Search Scholar, a website of quick scholar data access

Song Yiwen, Wang Weizhe, Chen Haoping, Long Jiaxuan *

June 23, 2018

Contents

*Song Yiwen, 517030910380, F1703015, SJTU
Wang Weizhe, 517030910381, F1703015, SJTU
Chen Haoping, 517030910369, F1703015, SJTU
Long Jiaxuan, 517030910378, F1703015, SJTU

1 Introduction

Our website is made up of several parts. First, the general data access part, which enables the user to search for data included in the database according to their wish. Second, the data visualization part, which help visualize the data and give a concrete demonstration of some abstract or complex data, such as the mentor-student relationship between teachers and students, the cooperators' relationship of an author, and the published paper number of a academic conference. Visualized data gives advantage to show a trend and a relationship.

1.1 Contributors

The section of Search Index Page and animations is completed by Wang Weizhe. The sections of Detailed Conference Page, Front-end Development and Paper Recommendation are completed by Song Yiwen. The section of Data Visualization is completed by Long Jiaxuan, and finally, the section of MVC part is completed by Chen Haoping.

2 Search Index Page

In our final work, I am in charge of index.php and most of its backend such as authorResultBackend.php, paperResultBackend.php and conferenceResultBackend.php. As the main part of these phps had been finished before, what

we need to do is to put them into a css framework firstly.Then we dealed with the detailed part: quick jump to any page, animation effects when turning pages and waiting. Therefore, we will introduce how to achieve them in detail. For convenience, we may just take author as an example.

2.1 Our improvement of origin css

In the origin css, when we click the paper button, the author would stay bright, though the content has changed to corresponding result for papers. This phenomenon actually will confuse users whether they have successfully get what they want and it is like Figure 1.

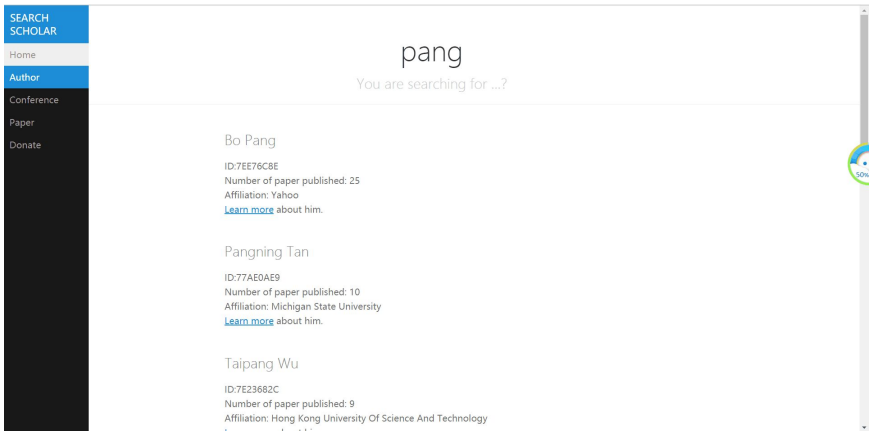


Figure 1: Click animations

After you click papers button, the content changes while the paper button is dim.

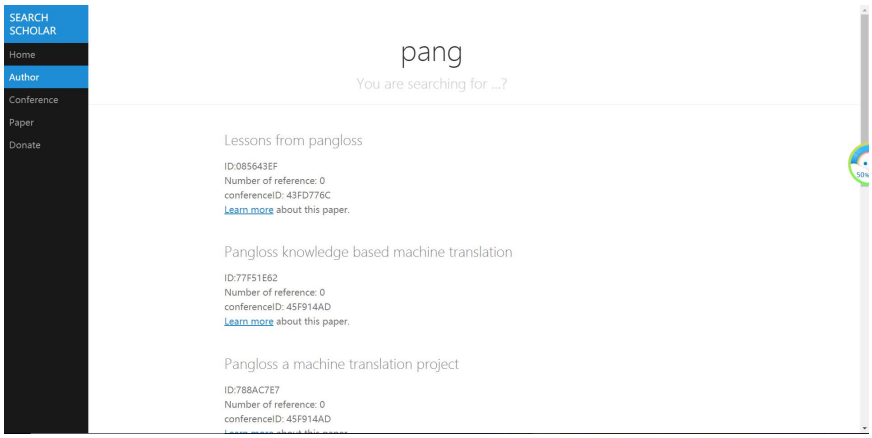


Figure 2: Highlight the chosen button

The reason of the problem is that author button is default button in the origin css.To solve the problem, we use addclass() function and removeclass() function, and the code is like:

```
var choice = 2;
window.onload=function(){
  $("#homeButton").click(function(){
    $("li").removeClass();
    $("#homeButton").addClass("pure-menu-item menu-item-
        divided pure-menu-selected");
    $("#authorButton").addClass("pure-menu-item");
    $("#conferenceButton").addClass("pure-menu-item");
    $("#paperButton").addClass("pure-menu-item");
    $("#donate").addClass("pure-menu-item");
    choice = 1;
    loadDataByPageIndex(1);
  })

  $("#authorButton").click(function(){
    $("li").removeClass();
    $("#homeButton").addClass("pure-menu-item");
    $("#authorButton").addClass("pure-menu-item menu-item-
        divided pure-menu-selected");
    $("#conferenceButton").addClass("pure-menu-item");
    $("#paperButton").addClass("pure-menu-item");
    $("#donate").addClass("pure-menu-item");
    choice = 2;
    loadDataByPageIndex(1);
  });
  $("#conferenceButton").click(function(){
    $("li").removeClass();
    $("#homeButton").addClass("pure-menu-item");
    $("#authorButton").addClass("pure-menu-item");
    $("#conferenceButton").addClass("pure-menu-item menu-item
        -divided pure-menu-selected");
    $("#paperButton").addClass("pure-menu-item");
    $("#donate").addClass("pure-menu-item");
```

```
      choice  = 3;
      loadDataByPageIndex(1);
    });
    $("#paperButton").click(function(){
      $("li").removeClass();
      $("#homeButton").addClass("pure-menu-item");
      $("#authorButton").addClass("pure-menu-item");
      $("#conferenceButton").addClass("pure-menu-item");
      $("#paperButton").addClass("pure-menu-item menu-item-
          divided pure-menu-selected");
      $("#donate").addClass("pure-menu-item");
      choice = 4;
      loadDataByPageIndex(1);
    })
  }
```

and it works.

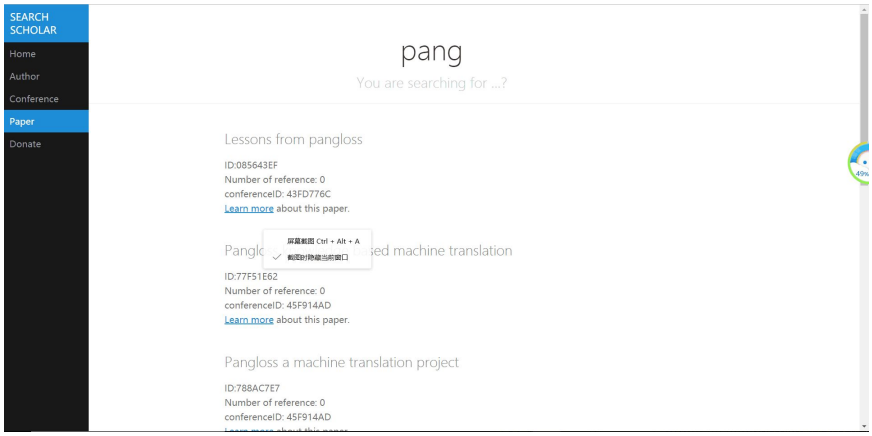


Figure 3: Realized Button Hightlight

2.2 animation effects when turning pages

In our former work, our page turning is prim and abrupt because there was no animation when tuning pages. If we don't have such a function, it may shock the users since the content displays and appears suddenly. To make it look more natural, we are going to add the animation. In this part, the main function we used is a jQuery function called slideToggle(). and the example code is like:

```
function loadDataByPageIndex(pageIndex) {
  $(".list").slideToggle(800);//key one
  start_time=Date.now();
  $.get("authorResultBackend.php", {
    pageIndex: pageIndex,
    scholarName: keyWord
  }, function (data) {
    $(".list").html(data);
    end_time=Date.now();
```

```
10          console.log("Load was performed.", "Time: ", end_time -
                start_time);
11          $(".list").slideToggle(1000);//key one
12        });
13    }
```

In fact, the key is to control when the animation should show and how to let it appears just after data's being done. We used to seperate slideToggle() from loadDataByPageoindex fuction (a function that can load data by pageindex which had been introduced in Lab 2). But it came to a problem that we can not make slideToggle() work just after finishing preparing data, which made it a out-of-step effect and a bit strange. In order to solve it, we just put it closely behind the $.get(). And there are the effect pictures like Figure 4 and Figure 5.

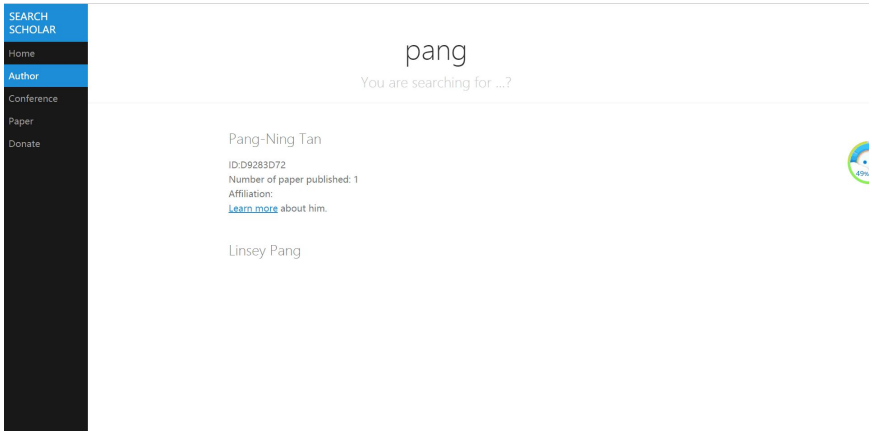


Figure 4: Content Slide

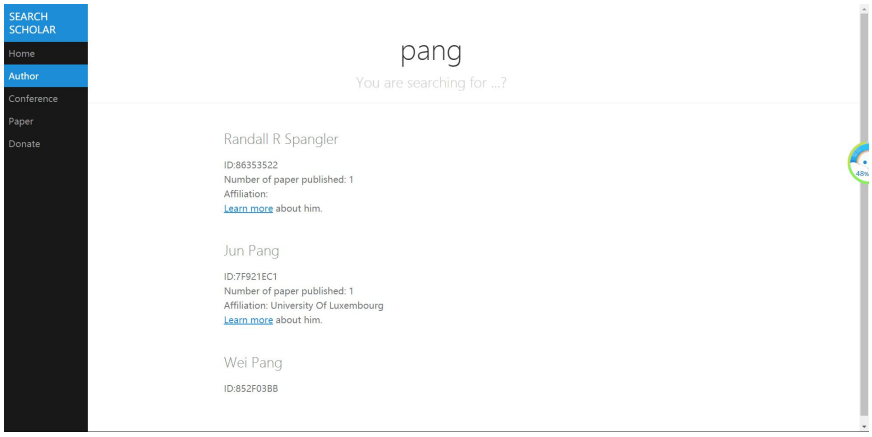


Figure 5: Content Slide

2.3 Loading animation

In our webpage, one thing is unavoidable. That is loading data. However, the time taken by it depends on various vectors. In order to tell the users that the web is working instead of terminating, we need to show them something when loading data. Obviously, a interesting animation should fit our need perfectly. As we decided to do it, there were also two basical problems——How and when to display it. To display a animation and hide it when we need, we use two jQuery function hide() and show() whose function is just as its name. After that, we choose to put it in the loadDataByPageoindex function to control when to display accurately.Here comes the sample code:

```
function loadDataByPageIndex(pageIndex) {
  $(".list").slideToggle(800,function(){$(".loading").show()
      ;});//key one
  start_time=Date.now();
  $.get("authorResultBackend.php", {
    pageIndex: pageIndex,
    scholarName: keyWord
  }, function (data) {
    $(".list").html(data);
    end_time=Date.now();
    console.log("Load was performed.", "Time: ", end_time -
        start_time);
    $(".loading").hide();//key one
    $(".list").slideToggle(1000);
  });
}
```

and the html part is like:

```
<div class="loading" style="text-align:center; margin-top:70
    px"><img src="loading.gif"></div>
<div class="container">
<div class="list">
</div><br>
</div>
```

In order to show its effect, I will show you two pictures: Before adding animation, the web-page is like Figure 6.

