

Data Analytics Web Application Technical Report

Group 6: 309174902 | 490048621 | 490588998

Introduction

System introduction

This system is a typical three-tier web application mainly designed and implemented using the MEAN stack.

The data source is a data set that contains the revision history of featured Wikipedia articles from the English Wikipedia page[1], which contains many JSON format files. This system stores the data set in a MongoDB non-relational database. By querying and summarizing the information in the data set on the server-side, the summarized summary and statistical information results are returned to the front-end pages, and the concise and clear chart information is generated and displayed on the pages.

This webpage program can also interact with third-party websites through published web APIs. By invoking publicly available API endpoints, the application is able to fetch the latest dataset directly from Wikipedia as well as displaying the hottest news topics related to the selected article.

This web application has been deployed on the Heroku Cloud Application Platform, which can be opened directly via <https://com5347angulargourpsix.herokuapp.com/>.

Division of labor

Front-end development: Shanzheng Liu (490048621)

Back-end development: David Dai(309174902) & Qizhen Zhu (490588998)

Functional design

Function module introduction

User module

In this system, the user's login and password modification functions are provided. After providing the email and password, the user can successfully register an account. The user must log in to perform a series of operations on the website. The system also provides the option to reset the user's password based on the authentication answer. After completing the query access operation on the page, the user can safely log out of the account.

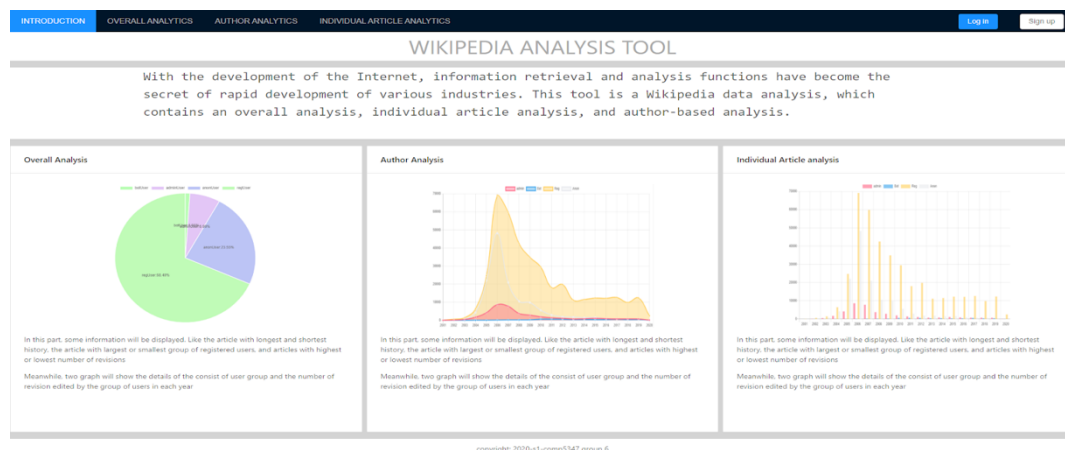


Figure1.1 Home page

Overall analytics module

In the article's overall analysis section, we have compiled the given data set to analyze it in an overall manner and extract constructive information, such as "the first two articles with the most revisions and the number of revisions." And the information is clearly displayed in the form of text.

In addition, by reading administrators.txt and bots.txt, we analyzed the types of users who modified the article, displayed four different types of user information in the form of pie charts, and added some web page interaction functions.

This module also provides a bar chart to show the relationship between the revision year and the user type, and can also switch between the line chart and the bar chart.

