Technical Report - Group 22

1. Introduction

This report will outline the design and architecture of our data analytic web application. The design follows a three-tier web application (MVC pattern) using MongoDB, Express.js, React and Node.js. JavaScript is used for both the frontend and backend. There are 4 main states of this application, the login/landing page, overall article analytics, individual article analytics and author analytics. The last three states are implemented within a single page, following SPA principle. As well as meeting the functional requirements of the brief, we've also considered non-functional requirements such as security, performance and usability to enhance our application.

The GitHub for our source code can be found here:
https://github.sydney.edu.au/rche4499/comp5347

2. Design and Architecture

Model uses Mongoose, View uses React and Controller uses Express.js framework. Frontend React application runs on localhost3000 and backend express framework runs on localhost5000. Communication between frontend and backend components happens through REST API calls, for example, from Overall Articles:

Front-End

```
// Retrieve list from Express App
useEffect(() => {
    // Overall: Data
    fetch('/api/topArticleRevisions/?topcount=' + selectedNumber).then(res => res.json()).then(list => setTopRevisions(list));
    fetch('/api/lowestArticleRevisions/?topcount=' + selectedNumber).then(res => res.json()).then(list => setLowestRevisions(list));
    fetch('/api/largestArticleGroup/?topcount=' + selectedNumber).then(res => res.json()).then(list => setLargestGroup(list));
    fetch('/api/smallestArticleGroup/?topcount=' + selectedNumber).then(res => res.json()).then(list => setSmallestGroup(list));
    fetch('/api/longesArticletHistory/?topcount=' + selectedNumber).then(res => res.json()).then(list => setLongestHistory(list));
    fetch('/api/shortestArticleHistory/?topcount=' + selectedNumber).then(res => res.json()).then(list => setShortestHistory(list));
}, [selectedNumber])
```

Back-End

```
//Overall: Data
router.get('/api/topArticleRevisions', controller.getTopArticleRevisions);
router.get('/api/lowestArticleRevisions', controller.getLowestArticleRevisions);
router.get('/api/largestArticleGroup', controller.getLargestArticleGroup);
router.get('/api/smallestArticleGroup', controller.getSmallestArticleGroup);
router.get('/api/longesArticletHistory', controller.getLongestHistory);
router.get('/api/shortestArticleHistory', controller.getShortestHistory);
```

External packages used to assist in UI and creating charts include Atlaskit, material-ui and react-chartjs-2.

Sorting Users

Sorting user type “Administrator” and “Bot”
With the text files (administrators.txt and bots.txt) given, string values were saved into an array containing user and user type. Depending on the file name, all the displayed users were categorized as ‘admin’ or ‘bot’ in the database.

Sorting user type “Anonymous” and “Registered”
On the other hand, if the users were not categorized as either ‘admin’ or ‘bot’, the data value of ‘anon’ had to be checked. By checking all the revisions in the database with ‘usertype’ set as false, users were able to be categorised by checking the ‘anon’ value. If ‘anon’ value does exist, the user is ‘anonymous’ user, whereas, if ‘anon’ value doesn’t exist, the user is ‘registered’ user.

2.1 Login/Landing Page

Model (/app/models/user.server.model.js)

The model ‘user.server.model.js’ includes 8 static functions which are used for queries relating to the MongoDB database. It handles the creation and manipulation of user data, which is stored in a JSON format. The password hashing function ‘bcrypt’ is used within Model to implement secure hashing of discreet data.

View (src/landingpage)

The View of the Login and Landing pages includes a ‘Login’ and ‘Register’ options, with a ‘Reset Password’ embedded within the login option. These options allow users to either login with their credentials, reset their password with answers to a security question, or register as a new user with various required input of their details.
Main functionalities in Login/Landing Page:

Login: Only upon successful login, the user is able to access the analytics functionality. The view provides a combination of buttons, input text and transitions for users to appropriately interact with. This includes a ‘POST’ form request which is handled by the User Controller.

Reset Password: The reset password which is found within the login option allows users to reset their password after answering their previously set questions. The users have to fill out a ‘POST’ form which thus is handled by the User Controller and is furthered to change the data in the database.

Register: Registering new users is also done through a ‘POST’ form submission, where all the details required for creating a new user’s JSON details are created. This is done through the incorporation of various text inputs and validity checks.

