Data analytic web application

Assignment two

University of Sydney

COMP 5347: Web Application Development

Contents

[1 Application Architecture 2](#_Toc483837954)

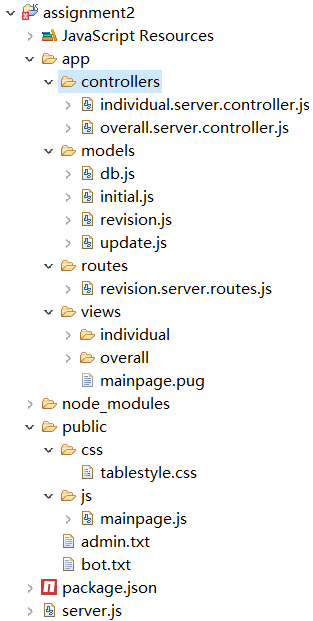
[2 Client Architecture 2](#_Toc483837955)

[2.1 Overall Level Status 2](#_Toc483837956)

[2.2 Individual Level Status 3](#_Toc483837957)

[3 Backend Architecture 3](#_Toc483837958)

# Application Architecture

****This application is designed using the MVC framework. The structure of the application is shown in Figure 1. “server.js” is the main entrance of the application, it will assign “revision.server.router.js” under the path “app/routes” as the route. When receiving request from client, the route will invoke different functions in different controllers located under the folder “app/controllers”, then controller functions will call different models (“app/models”) to handle the requests, finally render in “views/mainpage.pug”.

# Client Architecture

On the top of the page, is a dropdown selector, which will send request to back-end to collect the most recent count of revision for each article and the overall amount. Below the selector, is a table displaying overall status or top users based on selected article, the two table are built from backend based on the “app/views/individual/table.pug” and “app/views/individual/individualTable.pug” respectively. Down from the table, there are three buttons which can switch from three different visualizations. All

Figure 1 Structure

the constructing data are retrieved from back-end asynchronously when the page finished loading.

## Overall Level Status

Overall status can be accessed by default URL “localhost:3000” or navigating with selector settled to “Overall”. Four requests are send to back-end asynchronously:

1. Get selector. The selector is in duty of navigating to any individual article or overall status. Request is sent by URL: “index/getSelecter”, callback function will construct the select line by line with responded json.
2. Get overall table. The table displays the overall status for all 6-required information. Request is sent by URL: “index/overall/getTable”, the callback function will append the rendered html to a div with id “select”
3. Get bar chart. The bar chart shows the revision distribution by year and by user type. Request is sent by URL: “/index/overall/getBar”, results are rendered by using google static API after receive json file from back-end.
4. Get pie chart. The pie chart shows the overall distribution of different user types. Request is sent by URL: “/index/overall/getPie”, results are rendered by using google static API after receive json file from back-end.

## Individual Level Status

When an article in the dropdown box is selected, the application will firstly send updating request to back-end, when finished updating, the amount of records updated in mongodb will return to the front-end within a json object. After confirmed an alert which showing the number of documents updated in back-end, the client will request for necessary information from backend, which is similar with functions in the overall phase.

# Backend Architecture

When starting the application with “server.js”, “initial.js” which located in “app/models” will be executed. This model will read in bot and admin information, then assign every document in mongodb with a new attribute “userType”, the process will take some time for execution in the first time, but it saves huge processing time in later use.

After initialising complete, two message “bots & admins signed complete” and “anon signed complete” will show up in console log. Then the server side is ready to receive request from front-end.

The backend design follows the MVC framework:

|  |  |  |
| --- | --- | --- |
| Component | File Name | Responsibility |
| Route | revision.server.routes.js | Mapping request from client URL and controller functions. |
| Controllers | overall.server.controller.js | Handling request for overall status, including all information needed for table and charts, and request for constructing dropdown selector.  The responding of selector, two chart and table are all done asynchronously in each call-back function.  The table is responded using “async” lib, to make sure they have all finished call-backs. |
| individual.server.controller.js | Handling request for individual status, including all information needed for table and charts, and pulling update from Wikipedia API.  When request for update local database is invoked, the query for local latest revision happened first, then compare it with current date, if it is edited within one day, the application will do nothing, otherwise its call-back function it will query Wikipedia API to pull out all latest revisions, then using “insertmany()” function (supported by mongoose 4.3.0 and later version) to perform batch insert.  The responding of three chart and table are all done asynchronously in each call-back function. |
| Models | db.js | Connecting to mongoDB |
|  | revision.js | Responding for all query to mongoDB |
|  | initial.js | Read in bot and admin information, then assign every document in mongodb with a new attribute “userType” |
|  | update.js | Actual handler for querying from Wikipedia for new data, and insert them to mongoDB. |