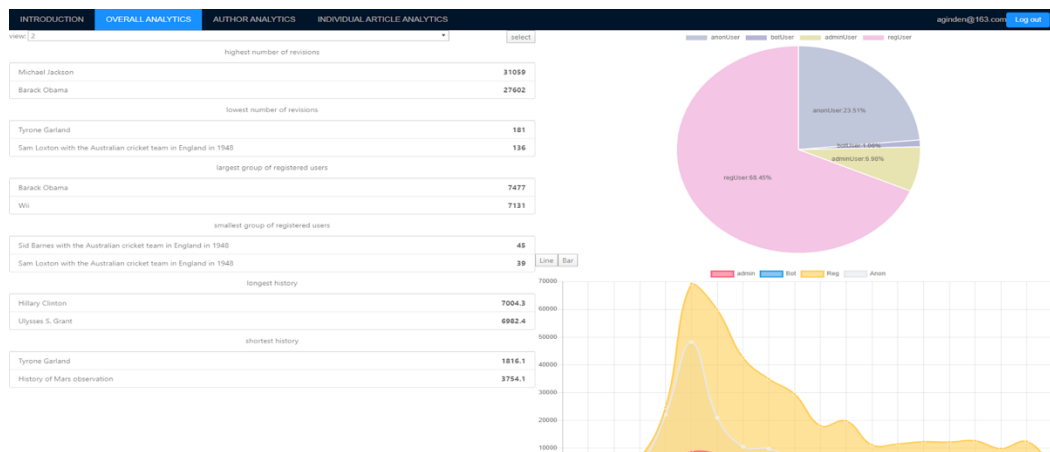


Figure1.2 Overall analytics page

Individual article analytics module

In this module, we conduct specific analysis of individual articles based on the articles selected by the user. We first check whether the history of the article is up-to-date, if not, update it through the MediaWiki API and pop up a message. In addition, we display and analyze some basic information for selected articles, such as headlines and related news from third parties. We also provide three charts. When the user selects a specific range through the year filter, the chart can analyze the relationship between the year, user type, and user in this article according to the range selected by the user.

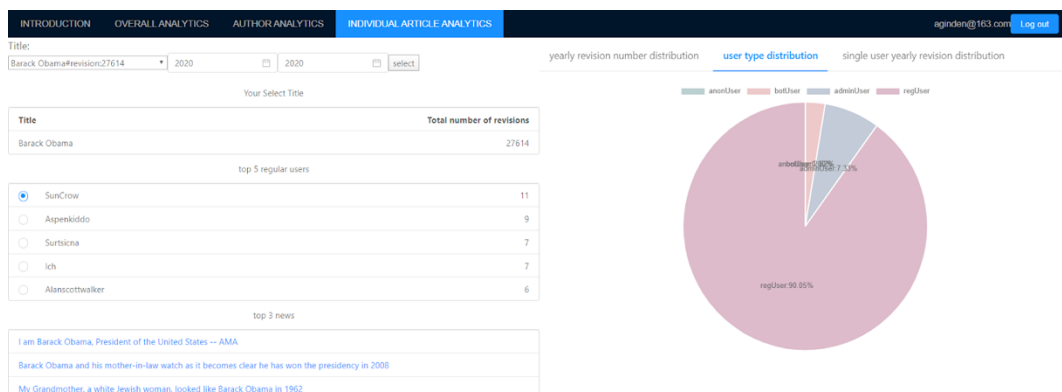


Figure1.3 Individual article analytics page

Author analytics module

In this module, users are able to search the author they interested in. After they input some letters into the input box, the autocomplete component will show the names relating to the keywords, and users can select the name directly from the name list. After pressing the search button, the article edited by the author you searched and the revision number will be shown in the article list. In the next step, users can select the article shown in the article list and click the button “select”. The timestamp of the revision record relating to the author and article will be shown in the timestamp list.

Figure1.4 Author analytics page

MVC Architecture Design

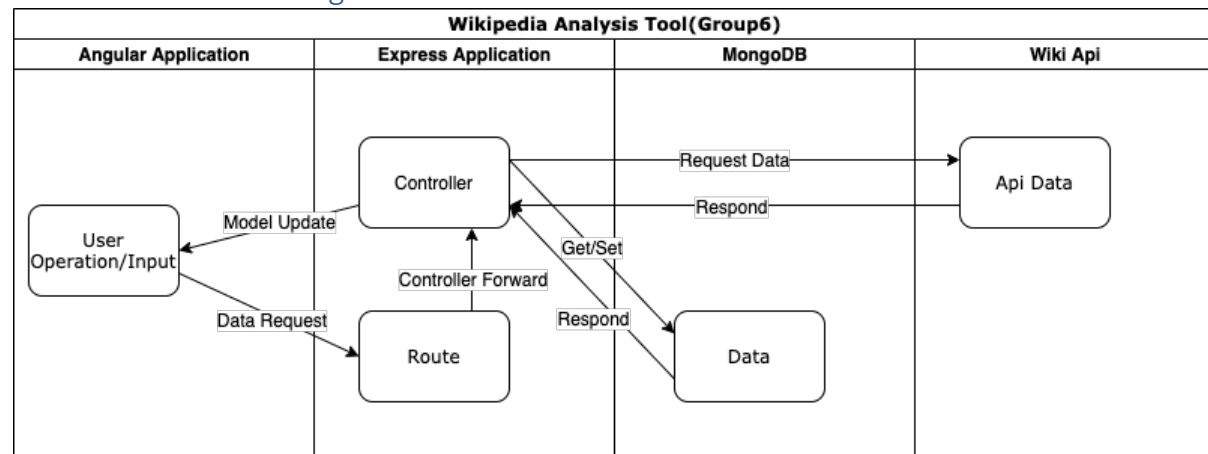


Figure2.1 The architecture of the application

The process of development in this application has been divided into two parts including front end and back end. The MVC design has been used in these two sub-projects. The Angular Application represents the “View”. When the user operates the application, angular will make requests to the Express Application for the update of the “Model” Data. Based on the route, the express application uses different “Controllers” to manage various requests including the data setting and getting in MongoDB. If the data relates to the WIKI API, the express application will make a request to WIKI and update the data in MongoDB to be the newest. Finally, these data are displayed on the front-end page.