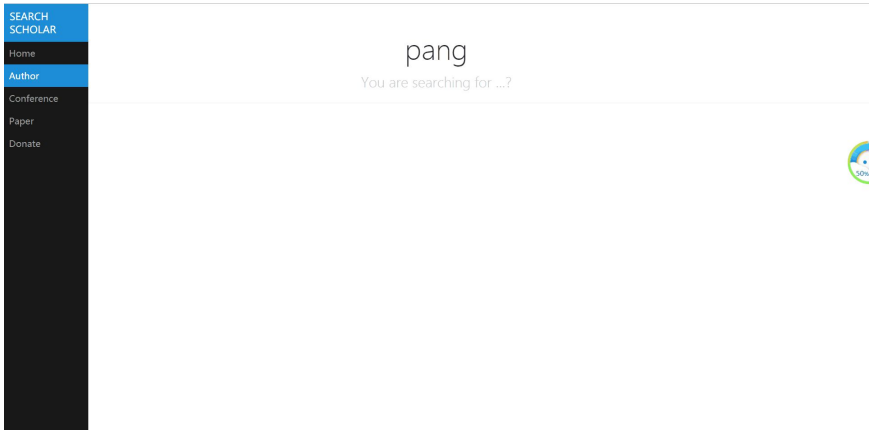


Figure 6: Before animation

After adding animation, the web-page changes into Figure 7.

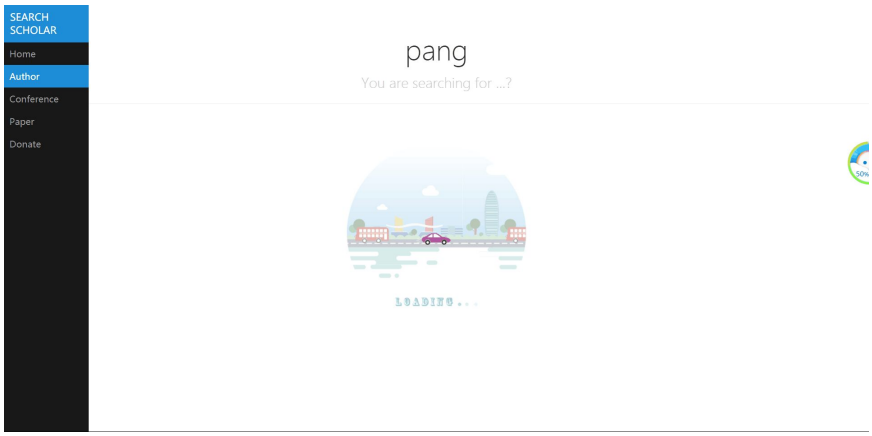


Figure 7: After animation

2.4 Quick Jump

In Lab 4, we are required to add “Next” and “Previous” buttons so that we can turn page. In fact, only these two buttons can not satisfy users at all. For example, if a user want to turn to Page 100, he has to click the “Next” button 100 times! It seems quite bored and ridiculous, doesn’t it? For convenience, we add a function called ”Quick Jump” and then users can turn to any page they want by just clicking 1 times. As we had a function called loadDataByPageoindex which can load data by pageindex, the only thing we need to do is to add a way to get the pageindex and convey it to the function loadDataByPageoindex. Hence, a select box should be a best choice. Here is the code: php part:

```
$pageIndex = $_GET["pageIndex"];
$startPoint = ($pageIndex-1)* 10;
......
$total = count($authorIds);
......
```

html part:

```
<center><select id="select">
<?php for($i=1;$i<=($total / 10)+1;$i++){
  if ($i == $pageIndex)//To tell user the page they are just at.
  echo "<option value=".$i." selected>Page ".$i."</option>";
  else
  echo "<option value=".$i.">Page ".$i."</option>";}?>
</select>
<input id="page" type="button" value="Turn to" onclick="page()"/></center>
```

JS part:

```
function loadDataByPageIndex(pageIndex) {
  $(".list").slideToggle(800,function(){$(".loading").show();});
  start_time=Date.now();
  $.get("authorResultBackend.php", {
    pageIndex: pageIndex,
    scholarName: keyWord
  }, function (data) {
    $(".list").html(data);
    end_time=Date.now();
    console.log("Load was performed.", "Time: ", end_time - start_time);
    $(".loading").hide();
    $(".list").slideToggle(1000);
  });
}
.......
function page()
{
  var da=document.getElementById("select").value;
  pageIndex = da;
  loadDataByPageIndex(da);
  console.log("NEXT: ", da);
}
```

Then there are the pictures. Original one is like Figure 8.

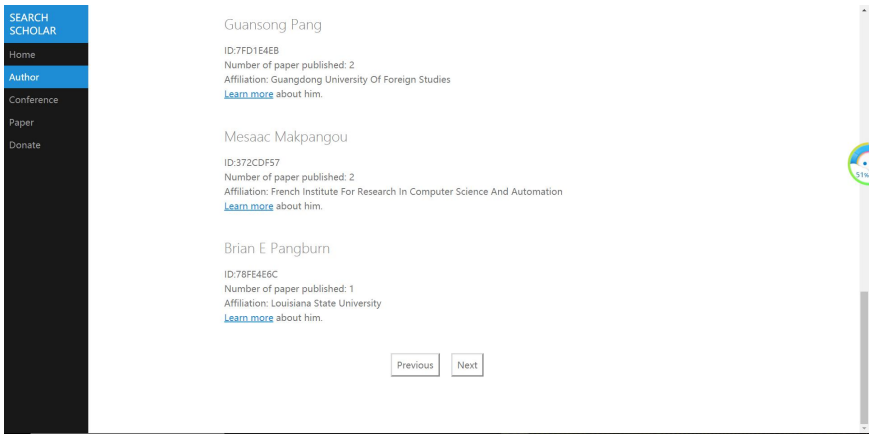


Figure 8: Before the direct-turn navigation button

And the new one is like Figure 9 and Figure 10.

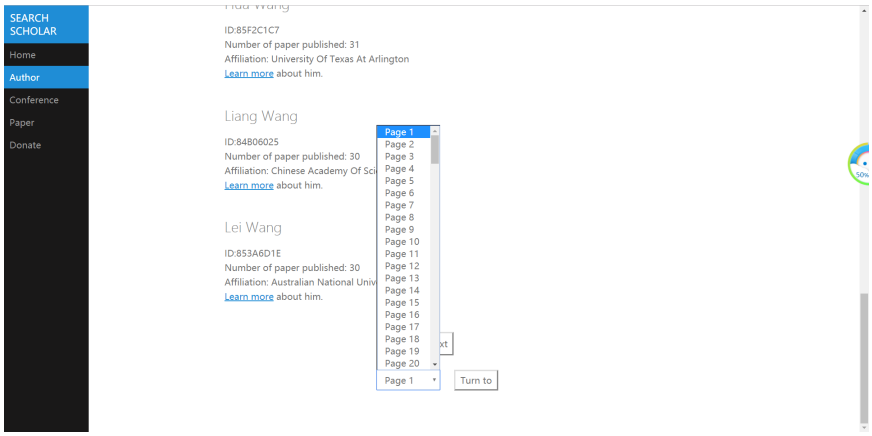


Figure 9: After the direct-turn button

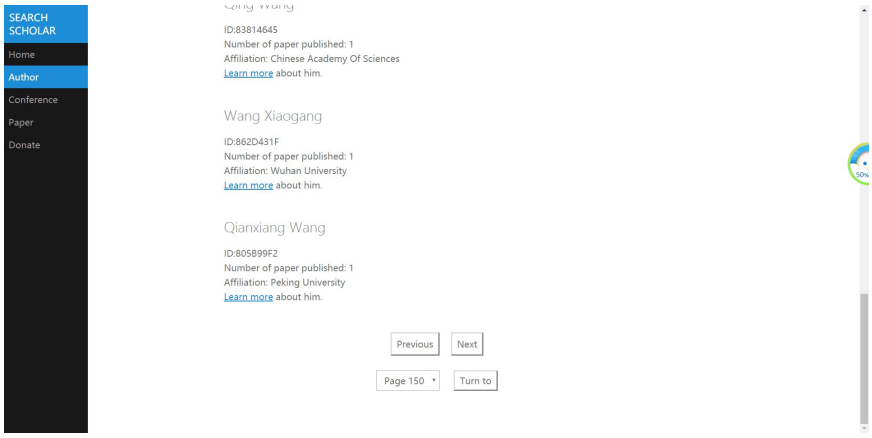


Figure 10: The effect of the direct-turn button

3 Detailed Conference Page

For the detailed conference Page, we demonstrated three aspects of the conference, which are, papers published in the conference, active authors of the conference, and the number of published papers in the conference. We will introduce the three parts separately.

3.1 Published papers

Published papers is a normal section of conference page, nearly everyone will come up with it, so we only give a brief introduction to it. The papers are collected from the conference and then sorted by the number of being cited. The extraction process is shown in the mysql code below.

```
SELECT
papers.paperID, COUNT(referenceID) AS num
FROM
papers
LEFT JOIN
paperreference ON papers.paperID = paperreference.referenceID
WHERE
conferenceID = '$conferenceID'
GROUP BY papers.paperID
ORDER BY COUNT(referenceID) DESC
```

Then we need to get all the authors of the paper, just as we have done before. We just show the mysql code again, without any interpretations.

```
SELECT
=TITLE, authors.AUTHORNAME, authors.AUTHORID, papers.PAPERID
FROM
academicrecord.papers
JOIN
```

```
academicrecord.paaffiliation ON papers.PAPERID = paaffiliation.
    PAPERID
JOIN
academicrecord.authors ON paaffiliation.AUTHORID = authors.
    AUTHORID
JOIN
academicrecord.conferences ON papers.CONFERENCEID = conferences
    .CONFERENCEID
WHERE
paaffiliation.PAPERID = '$paperid'
ORDER BY AUTHORSEQUENCE
```

And after all the search work done, we just need to use a circuit to do all the repeated search of the authors, and then show it on the webpage. The display of data will be talked about in the Front-end Development part.