Controller (app/controllers/users.server.controller.js)

The controller checks for authentication of users, determining whether users are allowed access to the main analytics or not. It also includes handling of users logging in, logging out, registering as a new user, and various error handling. It utilises the Model of User to be able to communicate changes required by the view, allowing changes or retrievals to and from the database.

2.2 Overall Article Analytics

Model (/app/models/article.js)

The model is in article.js, containing 8 static functions for querying the MongoDB database. In article.js, each schema maps to a MongoDB collection and it is defined with title, timestamp, user, anon and userstyle properties which are defined in the static functions.

Controller (app/controllers/analytics.server.controller.js)

There are 8 functions in the controller that can be accessed through the APIs defined in analytics.server.controller.js.

View (src/mainPage/overallArticles)

Main functionalities in overall articles:

Number of articles to search: Populates the user select option with a list of numbers between 1 to 8. Once the user selects the number, the number of articles from all categories changes correspondingly. As a default, two articles are displayed when accessing the page.

Top 2 articles with the highest/lowest number of revisions: Populates the result of articles with information of its title and the number of revisions. For the highest number of revisions, it fetches the first 2 (changed when selecting different number of articles to search) articles sorted in descending order according to number of revisions. Whereas, the lowest number of revisions fetches the first 2 articles sorted in ascending order.

Top 2 articles edited by largest/smallest group of registered users: Populates the result of articles with information of its article ID and number of groups. For the largest group, it fetches the first 2 (changed when selecting different number of articles to search) articles sorted in descending order according to number of groups. Whereas, the smallest group fetches the first 2 articles sorted in ascending order.

Top 2 articles with longest/shortest history: Populates the result of articles with information of its article ID and age. For the longest history, it fetches the first 2 (changed when selecting different number of articles to search) articles sorted in descending order according to its age. Whereas, the shortest history fetches the first 2 articles sorted in ascending order.

Bar Chart: Generates a bar chart of revision numbers distributed by year and user types across the whole dataset. From API call to the controller, receives a dataset containing the number of users in each year arranged by their user type. It is then populated into a package 'react-chartjs-2' to generate a bar chart.
Pie Chart Generates a pie chart of revision numbers and the percentage (displayed when hovering over the pie chart) distributed by user types across the whole dataset. It follows a similar API call with the bar chart but information only contains the total number for each user type. It is then populated into package 'react-chartjs-2' to generate a pie chart.

2.3 Individual Article Analytics

Model (/app/models/article.js)

The model is in article.js, containing 10 static functions for querying MongoDB database.

Controller (app/controllers/individualArticles.server.controller.js)

There are 9 functions in the controller that can be accessed through the APIs defined in individualArticles.server.routes.js.

View (src/mainPage/individualArticles)

Going through the main functionality:

Article Select: populates the user select option with a list of all articles that contain revisions in the database. Once an article has been selected by the user, the next features are displayed

From and to year select: populates both select options with years ranging from the year of the first revision for the article to the year of the last revision for the article currently in the database. A hook is used to update this every time a user changes the article selected. Thus the years are not hardcoded but vary according to each article. If the 'to' year selected is before the 'from' year selected, then an error message will be displayed notifying the user that the input is invalid.

Top 5 regular users: populates a table with the top 5 regular users within the time period specified by the user. A hook is used to update this every time either the article selected changes or the date range requested.

Reddit API: snoowrap, a JavaScript wrapper for the Reddit API is used to obtain the top news for a given article title. The top three news articles are obtained and displayed in a list.

Bar Chart - Distribution of Revisions and Pie Chart - Distribution by User Type
A dropdown is available for the user to select which chart he/she wants to view.

To populate these charts, both are one API call to the controller to get the data and populate it into the bar chart/pie chart in the package react-chartjs-2.

Bar Chart - Distributed by Top 5 Users
A radio select is provided for the end-user to choose which user they want to see the analytics for. Based on the user selected and the timeframe, an API call is made to obtain the data to populate the bar chart with the number of revisions of the chosen user between the specified time frame.

2.4 Author Analytics

Model (/app/models/article.js)

There are 3 static functions for querying the MongoDB database relating to Author Analytics.

Controller (app/controllers/authorAnalytics.server.controller.js)

Each of the 3 functions defined in the model can be accessed through the controller through an API. These functions provide access to author usernames and timestamps of their articles.

