DATAFLEXOR: AN EFFICIENT DATA ANAGEMENT APPLICATION USING FIREBASE AND ADVANCED AI MODELS

Weichuan Chen[1], Yu Sun[2]

[1]Taipei European School, No 31, JianYe Road, ShiLin District,
Taipei City, Taiwan
[2]Computer Science Department, California State Polytechnic University,
Pomona, CA 91768

ABSTRACT

This paper presents the development and evaluation of Dataflexor, a streamlined data management application designed to enhance efficiency in professional environments [1]. The application integrates Firebase services for user authentication and real-time data synchronization, as well as advanced language models like ChatGPT and Gemini for enhanced functionality [2]. Several challenges were addressed during development, including API integration, latency issues, and user data privacy [3]. Experiments were conducted to evaluate the performance of Firebase Cloud Functions and the response times of integrated APIs under varying traffic conditions [4]. The results revealed that while the application performs well under normal usage, significant performance drops occur during peak loads, indicating areas for further optimization. The study concludes that, with improvements in customization options, workflow optimization, and backend scalability, Dataflexor has the potential to become a powerful tool for professionals, offering both efficiency and flexibility in data management tasks [5].

KEYWORDS

Data Management, AI Integration, Cloud Functions, Real-Time Data Synchronization, Workflow Optimization

1. INTRODUCTION

The traditional office job in the United States starts at 9 AM and ends at 5 PM, representing 8-hour workdays for 5 days a week. According to Apollo Technical, an American employment agency, workers are productive for an average of 2 hours and 23 minutes during the standard 8-hour workday. Much time and money is being wasted every day because companies often require these workers to stay clocked in at the job site until 5 PM. This additional time that workers are forced to spend at the office takes away from personal leisure and recreational time, which can make their lives stressful, hurting their mental health and the happiness of the workforce.

The methodologies explored in this research compare various approaches to real-time data synchronization and data management using Firebase. The study by Chang and Myers (2020) introduced a tool for simplifying hierarchical data management in spreadsheets, which significantly reduced cognitive load and task completion time, though it was limited in scope to specific use cases. Similarly, the study by Ijsrem Journal (2023) optimized Firebase's synchronization techniques for mobile app development, providing crucial insights but remaining

David C. Wyld et al. (Eds): NWCOM, ASOFT, ITCCMA, CSITY, BDIoT, MLNLP, SIGPRO, AIFZ– 2024
pp. 77-85, 2024. - CS & IT - CSCP 2024 DOI: 10.5121/csit.2024.141907

confined to mobile applications. Li et al. (2018) further expanded Firebase's use in the "JustIoT" system for IoT device management, but it lacked flexibility in broader data management tasks [6]. In contrast, Dataflexor builds on these foundations by integrating Firebase's real-time capabilities into a versatile platform that supports a broader range of business applications, offering a more comprehensive and adaptable solution that addresses the limitations of the previous methodologies.

Dataflexor is my idea to help save time. I find that existing generalized applications like Excel and Google Sheets are very complicated, with a hundred different choices for customization and functions, it is often overwhelming. However, by simplifying spreadsheet applications, I mean to make my app easier to learn, more intuitive for new workers to save time from training and more business-specialized. Dataflexor is meant to improve the efficiency of the workers using it by catering to the needs of the business industry specifically by including pre-configured functions and automated data processing through the use of an AI API that will help assist the user by analyzing the spreadsheet and recommending changes in accordance with the context of the document [7].

In the experiments conducted, the primary focus was on identifying potential performance bottlenecks and ensuring the reliability of both Firebase Cloud Functions and the integrated APIs (such as ChatGPT or Gemini) under varying load conditions. The first experiment tested the latency of Firebase Cloud Functions by simulating high traffic scenarios. The findings indicated that while the system performed well under normal conditions, significant latency increases were observed during peak loads, highlighting the need for further optimization. The second experiment assessed the response times of the integrated APIs when subjected to varying levels of user activity. Similar to the first experiment, the APIs maintained efficient response times under moderate traffic but experienced noticeable delays during peak usage. These results underscore the importance of implementing load balancing, caching strategies, and possibly scaling backend resources to ensure consistent performance and a seamless user experience under all traffic conditions.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Compatibility Challenges

