  } |

這樣子就可以取得那個URL的Html，再從取下來的Html中拿取需要的資訊即可。注意到，我們使用了runOnUiThread，如果我們不加上他的話，會造成錯誤，是因為我們在執行GET指令時，我們在MainThread外還開了一個WorkThread，而我們是無法在MainThread以外的Thread更改TextView的內容的，所以需要加上runOnUiThread，讓我們回歸到MainThread中去更改TextView

POST因為需要傳送東西，所以會稍微更複雜一些，以下為一個POST範例:

|  |
| --- |
| private void handlePost(){  RequestBody formBody = new FormBody.Builder()  .add("username", "Pooh")  .add("password","jizz")  .build();  final Request request = new Request.Builder()  .url("Your URL")  .post(formBody)  .build();  client.newCall(request).enqueue(new Callback() {  @Override  public void onFailure(Call call,final IOException e) {  runOnUiThread(new Runnable() {  @Override  public void run() {  getText.setText(e.getMessage());  }  });  }  @Override  public void onResponse(Call call, Response response) throws IOException {  runOnUiThread(new Runnable() {  @Override  public void run() {  getText.setText("Register Successfully");  }  });  }  });  } |

只是多了一個FormBody儲存我們要傳送的資料，FormBody相對於Html中的Form標籤，如此一來即可順利的傳送資料

前端-Vue.js



簡介:

Vue(讀音近似view)，是一套用來建構用戶介面的progressive framework，與其他大型框架不同的是，Vue被設計可以自底向上逐層應用.而Vue主要專注在視圖層的開發,也就是所謂的view層。Vue不只容易上手,也可以很方便地與現成的libraries或專案進行結合。

特色:

1.Vue.js使用Virtual DOM

2.提供了Reaction和Composible的view components

3.集中在核心庫的應用，將其他功能像routing或全局管理交由其他相關的libraries

而和其他的模板相比Vue的模板更加貼近Html，所以使得Html開發者可以更加容易地將應用逐步地遷移至Vue。

最後Vue提供了Vue-cli腳手架，能讓開發者更輕易的建構項目，而Vue-cli包含了 Webpack，Browserify，甚至 no build system。

前端範例:

對於前端,我們主要想要達成的功能是動態更新的留言，而Vue.js的及時渲染功能可以讓我們很好的達成這一點

基礎語法:

|  |
| --- |
| <div id="ee">  {{content}}  </div> |
| new Vue({  el:"#ee",  data:{  content:"jizz"  }  }) |

在html中，要連接Vue.js的方法，是在標籤中加一行id，而Vue就可以把資料傳進標籤中

在script中，可以新增一個Vue的物件，而要連動到html的標籤中需要使用el:，而傳輸的資料則是放在data:中。

簡單的component，props介紹:

在Vue.js中，我們可以自己建立自己的conponent，這可以讓我們更方便的建構留言的模板:

|  |
| --- |
| Vue.component('post', {  props: ['title'],  template: '<h3>{{ title }}</h3>'  }) |

這樣你們就可以在html中使用名為<blog-post>的template，而底下的props是可以一個可以自訂義的屬性，當一個值傳遞給一個props時，他們就會變成conponent中的一個屬性，而在上述的template中你會發現你可以在coponent中實際使用props傳出來的值，像是:

|  |
| --- |
| <post title="My journey with Vue"></post>  <post title="Blogging with Vue"></post>  <post title="Why Vue is so fun"></post> |

v-on and v-for:

在動態更新中的核心，無疑是v-on和v-for了，v-on的作用是可以監聽DOM，實例如下:

|  |
| --- |
| <div id="bc">  <bc></bc>  </div> |
| Vue.component('bc',{  data:function(){  return{  count:0  }  },  template: '<button v-on:click="count++"你點擊郭 {{ count }} 次。</button>'  })  new Vue({ el:"#bc" }) |

在上述例子中，我們自訂了一個conponent bc，而在bc的template中我們使用v-on來監聽按鈕按下後的事件，在按下按鈕後我們會使變數count++並把他印出來。