View (src/mainPage/authorAnalytics)

Author Autocomplete: Displays all authors that have a user type of either ‘admin’ or ‘bot’. The autocomplete feature allows text input which narrows down the list of available authors to be searched for. A query to the database retrieves the list of authors which are populated into the options of the Autocomplete search.

List of Articles: Displays all articles that have a revision made by the selected author. This includes the article title and the number of revisions made specifically by that author. The list of articles are retrieved through an API call to get the list of articles written/modified by the selected author.

Revision Timestamp Popup: Upon clicking the specific article title, a number of timestamps of revisions are shown in a Modal Dialog popup. The timestamps for the article selected are retrieved through an API call. This corresponds to the number of revisions made by the author to that article.

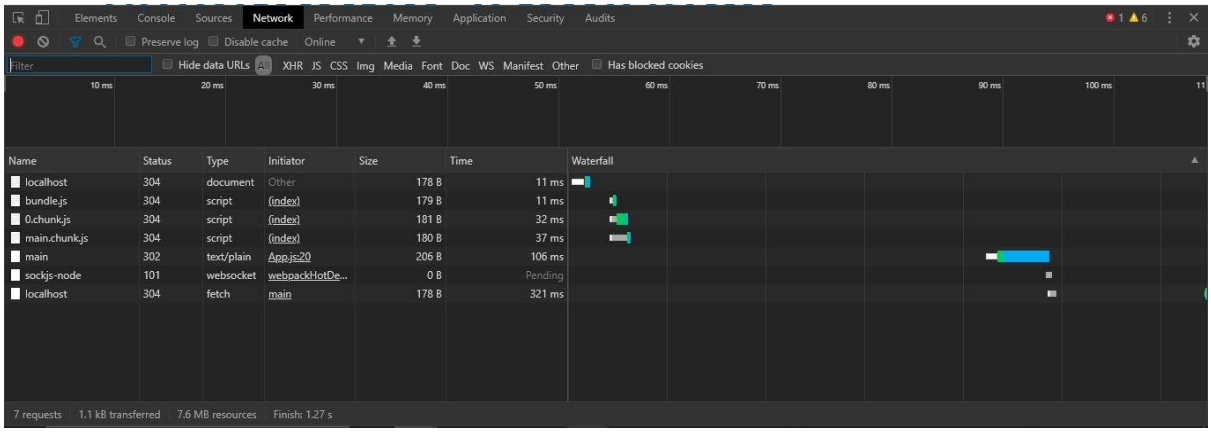
3. Non-Functional Requirements

3.1 Performance

React (Front-end framework) is a fast user interface without significant amount of work to specifically optimise for performance. React maintains a virtual DOM in JavaScript providing fast operation for determining menial changes needed to bring the documents to the desired state. Although it does not optimise the best performance in comparison to other front-end

frameworks, many performance optimisation techniques are available as React Apps for improvements. Also, our system included performance issues when retrieving lists from the back-end through hooks (‘useEffect’) which only re-rendered components when variables have changed. This lowered the consistency of data updates that had to be manually updated by changing the values.