Integrating advanced language models like ChatGPT or Gemini into our system presents compatibility challenges, especially when combined with Firebase for backend services. Ensuring seamless communication between our application, Firebase, and the API requires careful handling of various data formats and protocols. Potential issues include mismatched data structures, unexpected responses, and integration complexities with Firebase's real-time database. To address these, I could implement robust input and output validation mechanisms to ensure data consistency and use middleware to format requests and responses. Thorough testing across different scenarios could help identify and rectify compatibility issues early in the development process.

2.2.Exceeding API Rate Limit

Relying on third-party APIs and Firebase introduces concerns regarding rate limits, latency, and overall system performance during peak usage. Exceeding API rate limits or experiencing high latency could degrade user experience, while Firebase's real-time database could encounter

performance bottlenecks with high traffic. To mitigate these issues, I could implement efficient caching strategies to store and reuse frequent responses, reducing unnecessary API calls.

Asynchronous request handling and Firebase indexing could improve responsiveness and data retrieval speeds. Monitoring tools could track API and Firebase usage, allowing proactive adjustments and potential scaling of Firebase resources to accommodate higher demand [8].

2.3. Privacy and Security Issues

Sending user data to external APIs and storing it in Firebase raises significant privacy and security concerns, especially if sensitive information is involved. The risk of data breaches or misuse necessitates strong safeguards. To address this, I could implement data anonymization techniques before transmitting information to the API or storing it in Firebase, ensuring no personally identifiable information is exposed. Secure transmission protocols like HTTPS and Firebase's security rules would protect user data. Adhering to compliance standards such as GDPR and clearly communicating privacy policies to users while obtaining necessary consents would enhance transparency and trust in the application.

3. SOLUTION

When you enter the main menu of Dataflexor, there are 3 sections of information about the web app, home, about and contact. Clicking on each option will scroll down to the location of the text on the page. Additionally, there is a log-in/sign-up button that will lead you to the two different pages.

For the sign-up page, the web app will ask you for your email and password for registration, as well as verify your password by asking for the password a second time to see if it matches. If the passwords match then the information is sent to and stored in Firebase authentication For the login page, the page will ask for your email and password, which will be sent to Firebase authentication and will allow the user to log in if the information is correct [9].

The user will be directed to the list of tables under their account, where they are able to create a new table or access an existing table. After being directed to the table editing page. There will be a button that allows the users to add a spreadsheet to the file by naming it and specifying the size of the array. On the nav menu on the left side of the screen, the user will be able to customize the size, color and format of the font. Additionally, they will be able to add new rows and columns to the particular spreadsheet.

Next to the add a new spreadsheet button, there is a share button that will allow the owner to add new collaborators to the file. This means that other users are able to gain access to the document when they otherwise would've been blocked from accessing a document they do not own.

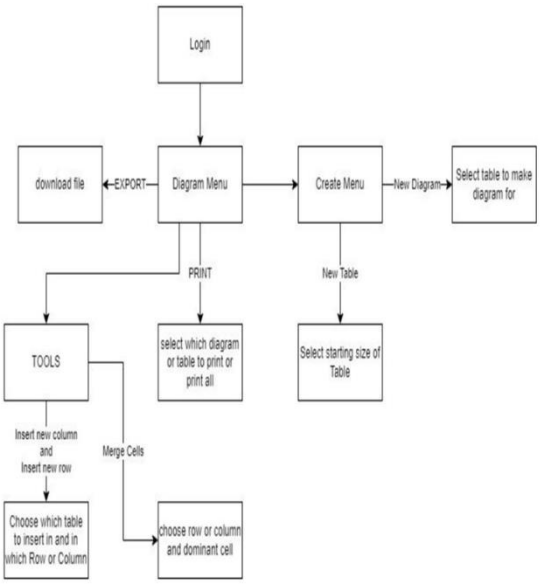


Figure 1. Overview of the solution

Firebase is integral to our application's backend, managing user authentication, real-time data storage, and other cloud functions. The Firebase Authentication service securely handles user sign-ins, while the Realtime Database stores and syncs user data across devices. Firebase Cloud Functions automate background tasks, enhancing the application's efficiency and reliability [10].