3.2 Active authors

The active author part enables the viewer to access the active authors of the conference. The authors' activeness is valued by the number of his paper published in this conference. Otherwise, if we sort the authors by their total number of publication, the several authors with great number of publication, like Xiaoou Tang of CUHK, Jiawei Han of UIUC, Michael Jordan of UCB, etc., will be at the top of almost every conference. However, there is a disadvantage of sorting the authors by their local publish number, which is, the search code runs much slower if we sort the authors by their global publish number, because the sql database has to judge whether an author's paper is in the conference, which takes a very long time.

```
SELECT
paaffiliation.AUTHORID,
authors.AUTHORNAME,
COUNT(paaffiliation.AUTHORID) AS num
FROM
paaffiliation
JOIN
papers ON papers.paperID = paaffiliation.paperid
JOIN
authors ON authors.authorid = paaffiliation.authorid
WHERE
conferenceID = '$conferenceID'
GROUP BY paaffiliation.authorID
ORDER BY num DESC
```

After selecting all the active authors of the conference, we still need to find the authors' affiliations. This part has also been done before, so we will not waste pages on it. The sql codes is shown below.

```
SELECT
AFFILIATIONNAME
FROM
academicrecord.affiliations
JOIN
academicrecord.paaffiliation ON paaffiliation.AFFILIATIONID =
    affiliations.AFFILIATIONID
```

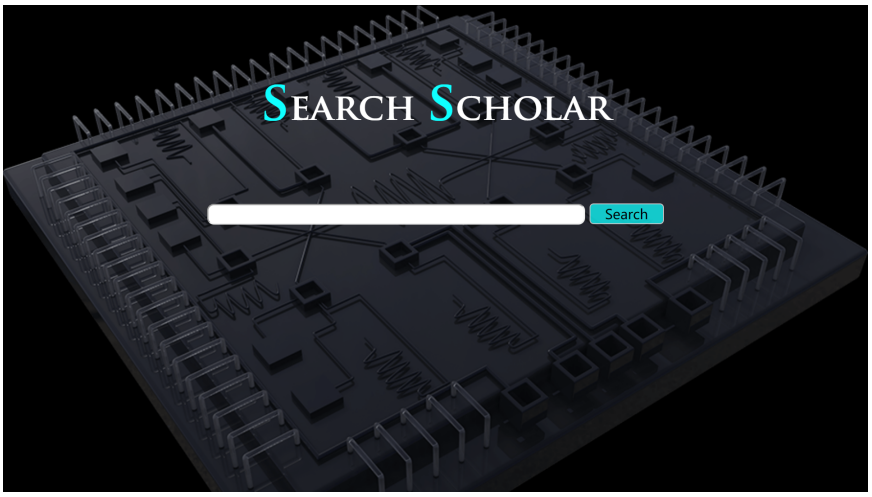


Figure 11: Original homepage

```
WHERE
paaffiliation.authorid = '$authorid'
GROUP BY paaffiliation.AFFILIATIONID
ORDER BY COUNT(paaffiliation.AFFILIATIONID) DESC
```

4 Front-end Development

Front-end development is not probably the most technical work, but it is indeed important to the website. No one wants to see a mess-organized website. Therefore, we need to use a clear css frame, organize the contents clearly, and make enough navigation bars to make the website easy to use. We will introduce these parts one by one.

4.1 CSS Frame

We believe a good academic website should have a clear, concise style, so we abandoned all the distracting elements like changing colors, shaking graphs, etc. Finally we decided to use the CSS frame made by Yahoo [1], the Pure.CSS framework. Then what we need to do is to transplant all the pages we made before into the framework. We only take the homepage as an example. The original homepage is a dark-theme homepage, with limited words and buttons.

After fitting in the frame of Pure CSS, we get a much clear and pretty homepage, with introduction to the website and several buttons.

The css code file and the styles defined are in the original codes given in the folder. Some of the key contents are

```
<head>
```

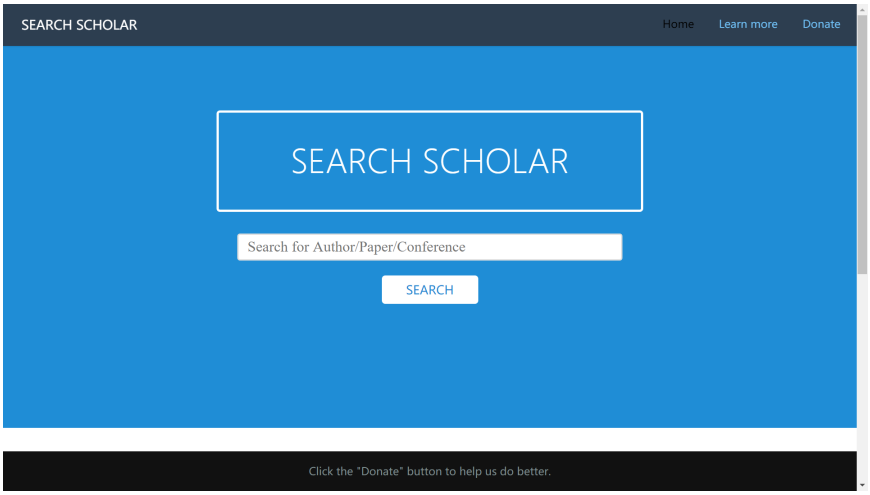


Figure 12: New Homepage, search bar

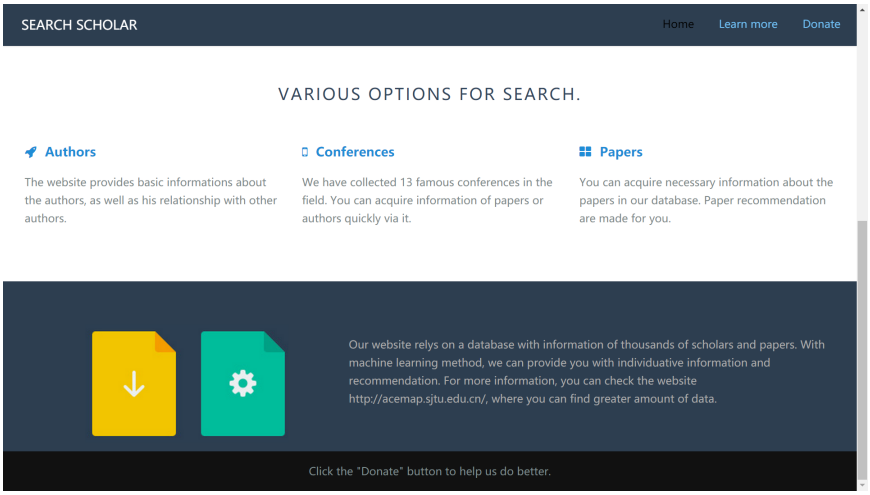


Figure 13: New Homepage, Introduction

```
  <title>Search Scholar: Home</title>

  <link rel="stylesheet" href="https://unpkg.com/purecss@1.0.0/
      build/pure-min.css" integrity="sha384-" crossorigin="
      anonymous">

  <!--[if lte IE 8]>
  <link rel="stylesheet" href="https://unpkg.com/purecss@1.0.0/
      build/grids-responsive-old-ie-min.css">
  <![endif]-->
  <!--[if gt IE 8]><!-->
  <link rel="stylesheet" href="https://unpkg.com/purecss@1.0.0/
      build/grids-responsive-min.css">
  <!--<![endif]-->

  <link rel="stylesheet" href="https://netdna.bootstrapcdn.com/
      font-awesome/4.0.3/css/font-awesome.css">

  <!--[if lte IE 8]>
  <link rel="stylesheet" href="css/layouts/marketing-old-ie.css
      ">
  <![endif]-->
  <!--[if gt IE 8]><!-->
  <link rel="stylesheet" href="css/layouts/marketing.css">
  <!--<![endif]-->
</head>

<body>
  <div class="header">
    <div class="home-menu pure-menu pure-menu-horizontal pure-
        menu-fixed">
      <a class="pure-menu-heading" href="">Search Scholar</a>

      <ul class="pure-menu-list">
        <li class="pure-menu-item pure-menu-selected"><a href="
            #" class="pure-menu-link">Home</a></li>
        <li class="pure-menu-item"><a href="http://ieee.seiee.
            com/" target="_blank" class="pure-menu-link">Learn
            more</a></li>
        <li class="pure-menu-item"><a href="donate.php?keyWord=
             " class="pure-menu-link">Donate</a></li>
      </ul>
    </div>
  </div>

  <div class="splash-container" style="margin-top:-150px;
      height:900px">
    <div class="splash">
      <h1 class="splash-head">Search Scholar</h1>
      <p class="splash-subhead">
        <form action="index.php" method="get" style="font-size
            :20px;" class="pure-form"; id="searchForm">

          <input type="text" id="keyWord" name="keyWord" class=
              "pure-form" style="font-size:22px; font-family:
              Times New Roman; width:570px;height:40px"
              placeholder="Search for Author/Paper/Conference">
```

```
                <a onclick="document.searchForm.submit()" href="
                javascript:" class="pure-button pure-button-
                primary" style="font-family:Microsoft Yahei; font
                -size:18px;">Search</a>

          </form>
        </p>
      </div>
    </div>
</body>
```

The codes above are the codes for generating the navigation bar and the search bar. The introduction part is generated through similar techniques.

4.2 Links to the Back-end

In general website development, we usually leave a place in the front-end for the back-end to transfer data. In our website, before MVC, we used a container in html file to contain all the html data transferred from the back-end.

```
<div class="container">
  <div class="list">
  </div><br>
</div>
```

In the above sample code, the container division is used to hold all the html data transferred from the back-end, while the list division in the container division is used to hold the navigation button, i.e. the "next" and "previous" and "jump to" button. And then we just use javascript to get the choice of the user (which content he'd like to see), and then control the contents transferred from the back-end. Here shows a sample code from the index.php, where the search results are demonstrated.

```
    var choice = 2;
window.onload=function(){
  $("#homeButton").click(function(){
    $("li").removeClass();
    $("#homeButton").addClass("pure-menu-item menu-item-divided
         pure-menu-selected");
    $("#authorButton").addClass("pure-menu-item");
    $("#conferenceButton").addClass("pure-menu-item");
    $("#paperButton").addClass("pure-menu-item");
    $("#donate").addClass("pure-menu-item");
    choice = 1;
    loadDataByPageIndex(1);
  });

  $("#authorButton").click(function(){
    $("li").removeClass();
    $("#homeButton").addClass("pure-menu-item");
    $("#authorButton").addClass("pure-menu-item menu-item-
        divided pure-menu-selected");
    $("#conferenceButton").addClass("pure-menu-item");
    $("#paperButton").addClass("pure-menu-item");
    $("#donate").addClass("pure-menu-item");
```

```
    choice = 2;
    loadDataByPageIndex(1);
  });

  $("#conferenceButton").click(function(){
    $("li").removeClass();
    $("#homeButton").addClass("pure-menu-item");
    $("#authorButton").addClass("pure-menu-item");
    $("#conferenceButton").addClass("pure-menu-item menu-item-
        divided pure-menu-selected");
    $("#paperButton").addClass("pure-menu-item");
    $("#donate").addClass("pure-menu-item");
    choice  = 3;
    loadDataByPageIndex(1);
  });

  $("#paperButton").click(function(){
    $("li").removeClass();
    $("#homeButton").addClass("pure-menu-item");
    $("#authorButton").addClass("pure-menu-item");
    $("#conferenceButton").addClass("pure-menu-item");
    $("#paperButton").addClass("pure-menu-item menu-item-
        divided pure-menu-selected");
    $("#donate").addClass("pure-menu-item");
    choice = 4;
    loadDataByPageIndex(1);
  })
}
var start_time = 0;
var end_time = 1000;
var pageIndex = 1;
var keyWord = "<?php echo $keyWord?>"; // change the keyWord
    recieved by php into javascript format

function loadDataByPageIndex(pageIndex) {
  $(".list").slideToggle(800,function(){$(".loading").show();})
      ;
  start_time=Date.now();
  if(choice == 2){
    $.get("authorResultBackend.php", {
      pageIndex: pageIndex,
      scholarName: keyWord
    }, function (data) {
      $(".list").html(data);
      end_time=Date.now();
      console.log("Load was performed.", "Time: ", end_time -
          start_time);
      $(".loading").hide();
      $(".list").slideToggle(1000);
    });
  }
  if(choice == 3){
    $.get("conferenceResultBackend.php", {
      pageIndex: pageIndex,
      conferenceName: keyWord
    }, function (data) {
      $(".list").html(data);
```

```
73        end_time=Date.now();
74        console.log("Load was performed.", "Time: ", end_time -
              start_time);
75        $(".loading").hide();
76        $(".list").slideToggle(1000);
77      });
78    }
79    if(choice == 4){
80      $.get("paperResultBackend.php", {
81        pageIndex: pageIndex,
82        paperName: keyWord
83      }, function (data) {
84        $(".list").html(data);
85        end_time=Date.now();
86        console.log("Load was performed.", "Time: ", end_time -
              start_time);
87        $(".loading").hide();
88        $(".list").slideToggle(1000);
89      });
90    }
91  }
92
93  function next() {
94    loadDataByPageIndex(++pageIndex);
95    console.log('NEXT', pageIndex);
96  };
97
98  function previous() {
99    if (pageIndex > 1) {
100     loadDataByPageIndex(--pageIndex);
101   } else {
102
103   }
104   console.log('previous', pageIndex);
105 };
106
107 function page()
108 {
109   var da=document.getElementById("select").value;
110   pageIndex = da;
111   loadDataByPageIndex(da);
112   console.log("NEXT: ", da);
113 }
```