Database design

Database structure

revid	Number
parentid	Number
minor	Boolean
user	String
anon	Boolean
userid	Number
timestamp	String
size	Number
sha1	String
parsedcomment	String
title	String
sha1hidden	Boolean
commenthidden	Boolean

Figure2.1 Collection revinfos structure

firstname	String
lastname	String
email	String
password	String
question	String
answer	String

Figure2.2 Collection users structure

Database installation instructions

Data import script install

For data importing, a python library script has been deployed in PiPy. Before using this script, python >=3.6 should be installed on your computer.

1: Open the command in Windows or terminal in Mac OS and input the command

```
pip install MongoFileImport
```

2: Find out your MongoDB URL and transfer it into python format. It looks like

```
mongodb+srv://username:password@cluster0-xxxxx.mongodb.net
```

3: Find out your name of the database and the name of an **existing** collection.

4: Get the absolute path of your data file which only contains JSON files.

5: Open your command or terminal again, and input the command like:

```
MongoFileImport --mongourl mongodb+srv://username:password@cluster0-xxxxx.mongodb.net --db test --dir d:/Downloads/Dataset_22_March_2020/Dataset_22_March_2020/revisions/ --cn collectionname
```

for this command, "--mongourl" refers to your MongoDB URL, "--db" refers to the name of the database. "--cn" refers to the name of the collection and the "--dir" means your path of the folder which you want to import.

In this project, the name of the database should be "node-angular"; and the name of the revision data collection should be "revinfos" and a new collection name "users" should be created. If you import your data in local database, the import command should be

```
MongoFileImport --mongourl mongodb://username:password@localhost --db node-angular --dir yourDataFolderPath --cn revinfos
```

If you have any questions, you can visit <https://github.com/shanzhengliu/MonogdbFileImport> to get more details and star it.

Frontend project install

- 1: use Command or Terminal to get into the root path of the frontend project “web-angular”.
- 2: run command “npm install” to install the dependencies and library for the project.
- 3: run command “npm start”, after compiling and rendering, the project will start.

```
D:\COMP5347Group6\web-angular>npm start
> web-angular@0.0.0 start D:\COMP5347Group6\web-angular
> ng serve

chunk (main) main.js, main.js.map (main) 571 kB [initial] [rendered]
chunk (polyfills) polyfills.js, polyfills.js.map (polyfills) 150 kB [initial] [rendered]
chunk (runtime) runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk (styles) styles.js, styles.js.map (styles) 2.09 MB [initial] [rendered]
chunk (vendor) vendor.js, vendor.js.map (vendor) 10 MB [initial] [rendered]
Date: 2020-05-12T09:48:35.683Z - Hash: 7149beabddf7332ca109 - Time: 18275ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
i Compiled successfully.
```

Figure2.3 Frontend project install

Backend project install

- 1: use Command or Terminal to get into the root path of the backend project “group-back”.
- 2: run command “npm install” to install the dependencies and library for the project.
- 3: run command “npm run start:server”, and the express server will start.

```
D:\COMP5347Group6\group-back>npm run start:server
> group-back@0.0.0 start:server D:\COMP5347Group6\group-back
> nodemon server.js

[nodemon] 2.0.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting node server.js
(node:12468) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
(node:12468) DeprecationWarning: current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new Server Discovery and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
(node:12468) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
Connected to mongoDB Atlas Database!
```

Figure2.4 Backend project install

Functional structure diagram

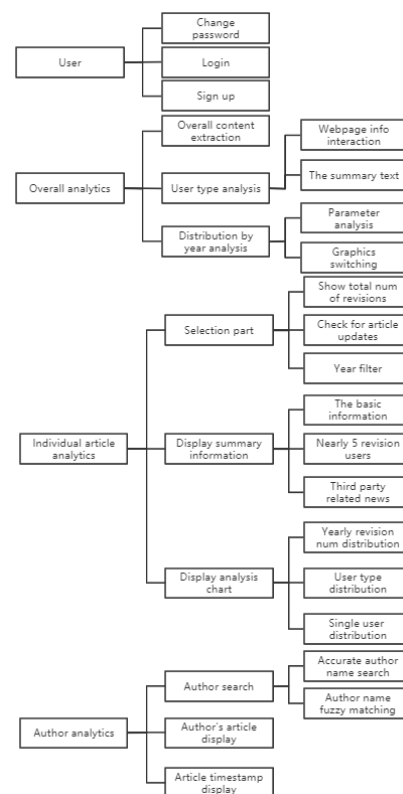


Figure3.1 Overall functional analysis

Nonfunctional design

Performance

Response time

In 90% cases, the response time of the general page was less than 3 seconds, among which the Overall analytics page had a response time of about 10 seconds due to the large amount of data.