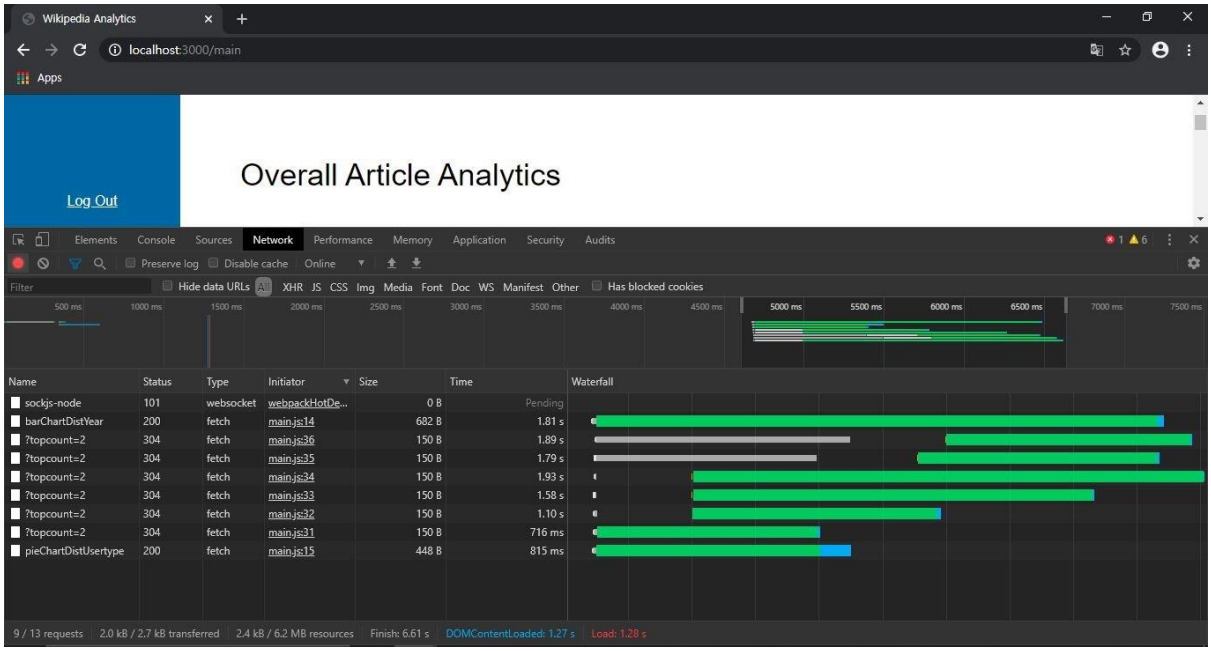
Browser Performance for Landing page (User login and register):



The landing page is quickly generated as it does not require any data call and everything is completed within 350ms.

Browser Performance for Main Page:

<u>Overall Articles:</u>

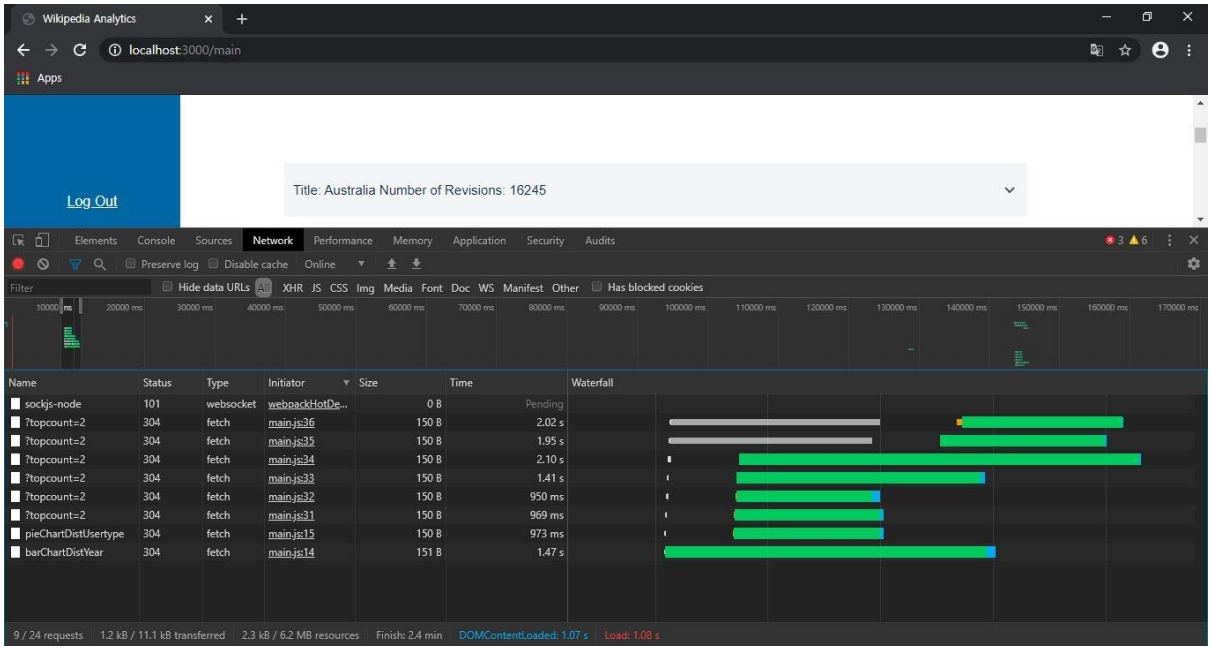


‘?topcount=2’ are the functionalities populating two articles for each category and ‘barChartDistYear’ and ‘pieCharDistUserType’ are the charts generated in overall article

analytics. As seen from the screen shot, the charts have larger size compared to other functionalities as it contains information of the entire dataset. The functionalities take approximately one to two seconds to generate which retrieving the article information and organising them into correct format takes around one second to accomplish.