5 Data visualization

5.1 teacher-student relationship

5.1.1 Problem Analysis

In this problem, we are required to form a photograph to demonstrate the relationships between the teacher and his/her students. So firstly we should divide this into two steps: Initially to use php language to write a document in the form of "json" to pass into html and js part, and use this tree diagram written in

the form of "json" as a treedata. Then by js language, we could use the treedata to form the photograph we want and present it in the pages wherever we want.

5.1.2 Solution Design

So we now should write the json document. So we ought to firstly connect to the mysql and search for the authorID and authorName we input and get the relationships between him/her and his/her students to form a table. Then we push all the statics into several arrays and begin our writing of the document. Via using some loops to input the data and judgements of the "if_teacher", we could successfully write down the json document. After the work above, we are now faced with the problem of js codes to fulfill our dreams to form a tree to present the relationships. In the fact that it's a little bit difficult to conclude the ways of it, We will describe the ways we used in the source code part.

5.1.3 Source code of json writing

```
<?php
$total = 0;
```

Here we use variable 'total' to get the students number and later we will deliver this into js.

```
$teacher = $_GET["authorId"];
$teacherName = $_GET["authorName"];
$connect = mysqli_connect("localhost","root","");

// connect to mysql
if (!$connect)
{
die ('Could not connect: '.mysqli_connect_error());
}

mysqli_select_db($connect, "academicrecord");

$sql1 = "SELECT DISTINCT bID, AUTHORNAME
FROM
academicrecord.if_teacher
JOIN
academicrecord.authors
ON
if_teacher.bID = authors.AUTHORID
WHERE
aID = '".$teacher."'
";
```

form the table of the first layer(the first level of the relationship)

```
$result1 = mysqli_query($connect,$sql1);
$middle_id = array();
$middle_name = array();

```

```
while ($row1 = mysqli_fetch_assoc($result1))
{
array_push($middle_id, $row1["bID"]);
array_push($middle_name, ucwords($row1["AUTHORNAME"]));
}
```

here we append the data into the array(first layer)

```
$filename = "tmp.json";
$content = "[{
\"name\":\"".$teacherName."\",
\"parent\":\"null\",
\"value\":15,
\"type\":\"black\",
\"level\":\"steelblue\",
\"children\":[";
$handle = fopen($filename, "w");
$str = fwrite($handle, $content);
// to write the root(grandfather)
$len1 = sizeof($middle_id);
for($i=0;$i<$len1;$i++)
{
$sql = "SELECT DISTINCT bID, AUTHORNAME
FROM
academicrecord.if_teacher
JOIN
academicrecord.authors
ON
if_teacher.bID = authors.AUTHORID
WHERE
aID = '".$middle_id[$i]."'
";
```

in the loop, we can distinctly search the ID to form the table of the second layer(the second level of relationship)

```
$result = mysqli_query($connect,$sql);
$bottom_id = array();
$bottom_name = array();

while ($row = mysqli_fetch_assoc($result))
{
array_push($bottom_id, $row["bID"]);
array_push($bottom_name, ucwords($row["AUTHORNAME"]));
}
```

append the data into the array(second layer)

```
$len2 = sizeof($bottom_id);

if($len2 != 0)
```

the aim of the judgement of whether len2==0 is to make sure whether this teacher has students and thus to ensure whether we should write 'children' in his/her part.

```

{
if($i!=$len1-1)
```

the aim of the judgement of whether i==len1-1 is to ensure the formula of the json file, because at the end of every loops it must contain a '}]'

```
{
$content = "
{
\"name\":\"".$middle_name[$i]."\",
\"parent\":\"".$teacherName."\",
\"value\":12,
\"type\":\"grey\",
\"level\":\"red\",
\"children\":[";
$str = fwrite($handle, $content);
for($j=0;$j<$len2;$j++)
{
if($j!=$len2-1)
{
$content = "
{
\"name\":\"".$bottom_name[$j]."\",
\"parent\":\"".$middle_name[$i]."\",
\"value\":6,
\"type\":\"steelblue\",
\"level\":\"orange\"
},";
$str = fwrite($handle, $content);
$total += 1;
}
if($j==$len2-1)
{
$content = "
{
\"name\":\"".$bottom_name[$j]."\",
\"parent\":\"".$middle_name[$i]."\",
\"value\":6,
\"type\":\"steelblue\",
\"level\":\"orange\"
}
]";
$str = fwrite($handle, $content);
$total += 1;
}
}
$content = "},";
$str = fwrite($handle, $content);
}
if($i==$len1-1)
{
$content = "
{
\"name\":\"".$middle_name[$i]."\",
\"parent\":\"".$teacherName."\",
\"value\":12,
\"type\":\"grey\",
\"level\":\"red\",
\"children\":[";
$str = fwrite($handle, $content);
for($j=0;$j<$len2;$j++)
```

```
{
if($j!=$len2-1)
{
$content = "
{
\"name\":\"".$bottom_name[$j]."\",
\"parent\":\"".$middle_name[$i]."\",
\"value\":6,
\"type\":\"steelblue\",
\"level\":\"orange\"
},";
$str = fwrite($handle, $content);
}
if($j==$len2-1)
{
$content = "
{
\"name\":\"".$bottom_name[$j]."\",
\"parent\":\"".$middle_name[$i]."\",
\"value\":6,
\"type\":\"steelblue\",
\"level\":\"orange\"
}
]";
$str = fwrite($handle, $content);
}
$total += 1;
}
$content = "
}
]";
$str = fwrite($handle, $content);
}

}
if($len2 == 0)
{
if($i!=$len1-1)
{
$content = "
{
\"name\":\"".$middle_name[$i]."\",
\"parent\":\"".$teacherName."\",
\"value\":9,
\"type\":\"grey\",
\"level\":\"green\"
},";
$str = fwrite($handle, $content);
}
if($i==$len1-1)
{
$content = "
{
\"name\":\"".$middle_name[$i]."\",
\"parent\":\"".$teacherName."\",
\"value\":9,
\"type\":\"grey\",
```

```
\"level\":\"green\"
}
]";
$str = fwrite($handle, $content);
}
}
}
$content = "
}]";
$str = fwrite($handle, $content);
fclose($handle);
//finishing writing
session_start();
$_SESSION["total"] = $total;
// to deliver the value of total into "authorStudent.php"
?>
```

now we have finished the writing of the json document. The following is the implementation of the tree.

5.1.4 source code of the js

```
<div class="content pure-u-1 pure-u-md-3-4" id="MainContent">
<!-- teacher student relationship -->
<div class='posts'>
<h1 class='content-subhead'>Students</h1>
<div id="load"><img src="loading.gif"></div>

<style type="text/css">
.node {
cursor: pointer;
}
```

the CSS-part is to represent the type of the circles and the links between them.

```
.node circle {
fill: #fff;
stroke: steelblue;
stroke-width: 3px;
}

.node text { font: 12px sans-serif; }

.link {
fill: none;
stroke: #ccc;
stroke-width: 2px;
}

</style>

<!-- load the d3.js library -->
<script src="https://code.jquery.com/jquery-1.12.4.js"></script
    >
<script src="http://d3js.org/d3.v3.min.js"></script>

<script>
var authorId = "<?php echo $authorId;?>";
```

```
var totalNum = 100;
var if_complete = false;
$(document).ready(function(){
$.get("authorStudentBackend.php",{
authorId: authorId,
authorName: "<?php echo $authorName;?>",
});
// wait for the php to run
setTimeout(function() {
d3.json("tmp.json", function(error, treeData) {
$("#load").hide();
root = treeData[0];
update(root);
});
}, 15000);

// load the external data

});

// ************** Generate the tree diagram  *****************
```

Here we announced some function including the size and shape of the svg vessel.

```
var margin = {top: 20, right: 120, bottom: 20, left: 600},
width = 1700 - margin.right - margin.left,
height = totalNum * 10 - margin.top - margin.bottom;

var i = 0;
duration = 750;

var tree = d3.layout.tree()
.size([height, width]);
```

Then we use the d3.js diagonal function to help draw a path between two points such that the line exhibits some nice flowing curves to make the connection.

```
var diagonal = d3.svg.diagonal()
.projection(function(d) { return [d.y, d.x]; });

var svg = d3.select("body").append("svg")
.attr("width", width + margin.right + margin.left)
.attr("height", height + margin.top + margin.bottom)
.append("g")
.attr("transform", "translate(" + margin.left + "," + margin.
    top + ")");



function update(source) {

// Compute the new tree layout.
var nodes = tree.nodes(root).reverse(),
links = tree.links(nodes);

// Normalize for fixed-depth.
nodes.forEach(function(d) { d.y = d.depth * 180; });
```

We then declare the variable / function node so that when we call it later it will know to select the appropriate object (a node) with the appropriate .id.

```
var node = svg.selectAll("g.node")
.data(nodes, function(d) { return d.id || (d.id = ++i); });
```

Now we enter the nodes.

```
var nodeEnter = node.enter().append("g")
.attr("class", "node")
.attr("transform", function(d) {
return "translate(" + d.y + "," + d.x + ")"; })
.on("click", click);

nodeEnter.append("circle")
.attr("r", function(d) { return d.value; })
.style("stroke", function(d) { return d.type; })
.style("fill", function(d) { return d._children ? d.level : "#
    fff"; });

nodeEnter.append("text")
.attr("x", function(d) {
return d.children || d._children ?
(d.value + 4) * -1 : d.value + 4 })
.attr("dy", ".35em")
.attr("text-anchor", function(d) {
return d.children || d._children ? "end" : "start"; })
.text(function(d) { return d.name; })
.style("fill-opacity", 1);
```

Here we transition nodes to their new position.

```
var nodeUpdate = node.transition()
.duration(duration)
.attr("transform", function(d) { return "translate(" + d.y + ",
    " + d.x + ")"; });

nodeUpdate.select("circle")
.attr("r", 10)
.style("fill", function(d) { return d._children ? d.level : "#
    fff"; });

nodeUpdate.select("text")
.style("fill-opacity", 1);
```

Transition exiting nodes to the parent's new position.

```
var nodeExit = node.exit().transition()
.duration(duration)
.attr("transform", function(d) { return "translate(" + source.y
     + "," + source.x + ")"; })
.remove();

nodeExit.select("circle")
.attr("r", 1e-6);

nodeExit.select("text")
.style("fill-opacity", 1e-6);
```

Then is the update of the links.

```
var link = svg.selectAll("path.link")
.data(links, function(d) { return d.target.id; });

// Enter any new links at the parent's previous position.
link.enter().insert("path", "g")
.attr("class", "link")
.style("stroke", function(d) { return d.target.level; })
.attr("d", function(d) {
var o = {x: source.x0, y: source.y0};
return diagonal({source: o, target: o});
});

// Transition links to their new position.
link.transition()
.duration(duration)
.attr("d", diagonal);

// Transition exiting nodes to the parent's new position.
link.exit().transition()
.duration(duration)
.attr("d", function(d) {
var o = {x: source.x, y: source.y};
return diagonal({source: o, target: o});
})
.remove();

// Stash the old positions for transition.
nodes.forEach(function(d) {
d.x0 = d.x;
d.y0 = d.y;
});
}
```

Toggle children on click.

```
function click(d) {
if (d.children) {
d._children = d.children;
d.children = null;
} else {
d.children = d._children;
d._children = null;
}
update(d);
}
</script>
</div>
</div>
</div>
```

5.1.5 Results display

After the source code, we may display the results we get via this method. The following is effect of the code.