The login response time was within 2 seconds, the response time of the refresh column was within 2 seconds, the response time of the refresh item pagination list was within 2 seconds, and the response time was within 5 seconds after the number of articles was selected to display in the Overall analytics page.

Precise searches by name during off-peak hours yield results in less than three seconds. A fuzzy search based on the first few letters of the author's name can get a match within 3 seconds.

System capacity

Because MongoDB belongs to NoSQL, the data is stored in the form of documents, with a maximum value of 16MB. The maximum database capacity should not exceed 500GB, and the disk space should be at least 10GB. Support for GB level data.

Security

The system strictly controls access rights. Before the user logs in, cannot view the system function and with the data chart to carry on the interaction, can only view the system introduction which the home page displays. After a user logs in, he or she can operate within the scope of his or her authority.

The system provides operation log management, which can track the historical usage of the system. The following code snippet shows the running logic of the system to distinguish whether a user is logged in or not.

```
async loginStatus(User: []): Promise<boolean> {
  const jsonStatus = await this.postData('http://127.0.0.1:3000/api/user/login',
User);
  if (await jsonStatus.confirmation === 'success') {
    console.log(jsonStatus);
    await this.ls.setObject('isLogin', true);
    await this.ls.setObject('username', jsonStatus.user );
    await this.ls.setObject('token', jsonStatus.jst);
    return true;
  } else {
    await this.ls.remove('isLogin');
    return false;
  }
}
```


Reliability

The system has been placed on the Heroku Cloud Application Platform, and the system can operate for 7x24 hours. The total time of failure and shutdown in the continuous operation throughout the year will not exceed 7 days.

The robustness of the system is strong, and it can deal with many kinds of abnormal situations in the process of system operation. For example, when the user logs in and registers, the system checks

the data entered by the user and provides a prompt to prevent the data abnormal caused by illegal data input by the user.

Log in



Please input your E-mail address to log in!

☒ Remember me

[>Forgot password](#)

Figure3.1 The system handles exception input

```
src > app > view > users > login > login.component.html server.js
login.component.html x
11 <div style="..."></div>
12 <div class="login-page">
13 <form nz-form [formGroup]="loginForm" class="login-form" (ngSubmit)="onLogin()">
14 <nz-form-item>
15 <nz-form-control nzErrorTip="Please input your E-mail address to log in!">
16 <nz-input-group nzPrefixIcon="user">
17 <input type="email" nz-input formControlName="userEmail" placeholder="Useremail">
18 </nz-input-group>
19 </nz-form-control>
20 </nz-form-item>
21 <nz-form-item>
22 <nz-form-control nzErrorTip="Please input your Password!">
23 <nz-input-group nzPrefixIcon="lock">
24 <input type="password" nz-input formControlName="password" placeholder="Password">
25 </nz-input-group>
26 </nz-form-control>
27 </nz-form-item>
```

Figure3.2 Code implementation

Compatibility

Our system supports both Mac OS and Windows, and we have a way of dealing with different operating systems.

For example, when analyzing the user types of the revision article, our system needs to read the local files to differentiate the different user types, and the different operating systems cause problems with file string handling. So we did some analysis and code handling of the problem.

The following code is taken from individual.controller.js to solve the file reading problem on different platforms.

```
if(os.type().toString().toUpperCase().indexOf("WINDOWS")!==-1) {
    botUser = fs.readFileSync(path.join(__dirname,
'../../user_filter/bots.txt')).toString().split("\r\n");
    adminUser = fs.readFileSync(path.join(__dirname,
'../../user_filter/administrators.txt')).toString().split("\r\n");
    userAdminBot = botUser.concat(adminUser).sort();
}
else
{
    botUser = fs.readFileSync(path.join(__dirname,
'../../user_filter/bots.txt')).toString().split("\n");
    adminUser = fs.readFileSync(path.join(__dirname,
'../../user_filter/administrators.txt')).toString().split("\n");
    userAdminBot = botUser.concat(adminUser).sort();
}
```

Ref.

[1]Featured articles. (2020). Retrieved 16 May 2020, from https://en.wikipedia.org/wiki/Wikipedia:Featured_articles