而v-for則是可以把Vue中data自訂的陣列在網頁上印出，這樣就可以實現動態渲染的效果:

|  |
| --- |
| <div id="app">  <ul>  <li v-for="(item, index) in list">  index: {{ index }},  name: {{ item.name }}  </li>  </ul>  </div> |
| new Vue({  el: '#app',  data: {  list: [  { id: '712223456789', name: 'jizz1' },  { id: '234567122890', name: 'jizz2' },  { id: '345678907122', name: 'jizz3' }  ]  }  }); |

其中的item是使用者可以自行設定的名稱，index的功用是可以知道這是陣列中第幾個渲染出的物件以方便刪除。

v-bind and v-model:

當開發者想要根據使用者的選擇來更改class或是style，這時候就要使用v-bind來進行綁定，以變更class作為實例:

|  |
| --- |
| <div v-bind:class="{a:isActive,b:!isActive}" id="jj">  <p>jizz</p>  <button v-on:click="changetrue">change to a</button>  <button v-on:click="changefalse">change to b</button>  </div> |
| new Vue({  el:"#jj",  data: {  isActive:true  },  methods: {  changetrue:function(){  this.isActive=true  alert(this.isActive)  //return  },  changefalse:function(){  this.isActive=false  alert(this.isActive)  //return  }  }  }) |

我們可以看到當我們按下按鈕之後，就可以根據自訂功能來變更isActive的值，而當isActive被改動後，由於v-bind的綁定，html中的class也就可以即時更新了。

而與v-bind不同的是，v-model其實是雙向的綁定，即是說當v-model被放到input中時，不只可以用v-on監聽button來改動，也可以直接輸入input中來改動，實例如下:

|  |
| --- |
| <div id="bv">  <input v-model="say">  <br>I want to say: {{ say }}  </div> |
| new Vue({  el: '#bv',  data: {  say: '7122',  }  }); |

這樣就可以即時的在網頁印出input中的值了。

todo list:

當上面的東西都初步了解後，就可以試試看寫一個todo list了:

|  |
| --- |
| <div id="todo-list-example">  <input  v-model="newTodoText"  v-on:keyup.enter="addNewTodo"  placeholder="Add a todo"  >  <ul>  <li  is="todo-item"  v-for="(todo, index) in todos"  v-bind:key="todo.id"  v-bind:title="todo.title"  v-on:remove="todos.splice(index, 1)"  ></li>  </ul>  </div> |
| Vue.component('todo-item', {  template: '\  <li>\  {{ title }}\  <button v-on:click="$emit(\'remove\')">X</button>\  </li>\  ',  props: ['title']  })  new Vue({  el: '#todo-list-example',  data: {  newTodoText: '',  todos: [  {  id: 1,  title: 'Do the dishes',  },  {  id: 2,  title: 'Take out the trash',  },  {  id: 3,  title: 'Mow the lawn'  }  ],  nextTodoId: 4  },  methods: {  addNewTodo: function () {  this.todos.push({  id: this.nextTodoId++,  title: this.newTodoText  })  this.newTodoText = ''  }  }  }) |

這裡擷取了官方網站的範例，而做出todo list後，就可以拓展功能來做出留言的模板。

基礎留言模板:

以下是我做出的基礎留言模板:

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <meta http-equiv="X-UA-Compatible" content="ie=edge">  <title>留言板模板</title>  <link rel="stylesheet" href="http://cdn.bootcss.com/bootstrap/3.3.0/css/bootstrap.min.css">  </head>  <body>  <header>  <nav class="navbar navbar-default" role="navigation">  <div class="container-fluid">  <h1 class="text-center"><a href="#" class="">留言板模板</a></h1>  </div>  </nav>  </header>  <div id="cc">  <div class="container" style="margin-bottom: 20px;">  <input autofocus v-model="newcontent" class="form-control" placeholder="请输入留言" @keyup.enter="addcontent">  <br>  <button @click="addcontent" class="btn btn-primary btn-block">提交</button>  </div>  <div class="container">  <div class="panel panel-default">  <div class="panel-heading">留言列表</div>  <div class="panel-body">  <ul class="list-group">  <li class="list-group-item" v-for="(content,index) in contents">{{ content.title }} <button class="btn btn-primary btn-sm" data-toggle="modal" data-target="#layer" @click="removecontent(index)" >删除</button>  </ul>  <button @click="removecontents" v-show="contents.length > 0" class="btn btn-danger btn-block">清空留言</button>  </div>  </div>  </div>  </div>  <script src="https://unpkg.com/vue@2.2.6/dist/vue.js"></script>  <script src='./vue.js'></script>  </body>  </html> |
| var app = new Vue({  el:'#cc',  data:{  contents:[],  newcontent:"",  newid: 0  },  /\*watch:{  contents:{  handler:function(contents){  contentsg.save(contents)  }//,  //deep:true  }  },\*/  methods:{  addcontent:function(){  var value = this.newcontent && this.newcontent.trim()  if(!value){  return  }  this.contents.push({  id: this.newid++,  title: value  })  this.newcontent=''  },  removecontents:function(){    return this.contents=[]  },  removecontent:function(index){  //alert(index)  return this.contents.splice(index,1);  }  }  }) |