5.1.6 some improvements we did

After the realization of the tree diagram represents the relationship between teachers and students, we think it too monotonous and decide to make it more gorgeous. So this is the time for beautification. After glancing over many examples of this, we find that if we add three properties into the json file for every people, this will be easy. So we add three value, type, level to respectively control the size and color to make it more gorgeous and clear. This effect has been demonstrated in the graph.

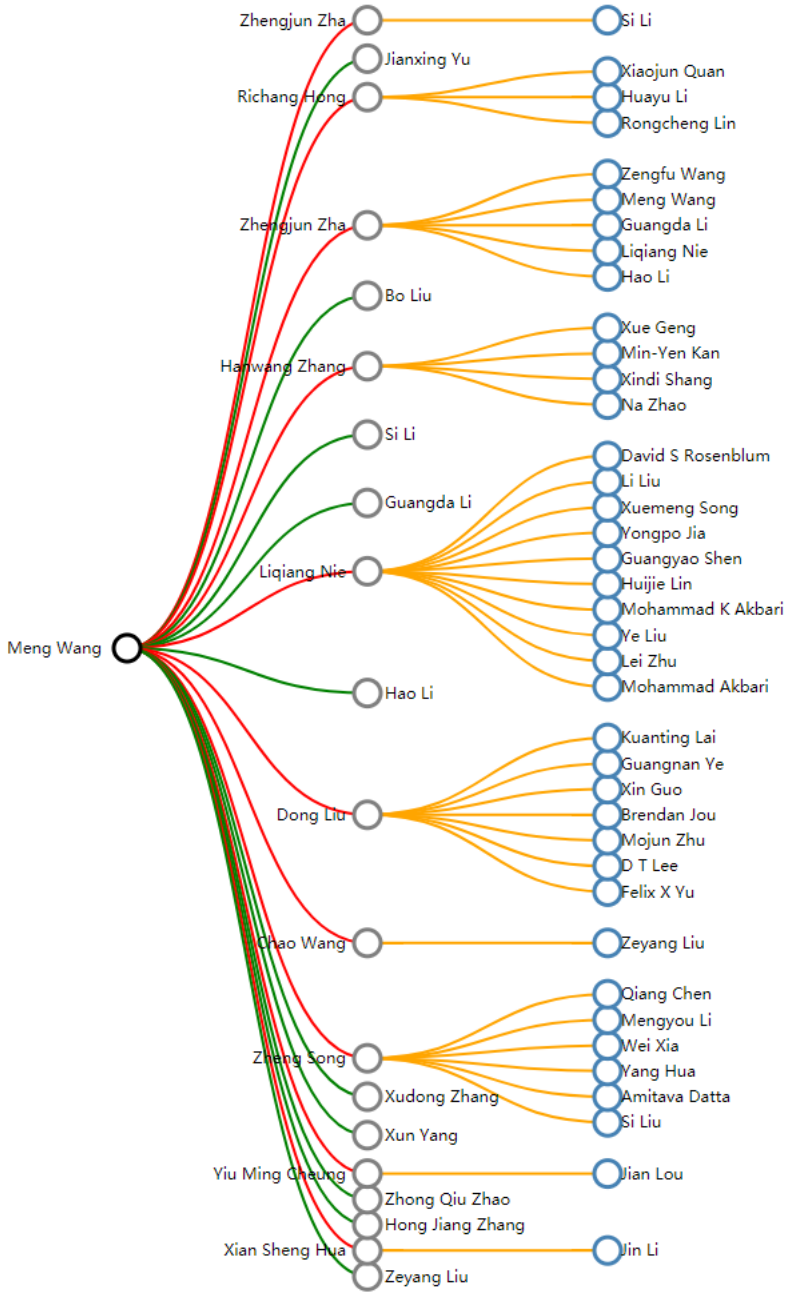


Figure 14: This is the total relationships of Wang Meng.

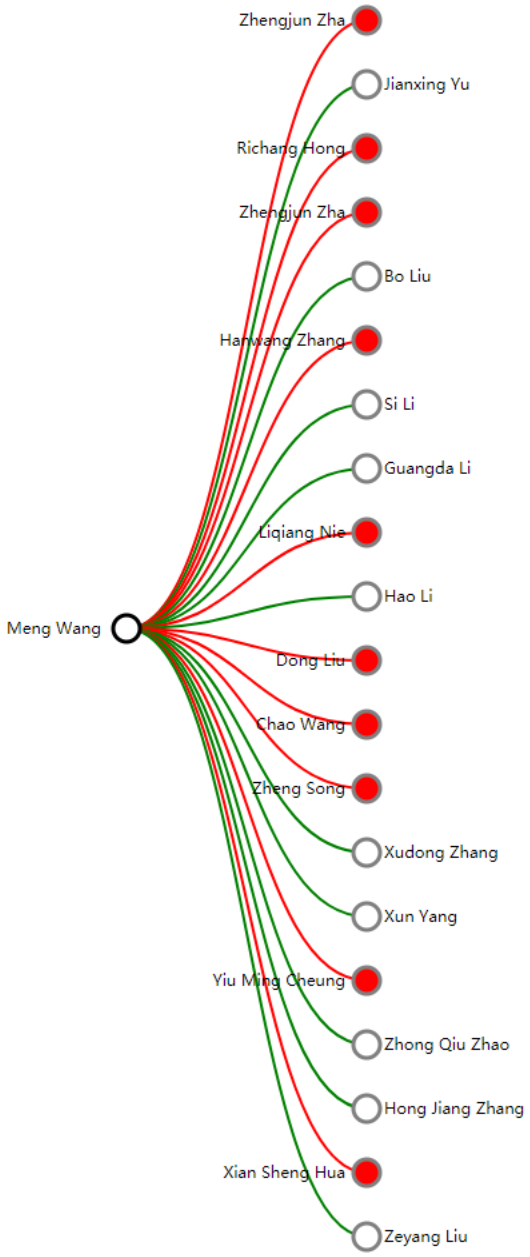


Figure 15: Now we just present one layer and the result is like this.

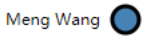


Figure 16: Now we only have one single node.

5.2 author cooperator

5.2.1 Problem Analysis

In this part, we are asked to form the graph to demonstrate the cooperations between authors and use Force-Directed Graph to indicate it. So just like the procedure above, we could solve this problem via that way.

5.2.2 Solution Design

Though I have mentioned that this is very similar to the problem above, the formula in the json and some codes in javascript differs a lot. And the database that we constructed is more complex than the former one. But the main procedure is the same. Details will be discussed in the source code.

5.2.3 Source code of js writing

Author1	Name1	Author2	Name2	Label1	Label2
000E5E1E	ramasamy uthurus	7F612BC9	padhraic smyth	0	1
000E5E1E	ramasamy uthurus	793344FC	inderjit s dhillon	0	0
000E5E1E	ramasamy uthurus	7D70A55	paul s bradley	0	0
000E5E1E	ramasamy uthurus	7F6E941F	rajesh parekh	1	0
000E5E1E	ramasamy uthurus	7EF4F63A	jingrui he	0	0
000E5E1E	ramasamy uthurus	753464A2	yehuda koren	1	0
000E5E1E	ramasamy uthurus	7E31FCC	rayid ghani	0	0
000E5E1E	ramasamy uthurus	7CE36F02	robert grossman	0	0
000E5E1E	ramasamy uthurus	0ADCAA8	ted e senator	0	0
000E5E1E	ramasamy uthurus	242EDBF	balaji krishnapurar	1	0
000E5E1E	ramasamy uthurus	8525A378	jonathan silverstei	1	0
000E5E1E	ramasamy uthurus	7DC9EEB	jimeng sun	0	0
000E5E1E	ramasamy uthurus	7D1717C8	jianying hu	0	0
000E5E1E	ramasamy uthurus	7F6DB0E3	john younger	1	0
000E5E1E	ramasamy uthurus	76014D6E	nitesh v chawla	0	0
000E5E1E	ramasamy uthurus	80045A2F	fei wang	0	0
000E5E1E	ramasamy uthurus	776F59C0	david madigan	1	0
000E5E1E	ramasamy uthurus	762149CF	k p unnikrishnan	1	0

Figure 17: Label1 indicates that 1 is a cooperator of 2, Label2 indicates 2 is a cooperator of 1

```
<?php
for($i=0;$i<$len;$i++)
{
if((int)$label1[$i]==1 and (int)$label2[$i]==0)
```

```
{
$content="    {\"id\":\"".$array[$i]." ".$arrayid[$i]."\",\"
    group\":2},
";
$str=fwrite($handle,$content);
}

if((int)$label1[$i]==0 and (int)$label2[$i]==1)
{
$content="    {\"id\":\"".$array[$i]." ".$arrayid[$i]."\",\"
    group\":3},
";
$str=fwrite($handle,$content);
}

if(((int)$label1[$i]==0 and (int)$label2[$i]==0) or (int)
    $label1[$i]==1 and (int)$label2[$i]==1)
{
$content="    {\"id\":\"".$array[$i]." ".$arrayid[$i]."\",\"
    group\":4},
";
$str=fwrite($handle,$content);
}
}
// this part is aimed to ensure the color(group presents it)
    and the fundemental author
for($i=0;$i<$len;$i++)
{
$sql3 = "SELECT Author2,Name2 FROM co_relation WHERE Author1='"
     . $arrayid[$i] . "';";
$result3 = mysqli_query($con,$sql3);
while($row3 = mysqli_fetch_array($result3))
{
if(in_array($row3[0],$arrayid))
{
$content="    {\"source\":\"".$array[0]." ".$arrayid[0]."\",\"
    target\":\"".$row3[1]." ".$row3[0]."\",\"value\":1}";
$str=fwrite($handle,$content);
break;
}
}
if($row3)
{
if(in_array($row3[0],$arrayid))
{break;}
}
}
// this part is to add the cooperators into json document.
?>
```

note:group represents the color, source and target represent the two nodes of the link.
Now we have finished the json document of the author-cooperator graph, the next is js code.

```
<div class="content pure-u-1 pure-u-md-3-4" id="MainContent">
<!-- cooperator relationship -->
```

```
<div class='posts'>
<h1 class='content-subhead'>Cooperators</h1>
<div id="load" style="text-align:center; margin-top:70px"><img
    src="loading.gif"></div>

<style type="text/css">
.links line {
stroke: #888;
stroke-opacity: 0.6;
}

.nodes circle {
stroke: #fff;
stroke-width: 1.5px;
}
</style>

<svg width="760" height="400"></svg>
<script src="https://code.jquery.com/jquery-1.12.4.js"></script
    >
<script src="https://d3js.org/d3.v4.min.js"></script>

<script>
var authorId = "<?php echo $authorId;?>";
$(document).ready(function(){
$.get("authorCooperatorBackend.php",{
authorId: authorId,
authorName: "<?php echo $authorName;?>",
});
});


// load the external data
```

now we are going to define a svg vancas and ascertain its height and width.

```
var svg = d3.select("svg"),
width = +svg.attr("width"),
height = +svg.attr("height");

var color = d3.scaleOrdinal(d3.schemeCategory20);
```