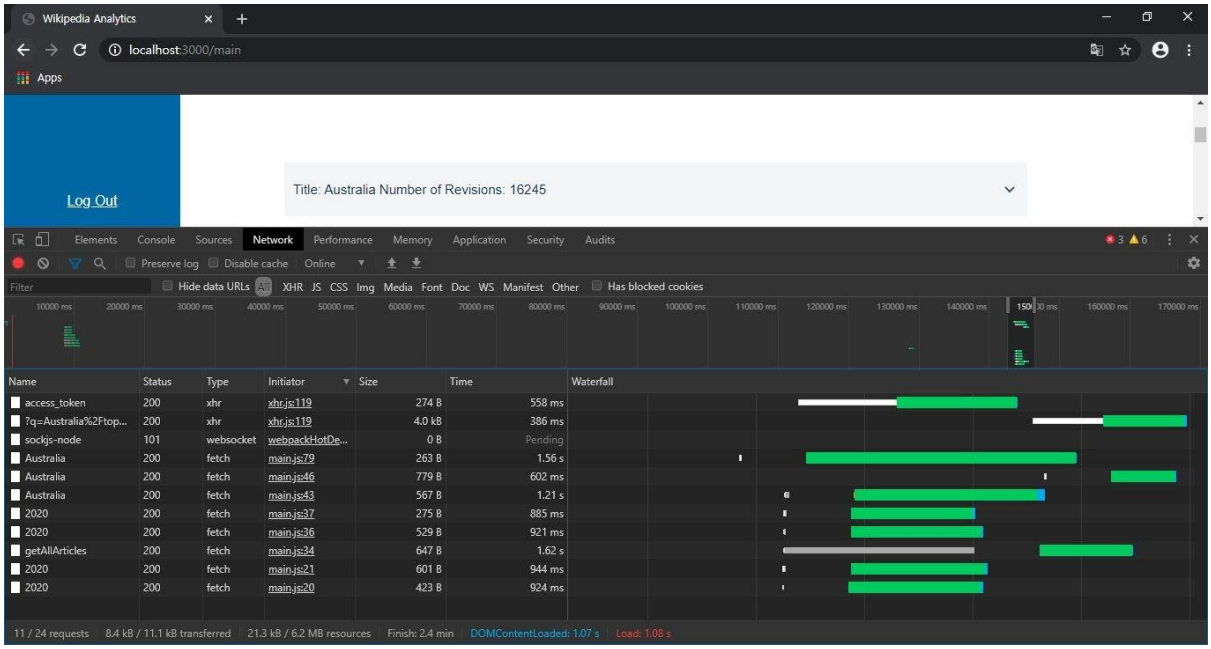
Individual Articles:

Selecting Individual articles:



Similar to the overall article analytics, retrieving the article information takes approximately half of the overall time for the page to generate.

Selecting an article:



Functionalities tend to take a little longer than the overall article analytics (when including the time for individual articles analytics page to open) as title information needs to be sent to the

backend to operate the sorting system which is received back to the frontend to be displayed.

3.2 Security

Hashing passwords

Passwords are entered by users in the UI and then are stored in the database. However, to ensure security and privacy of sensitive data, the hashing function ‘brcypt’ is used. Passwords are hashed with an added salt in order to force their uniqueness and increase the complexity of hashing. This added salt does not required any difference in user input and reduces opportunities of rainbow table attacks.

3.3 Usability

The concept of a single page application means that the user can interact with all the data analytic elements under the same URL (/main). One of the benefits of a single page application is usability, as the user doesn’t have to wait for a page to reload to interact with other features. User design has been considered when building the application and we’ve tried to make all the features intuitive and easy to use for the end-user.

4. Conclusion

In conclusion, in this report we have outlined the details of our data analytics web application by discussing how we’ve met the functional requirements requested. By adopting the MVC architecture, using a React frontend and Express.js backend, the final deliverable is a web application programmed entirely in JavaScript. Moreover, we’ve also considered non-functional requirements in the design of our web application.