後端

## Node.js & Express.js

Node.js提供了一個擴充性極佳的應用程式平台，相當具有發展潛力。又因為Web的盛行，以及Javascript 相關技術的蓬勃發展，讓Node.js對多數人來說非常好上手，更可以應用於各種領域和平台。此外，Node.js由於有腳本語言的特性，可以用來快速開發應用，大幅度縮短開發時間。

Express.js是目前最為知名也最多人使用的Node.js網站框架，其Middleware的設計，為它帶來許多擴充能力，無論大網站還是小網站，都適合它來開發。

以下為使用Node.js與使用Express.js後的比較：

|  |
| --- |
| //Node.js  const http = require('http');  const hostname = '127.0.0.1';  const port = 3000;  const server = http.createServer((req, res) => {  res.statusCode = 200;  res.setHeader('Content-Type', 'text/plain');  res.end('Hello World\n');  });  server.listen(port, hostname, () => {  console.log(`Server running at http://${hostname}:${port}/`);  }); |

|  |
| --- |
| //Express.js  var express = require('express');  var app = express();  app.get('/', function (req, res) {  res.send('Hello World!');  });  app.listen(3000, function () {  console.log('Example app listening on port 3000!');  }); |

由舉例可看出Express.js包裝了大多數繁複的HTTP低階操作，使得我們的程式碼更為精簡。

## MongoDB & Mongoose

對於Node.js而言，MongoDB可說是相當友善的資料庫系統，以Javascript當作腳本語言而且使用JSON儲存資料，這意味著，Node.js可以直接使用JSON與MongoDB交換資料，不用再做大量轉換資料的工作，能省下不少時間。

與傳統的關聯式資料庫不一樣的是，MongoDB中的每筆資料，都是一組組的JSON，可以儲存擁有多層結構的資料。

|  |
| --- |
| {  "name":"INFOR",  "contact":{  "tel":"0971227122",  "email":"7122@jizz.com"  }  } |

Mongoose是目前在Node.js領域裡主流的MongoDB模組，幾乎所有的Node.js專案都使用它。Mongoose提供了一個簡單的方式，讓開發者可以使用操作Javascrip物件的形式，去存取MongoDB的資料。

|  |
| --- |
| var Schema = mongoose.Schema;  //用schema定義collection的結構  var userSchema = new Schema({  username: String,  passwd: String  });  //用定義好的schema產生一個modle  mongoose.model( 'User' , userSchema);  //用產生的model對collection進行操作  var Users = mongoose.model('User'); |

五、參考資料

<https://blog.csdn.net/qq_23018915/article/details/50831527>

<https://developer.android.com/reference/android/widget/Filter>

<https://developer.android.com/training/implementing-navigation/nav-drawer>

<http://givemepass.blogspot.tw/2011/10/baseadapterlistview.html>

<https://www.jianshu.com/p/4a77ae4cd82f>

<https://blog.csdn.net/xgjianstart/article/details/70054367>

<https://vuejs.org>

<https://www.tenlong.com.tw/products/9789572244753>

<https://nodejs.org/dist/latest-v8.x/docs/api/synopsis.html#synopsis_example>

<http://expressjs.com/zh-tw/starter/hello-world.html>

<http://blog.chh.tw/posts/mongodb-odm-mongoose/>