This is the definition of the color function.

```
var simulation = d3.forceSimulation()
.force("link", d3.forceLink().id(function(d) { return d.id; }))
.force("charge", d3.forceManyBody())
.force("center", d3.forceCenter(width / 2, height / 2));
```

To create a mechanical simulator.

```
// wait for the php to run

setTimeout(function() {
$("#load").hide();
d3.json("tmp1.json", function(error, graph) {
if (error) throw error;

var link = svg.append("g")
```

```
.attr("class", "links")
.selectAll("line")
.data(graph.links)
.enter().append("line")
.attr("stroke-width", function(d) { return Math.sqrt(d.value);
    });
```

Define the link message.

```
var node = svg.append("g")
.attr("class", "nodes")
.selectAll("circle")
.data(graph.nodes)
.enter().append("circle")
.attr("r", 5)
.attr("fill", function(d) { return color(d.group); })
.call(d3.drag()
.on("start", dragstarted)
.on("drag", dragged)
.on("end", dragended));
```

Define the nodes of the people.

```
node.append("title")
.text(function(d) { return d.id; });

simulation
.nodes(graph.nodes)
.on("tick", ticked);

simulation.force("link")
.links(graph.links);

function ticked() {
link
.attr("x1", function(d) { return d.source.x; })
.attr("y1", function(d) { return d.source.y; })
.attr("x2", function(d) { return d.target.x; })
.attr("y2", function(d) { return d.target.y; });

node
.attr("cx", function(d) { return d.x; })
.attr("cy", function(d) { return d.y; });
}
});
}, 2000);
```

Define the behavoir once drafting the nodes.

```
function dragstarted(d) {
if (!d3.event.active) simulation.alphaTarget(0.3).restart();
d.fx = d.x;
d.fy = d.y;
}
```

Define the behavoir while drafting.

```
function dragged(d) {
d.fx = d3.event.x;
d.fy = d3.event.y;
}
```

Define the behavoir when drafting is over.

```
function dragended(d) {
if (!d3.event.active) simulation.alphaTarget(0);
d.fx = null;
d.fy = null;
}
</script>

</div>
</div>
</div>
```

5.2.4 Result Display

The following is effect of the Force-Directed Graph.

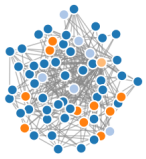


Figure 18: This is the cooperators of Wang Meng.

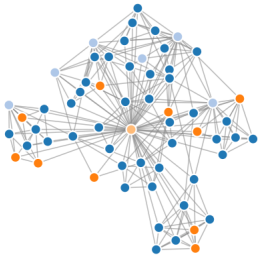


Figure 19: After drafting, the shape and position have changed.

6 Paper Recommendation

We tried to recommend paper by their similarities. More similar papers are more likely to be in the recommend list. There are many possible links between papers that show their similarities, and here are the three variables that we choose to symbolize the similarity between papers.

- Papers that are cited, have cited, sharing the same citation with this paper, or being the same citation with this paper of other papers.
- Papers that share the same authors with this paper.
- Papers that share the same keywords in the title with this paper.

The first and second links are easy to evaluate. We just need to write corresponding mysql codes and these variables will be easily retracted. For the first one,

```
def paperWithSameCitation(self, paperID, cursor):
  # returns 2-dimensional list with [[paperID,
      sameReferenceNumber]]
  command = """
    SELECT
    paperid
    FROM
    academicrecord.paperreference
    WHERE
    referenceid IN (SELECT
    REFERENCEID
    FROM
    academicrecord.paperreference
    WHERE
    paperid = '%s')
    and paperid != '%s'""" % (paperID, paperID)
  cursor.execute(command)
  result = cursor.fetchall()
  return_list = []
  all_papers = []
  for i in result:
    if i[0] in all_papers:
      position = all_papers.index(i[0])
      return_list[position][1] += 1
    else:
      all_papers.append(i[0])
      return_list.append([i[0], 1])
  return return_list

def paperWithSameBeingReference(self, paperID, cursor):
  # also returns 2-dimensional list [[paperID, Num]]
  command = """
    SELECT
    REFERENCEID
    FROM
    academicrecord.paperreference
    WHERE
    paperid IN (SELECT
    paperid
    FROM
    academicrecord.paperreference
    WHERE
    referenceid = '%s')
    AND referenceid != '%s'""" % (paperID, paperID)
  cursor.execute(command)
  result = cursor.fetchall()
```

```
  return_list = []
  all_papers = []
  for i in result:
    if i[0] in all_papers:
      position = all_papers.index(i[0])
      return_list[position][1] += 1
    else:
      all_papers.append(i[0])
      return_list.append([i[0], 1])
  return return_list

def paperCiting(self, paperID, cursor):
  # returns 1-dimensional list [paperID]
  command = """
    SELECT
    paperid
    FROM
    academicrecord.paperreference
    WHERE
    referenceid = '%s'""" % paperID
  cursor.execute(command)
  result = cursor.fetchall()
  return_list = []
  for i in result:
    return_list.append(i[0])
  return return_list

def paperBeingCitedBy(self, paperID, cursor):
  # returns 1-dimensional list [paperID]
  command = """
    SELECT
    referenceid
    FROM
    academicrecord.paperreference
    WHERE
    paperid = '%s'""" % paperID
  cursor.execute(command)
  result = cursor.fetchall()
  return_list = []
  for i in result:
    return_list.append(i[0])
  return return_list
```

And for the second one,

```
def paperWithSameCoauthors(self, paperID, cursor):
  # returns 2-dimensional list with [[paperID, co-authorNumber
    ]]
  command = """
    SELECT
    paperid
    FROM
    academicrecord.paaffiliation
    WHERE
    authorid IN (SELECT
    authorid
    FROM
    academicrecord.paaffiliation
```

```
    WHERE
    paperid = '%s')
    and paperid != '%s'""" % (paperID, paperID)

  cursor.execute(command)
  result = cursor.fetchall()
  return_list = []
  all_papers = []
  for i in result:
    if i[0] in all_papers:
      position = all_papers.index(i[0])
      return_list[position][1] += 1
    else:
      all_papers.append(i[0])
      return_list.append([i[0], 1])
  return return_list
```

These two codes are easy to identify, and they returns the number of same characters other similar papers share with it.
Then for the third one, keyword between papers, it becomes more complex. As many papers contain words irrelevant to the content of the paper such as "and", "or", "a", "an", "the", "of", etc. The problem is how to identify words that are related to the theme of the paper and those aren't. A very easy way is to count the words that occurred in the titles and then mark the most frequent ones as irrelevant. This makes sense because if a word occurs too many times, it is less likely to be a research field or research subject. According to Google Scholar, in computer science, the topic, word segmentation, has 1,090,000 papers out of the 5,720,000 total papers. Therefore, we set the occurring rate at $0.2 * 0.2 = 0.04$, considering that the same topic has different sub-topics. Therefore, we marked the words that occurred 4% or more often as common words, which will not be considered in the recommendation procedure. After getting the common word list, it's the time to generate the recommended paper list according to titles.

```
def paperWithSimilarTitle(self, paperID, cursor,
    minLengthOfWord):
  # 2-dimensional list [[paperID, Num]]
  command = """
    SELECT
    title
    FROM
    academicrecord.papers
    WHERE
    PAPERID = '%s'""" % paperID
  cursor.execute(command)
  paperTitle = cursor.fetchall()[0][0]
  print(paperTitle)
  bigWordsInTitle = paperTitle.split(" ")
  return_list = []
  all_papers = []
  for word in bigWordsInTitle:
    if word in common_word_list:
      continue
    print(word)
    command = """
      SELECT
```

```
      paperid
      FROM
      academicrecord.papers
      WHERE
      title like '%%%s%%'
      AND paperid != ' %s '""" % (word, paperID)
    cursor.execute(command)
    result = cursor.fetchall()
    for i in result:
      if i[0] in all_papers:
        position = all_papers.index(i[0])
        return_list[position][1] += 1
      else:
        all_papers.append(i[0])
        return_list.append([i[0], 1])
  return return_list
```

Worth talking, as common words in title does not necessarily mark that these two papers are in common, we give the value of common words as 0.5, while other factors get a value of 1.0. Therefore, the importance of similar words are weakened, and the connection between the original paper and recommended papers are strengthened. In the last, we put all the recommendations together. If the recommendation number exceeds 5 in the first two factors listed, the program will not do the title keyword recommendation to save time. Otherwise, the title keyword recommendation is hold as usual. The list is sorted by recommendation value as returned.

```
def all_recommendation(self, paperID, cursor):
  # 2-dimensional list [[paperID, similarityValue]]
  same_author = PaperRecommendation().paperWithSameCoauthors(
      paperID, cursor=cursor)
  same_citation = PaperRecommendation().paperWithSameCitation(
      paperID, cursor=cursor)
  same_reference = PaperRecommendation().paperWithSameCitation(
      paperID, cursor=cursor)
  being_citation = PaperRecommendation().paperBeingCitedBy(
      paperID, cursor=cursor)
  citing = PaperRecommendation().paperCiting(paperID, cursor=
      cursor)

  return_list = []
  all_papers = []

  for i in same_author:
    if i[0] in all_papers:
      position = all_papers.index(i[0])
      return_list[position][1] += i[1]
    else:
      all_papers.append(i[0])
      return_list.append(i)

  for i in same_citation:
    if i[0] in all_papers:
      position = all_papers.index(i[0])
      return_list[position][1] += i[1]
    else:
```

```
      all_papers.append(i[0])
      return_list.append(i)

  for i in same_reference:
    if i[0] in all_papers:
      position = all_papers.index(i[0])
      return_list[position][1] += i[1]
    else:
      all_papers.append(i[0])
      return_list.append(i)

  for i in being_citation:
    if i in all_papers:
      position = all_papers.index(i)
      return_list[position][1] += 1
    else:
      all_papers.append(i)
      return_list.append([i, 1])

  for i in citing:
    if i in all_papers:
      position = all_papers.index(i)
      return_list[position][1] += 1
    else:
      all_papers.append(i)
      return_list.append([i, 1])

  if len(return_list) < 5:
    similar_title = PaperRecommendation().paperWithSimilarTitle
        (paperID, cursor=cursor, minLengthOfWord=8)
    for i in similar_title:
      if i[0] in all_papers:
        position = all_papers.index(i[0])
        return_list[position][1] += i[1] * 0.5
      else:
        all_papers.append(i[0])
        return_list.append([i[0], i[1] * 0.5])

  sortList(return_list)

  return return_list[:5]


def sortList(l):
  # input: 2-dimensional list
  for i in range(len(l)):
    for j in range(len(l)):
      if l[i][1] < l[j][1]:
        l[i], l[j] = l[j], l[i]
  return
```

Then all the paper recommendation are written into the database to achieve a quicker recommending speed. Only the leading five recommendation papers are written. Others are thrown away.

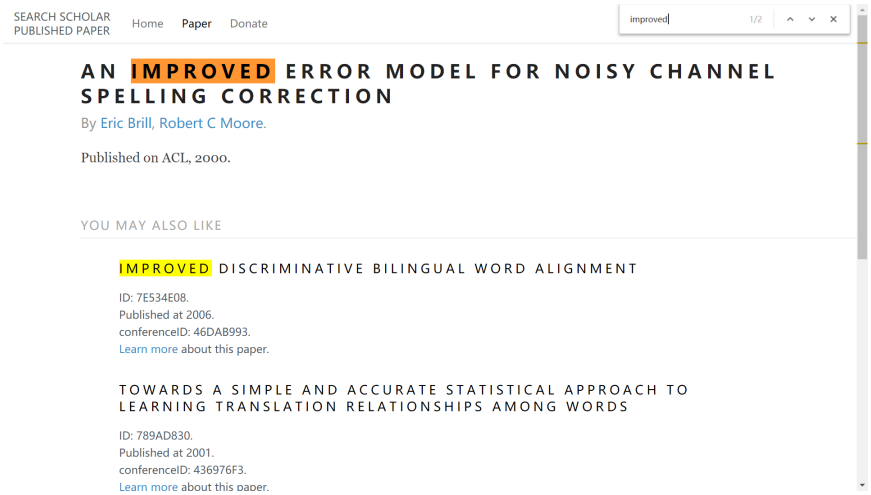


Figure 20: Paper Recommendation

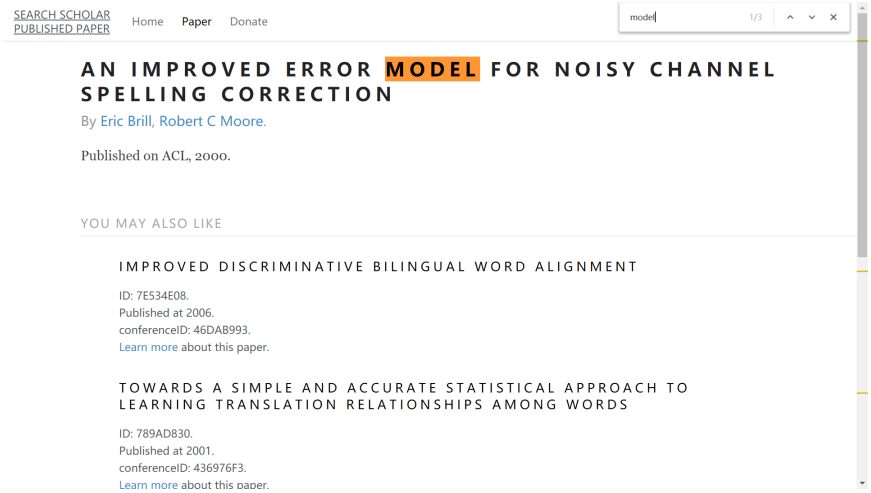


Figure 21: Paper Recommendation

7 Codeigniter

7.1 Reasons to use CI

7.1.1 Importance of framework

Without using the framework, even for a small project like this, it's complicated in structure, both in a single PHP file and the whole program. One reason of this problem is that we need to not only query the MySQL but also do the presentation in a single PHP, the huge amount of code looks bewildering in first glance. Also, we just put all these kind of PHPs simply at the root of localhost, seems in a little disorder.
While the second case above can be tackled by creating folders to divide them into several parts, the best approach to separate the process of searching in SQL and presentation is to use framework. With the framework, we can drastically cut down the lines of code in a single page.

7.1.2 Why to choose Codeigniter

At first, I choose Augular to build the framework, however, it's quite difficult to connect PHP file using Augular (In fact, it connects Node.js) , so the better choice for me is to use CI.

7.1.3 Benefits

Framework for Application. CI provides a rich set of libraries for commonly needed tasks, as well as a simple interface and logical structure to access these libraries. For example, we can easily do the form validation by calling methods in CI library.

M-V-C separation. It uses the Model-View-Controller approach, which allows great separation between logic and presentation.

Fast and Light weight. The core system requires only a few very small libraries. Additional libraries are loaded dynamically upon request, based on your needs for a given process, so the base system is very lean and quite fast.

Free. We are not required to pay money for it.

7.1.4 Application flow chart

Before we start to introduce the application of CI on our program, it's better to first clarify the flow of CI. The index.php serves as the front controller, initializing the base resources needed to run CodeIgniter. The Router examines the HTTP request to determine what should be done with it. (We are not going to use the router now) Before the application controller is loaded, the HTTP

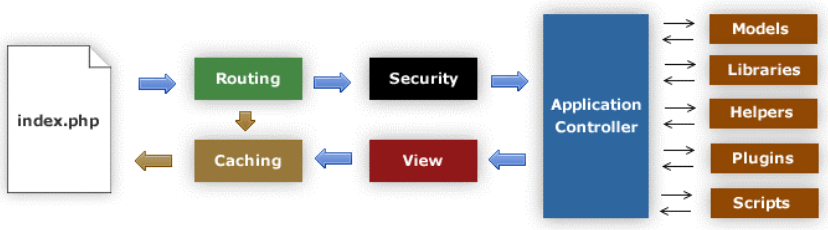


Figure 22: Application flow chart of CI

request and any user submitted data is filtered for security. The controller loads the model, core libraries, helper, and any other resources needed to process the specific request. The finalized View is rendered then sent to the web browser to be seen.

7.2 Carry out CI onto our program

Since our team will hand in the origin code of our project,there's no need to dwell on the specific realization of my part. I world like to just give a brief abstract on our ideas and methods here. But at the very beginning, we should enter the folder 'config' , find PHP file 'database.php' and change data here using your own user name, password and database name like this:

```
'hostname' => 'localhost',
'username' => 'root',
'password' => '',
'database' => 'academicrecord',
```

You are also required to find PHP file 'config.php' in the same folder and find the variable $config['base_url'] and change it into:

```
$config['base_url'] = 'http://localhost/';
```

Finally, you can come into our website by entering the url 'localhost/page/create' (the original URL is 'localhost/index.php/page/create' and we have hiden 'index.php', and we will introduce this improvement later)

7.2.1 Concise introduction of M-V-C separation in our project

The fancy feature of CI is to separate the project into three parts, namely model(M),view(V),controller(C), here we use model to do all the stuff related to searching in MySQL and show the views to the user. We use controller to bring the models and views together and react to the user's requests.
Models. In my part, I create three models:
author_model:do all the search related to authors, has functions:

```
class author_model extends CI_Model {
    public function __construct(){
      $this->load->database();
```

```
    }

    public function count_authors($name){...}

    public function get_authors($name,$page=0){...}

    public function get_affiliation($authorId){...}

    public function count_author_paper($authorId){...}

    public function get_author_paper($authorId,$authorName,
        $page=0){...}

    public function get_author_student($teacher,$teacherName)
        {...}

    public function get_author_cooperator($s_n){...}
}
```

conference_model:do all the search related to conferences, has functions:

```
class conference_model extends CI_Model {
  public function __construct(){
    $this->load->database();
  }

  public function count_conferences($name){...}

  public function get_conferences($name,$page=0){...}

  public function get_num_of_papers($conferenceID){...}

  public function get_graph_information($conferenceID){...}

  public function count_conference_paper($conferenceID){...}

  public function get_conference_paper($conferenceID,$page=0)
      {...}

  public function get_conference_author($conferenceID,$page=0)
      {...}
}
```

paper_model:do all the search related to papers, has functions:

```
class paper_model extends CI_Model {
  public function __construct(){
    $this->load->database();
  }

  public function count_papers($name){...}

  public function get_papers($name,$page=0){...}

  public function get_more_detail($paperid){...}

  public function get_paper_recommended($paperid){...}
```

We create one view for each of our web page, and since each view has several components, we put them in a folder correspondingly to make the whole project look compact.
The 'home' folder is for the page we enter first and do our search.
The 'main_page' folder is for showing the results we get after we have clicked 'search' button, 'author_brief.php' ,'conference_brief.php' and 'paper_brief.php' show the result of authors, conferences and papers related to the keyword respectively, while 'donate.php' show the donate page. The 'mainpage.php' is responsible to use javascript to show whatever results the user want to see. Finally, the 'buttons.php' show the buttons used to turn the pages. Since almost all the components need a button to control the page showed, it's better to separate the button component from them and load this component whenever other components are loaded.
The 'author_page', 'conference_page', 'paper_page' folder cope with the presentation of more details for a particular author,conference and paper respectively.
The 'error' folder is a default one of the CI frame and cope with the errors.

Controller. It's OK to have several controllers to control the project, but since our project is not quite a complex one, it's more compact to hold only one controller called 'Page'. The role of the controller is to cope with the requests of the views, get result from the models and finally controls what should be seen. Here's the graph of how our project runs. In this graph, all the pages is

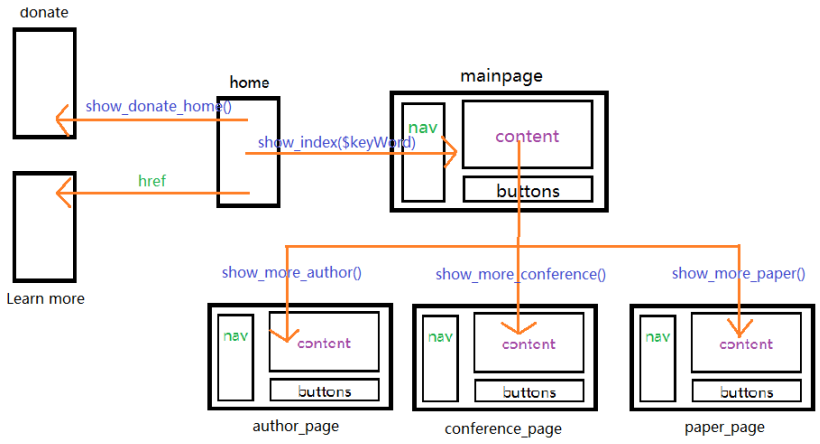


Figure 23: The flow chart of our project

represented by a rectangle and the orange line represents the possible flow. The blue texts represents a function of the same name in 'Page' controller, which loads the view we want.
There's also other functions in controller 'Page' to load different views in a same

page, which we will talk about later.

7.2.2 A concrete example of MVC separation

Since the mechanism of showing each web page and using the models to fetch the results from SQL is quite alike, I would like to take 'main_page' as example to show the specific mechanism.
There are five buttons on the sidebar of the page. In the original approach, each of which is linked with an independent PHP file to update the contents in the right part, while each PHP file should not only include how to present the content, but also how to connect the database and fetch the results we want. Now, with the help of CI, we are able to divide this process in two parts. Just

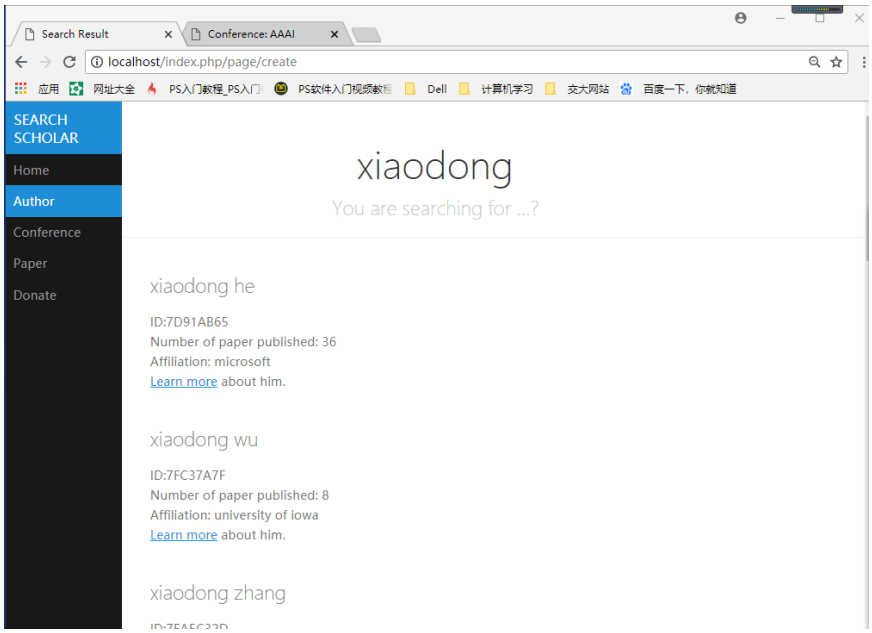


Figure 24: The presentation of mainpage

take what will happen after clicking button 'Author' as example. The javascript function is run after 'Author' button is clicked.

```
if(choice == 2){
$.post("/index.php/page/show_authors", {
pageIndex: pageIndex,
scholarName: keyWord
}, function (data) {
$("#results").html(data);
```

Compared to the original version, the only change in JS function is to post the request on a function called 'show_authors()' in controller 'Page' (This url is in the form of "http://localhost/index.php/['controller name']/[function

name]/parameters" and we have set our base URL as 'http://localhost/') Then, let we see how the function show_authors() works.

```
public function show_authors(){
$name = $this->input->post("scholarName");
$page=$this->input->post("pageIndex");
$data['pageIndex']=$page;
$page=$page-1;
$data['total']=$this->author_model->count_authors($name);
$data['authors'] = $this->author_model->get_authors($name,$page
    );
$data['title'] = 'authors';
$data['name']=$name;
$data['startPoint']=$page*10;

$this->load->view('main_page/author_brief',$data);
$this->load->view('main_page/buttons',$data);
}
```

We first set two variables 'name' and 'page' to get the parameters of the 'post' request, here we use CI syntax:

```
$variable = $this->input->post('parameter name');
```

to convey the value of the parameter.
The data array is the information we are going to convey to the views. We store the variables into data array by giving each of them a unique label:

```
$data['label']=$variable;
```

Notice data['total'] and data['authors'] is acquired by calling a function of author_model respectively. Of course, we should include the model in the controller:

```
public function __construct(){
parent::__construct();
$this->load->model('author_model');
$this->load->model('paper_model');
$this->load->model('conference_model');
$this->load->helper('url');
$this->load->helper('form');
$this->load->library('form_validation');
}
```

It's also important to show how these functions of author_model works. We get two variables in get_authors($name,$page=0) and use them to do the query in MySQL. Since we have already done the configuration at the very beginning, we don't need to make connection to the database anymore. As for the query command, we use it in CI manner,too:

```
$query = $this->db->query("SELECT AUTHORID FROM authors WHERE
    AUTHORNAME LIKE'%".$name."%'");
```

Then, we use an auxiliary function to get the result of query into an associative array:

```
$result=$query->result_array();
```

We also use another auxiliary function to get the first row of the results in an associative array:

```
$affiliationName = $query2->row_array()['AFFILIATIONNAME'];
```

Finally, using the syntax above, we can rewrite the commands into a CI form and return the data we want. What we return will be stored in a label in the 'data' array. In this case, we return an associative to $data['authors'] and a number to $data['total'].

```
    public function count_authors($name){

$query = $this->db->query("SELECT AUTHORID FROM authors WHERE
    AUTHORNAME LIKE'%".$name."%'");
return $query->num_rows();
}


public function get_authors($name,$page=0){
  $startPoint = $page* 10;
  $query = $this->db->query("SELECT AUTHORID FROM authors WHERE
       AUTHORNAME LIKE'%".$name."%'");
  if ($query->num_rows()!=0)
  {
    $authorIds = array();
    foreach ($query->result_array() as $row)
    {
      array_push($authorIds, "'".$row["AUTHORID"]."'");
    }
    $total = count($authorIds);
    $authorIds = join(",", $authorIds);
    $query = $this->db->query("SELECT
      paaffiliation.AUTHORID, authors.AUTHORNAME, COUNT(
          paaffiliation.AUTHORID)
      FROM
      academicrecord.paaffiliation
      JOIN
      academicrecord.authors ON authors.AUTHORID =
          paaffiliation.AUTHORID
      WHERE
      paaffiliation.AUTHORID IN ($authorIds)
      GROUP BY paaffiliation.AUTHORID
      ORDER BY COUNT(paaffiliation.AUTHORID) DESC LIMIT
          $startPoint, 10
    ");
    $result=$query->result_array();
    foreach ($result as &$row)
    {
      $authorId = $row["AUTHORID"];
      $query2 = $this->db->query("SELECT
      AFFILIATIONNAME
      FROM
      academicrecord.affiliations
      JOIN
      academicrecord.paaffiliation ON paaffiliation.
          AFFILIATIONID = affiliations.AFFILIATIONID
      WHERE
      paaffiliation.authorid = '".$authorId."'
```

```
        GROUP BY paaffiliation.AFFILIATIONID
        ORDER BY COUNT(paaffiliation.AFFILIATIONID) DESC
        ");
        $affiliationName = $query2->row_array()['AFFILIATIONNAME'];
        $row['AFFILIATIONNAME']=$affiliationName;
        }
      return $result;
    }
    else
    {
      $result=array();
      return $result;
    }
}
```

The last step is to present what we have acquired to the user, we build a new page called 'author_brief.php' in folder 'main_page'. We load a view of 'author_brief' in the controller and convey $data array to it. Since we only update the content of the right side of the web page, what should appear in 'author_brief' is a table containing results we want(The css styles are added in 'mainpage.php')

```
<table>
  <?php foreach ($authors as $a): ?>
  <tr>
    <?php echo "<div class='content'>
      <h2 class='content-subhead'>".$a['AUTHORNAME']."</h2>
      <p>
      ID:".$a['AUTHORID']."<br>
      Number of paper published: ".$a['COUNT(paaffiliation.AUTHORID)']."<br>
      Affiliation: ".$a['AFFILIATIONNAME']."<br>
      <a href=\"";
      echo site_url("page/show_more_author?authorId=".$a['AUTHORID']."&authorName=".urlencode($a['AUTHORNAME']))
      ."\" target=_blank style='color:#1f8dd6'>Learn more</a> about him.
      </p>
    </div>";?>
  </tr>
  <?php endforeach; ?>
</table>
```

We can just fetch the data by label in' author_brief.php'. Also, we load an additional view of buttons in the controller, this just equals putting the codes of the layout of the buttons below the codes of 'author_brief.php'. The benefit of this division is to let each view component have a more particular use and cut down the number of lines in a single file.

Form validation. It's not difficult to realize form validation using CI, thanks to a library named 'form_validation', we should load it first in the construct function of our controller.

```
$this->load->library('form_validation');
```

Then we should set some validating rules to our project. Since we match anything which 'like' what you enter. If nothing is entered, in our original program, a maximum running time error will occurred. So, we set a rule which requires the user to at least enter a letter to let the search run, or it will return an warning and doesn't do anything until the user enter something and click the 'search' button again.

```
$this->form_validation->set_rules('keyWord', 'KeyWord', '
    required');
```

In fact, there are plenty of rules toward what the user enters, just using the similar method. But since our original intention is to develop a searching project with less restrictions and let the user only enter once to get whatever result he wants, we are not going to impose other unnecessary rules. The flow of this page should be: return back to itself when form_validation is not passed, and turn to main_page if the form_validation is passed. So the realization of the function of home page should be:(run() is a auxiliary function of form_validation,judging whether the form validation is passed)

```
public function create(){
    $this->form_validation->set_rules('keyWord', 'KeyWord', '
        required');
    if ($this->form_validation->run() === FALSE){
    $this->load->view('home/home');
  }
  else{
    $keyWord=$this->input->post('keyWord');
    $data['keyWord']=$keyWord;
    $this->load->view('main_page/mainpage',$data);
  }
}
```

Finally, we add a division in our view to show the validation errors, since the default color is not conspicuous in our blue-dominant background, we change the color to be yellow.

```
<div style='color:yellow'><?php echo validation_errors(); ?></
    div>
```

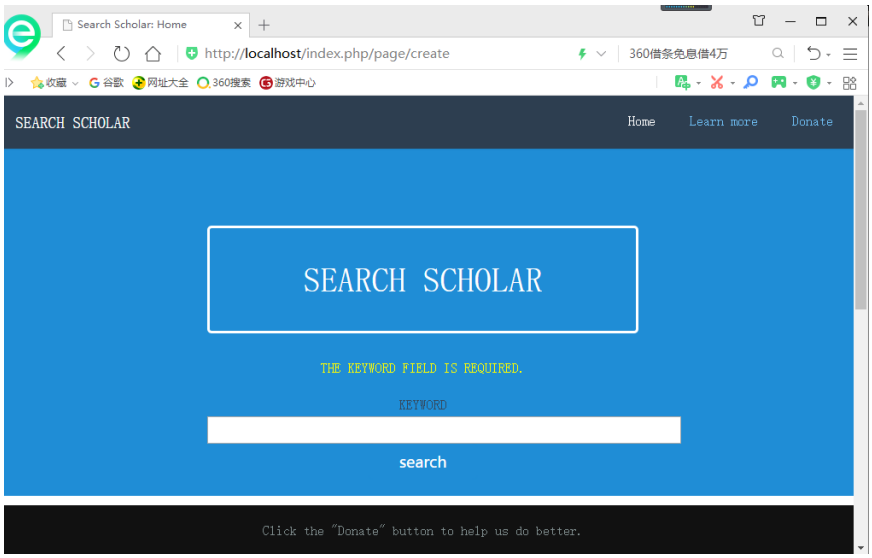
The final outlook should be like Fig 7.2.2

7.3 Small improvement on CI

It looks redundant to have our URL include 'index.php', so we want to eliminate it from the URL. We first open the 'conf/httpd.conf' of our apache and remove the '#' before:'LoadModule rewrite_module modules/mod_rewrite.so'. Then we replace 'AllowOverride None' relating to '.htaccess' with 'AllowOverride All'. Second, we build a '.htaccess' file at the root of our website with content:

```
RewriteEngine on
RewriteCond $1 !^(index\.php|images|resources|robots\.txt)
RewriteRule ^(.*)$ /index.php/$1 [L]
```

Third, we revise our 'config.php' in folder 'config', and change the line:




```
$config['index_page'] = "index.php";
```

Into:

```
$config['index_page'] = "";
```

Finally, we are able to enter 'localhost/page/create' to visit our project!

8 Some improvements on other parts

8.1 When no results are found

In the original program, we just use 'mysqli_fetch_assoc($query)' function to get the result we want, but this method returns an error on the view when no result is valid. Now we first judge whether a query returns no results in the model using auxiliary function 'num_rows()' , is no data is acquired in the query, 'num_rows' returns 0. In the model, we return the result_array of the $query when the number of rows not equals 0 and return an empty array when no data match the keyword we entered. We check whether this method take effect on our web page, and it works.

References

[1] https://purecss.io. Pure.CSS, a set of small, responsive CSS modules that you can use in every web project. Yahoo Inc., 2014.

[2] http://bl.ocks.org/d3noob/8375092, Interactive d3.js tree diagram. d3noob, 2018.

[3] https://bl.ocks.org/mbostock/4062045, Force-Directed Graph. Mike Bostock, 2018.