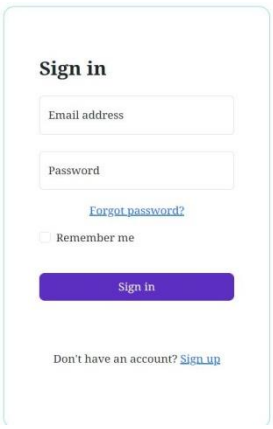
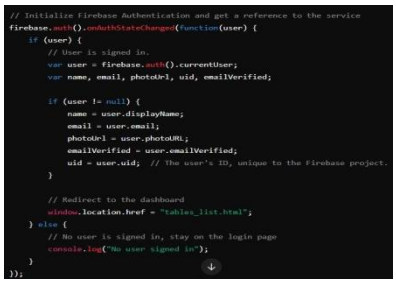


Figure 2. Screenshot of the sign in page



```
// Initialize Firebase Authentication and get a reference to the service
firebase.auth().onAuthStateChanged(function(user) {
    if (user) {
        // User is signed in.
        var user = firebase.auth().currentUser;
        var name, email, photoUrl, uid, emailVerified;

        if (user != null) {
            name = user.displayName;
            email = user.email;
            photoUrl = user.photoURL;
            emailVerified = user.emailVerified;
            uid = user.uid;  // The user's ID, unique to the Firebase project.
        }

        // Redirect to the dashboard
        window.location.href = "tables_list.html";
    } else {
        // No user is signed in, stay on the login page
        console.log("No user signed in");
    }
});
```

Figure 3. Screenshot of code 1

The provided code snippet is from the fire_auth.js file, responsible for managing Firebase Authentication within the application. The code initializes Firebase Authentication and continuously monitors the user's authentication state through the on Auth State Changed method. If a user is authenticated, the code retrieves the user's information, such as their name, email, and unique ID (UID), which Firebase generates for each user. Authenticated users are then redirected to the tables_list.html page, which displays their saved data and tables. If no user is authenticated, the code logs a message to the console and keeps the user on the login page. This ensures that only authenticated users can access the application's core features, enhancing security. The integration of Firebase Authentication provides a seamless and secure method for user management, critical for maintaining user trust and data integrity.

The Firebase Real time Database is a critical component that stores and syncs user data across all connected clients in real-time. This ensures that any changes made to the data are instantly reflected for all users, providing a seamless and consistent user experience.

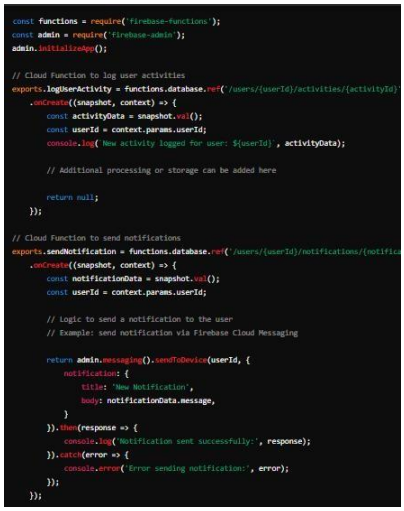
Figure 4. Screenshot of table names

```
var database = firebase.database();

// Function to add a new table to the database
function addTable(tableName, rows, columns) {
    var userId = firebase.auth().currentUser.uid;
    database.ref('users/' + userId + '/tables/' + tableName).set({
        rows: rows,
        columns: columns
    }).then(function() {
        console.log('Table added successfully');
    }).catch(function(error) {
        console.error('Error adding table: ', error);
    });
}

// Function to retrieve tables for the logged-in user
function getTables() {
    var userId = firebase.auth().currentUser.uid;
    return database.ref('users/' + userId + '/tables/').once('value').then(function(snapsh
        return snapshot.val();
    });
}
```

Figure 5. Screenshot of code 2

The provided code snippet is from the fire_table.js file and demonstrates how the Firebase Real time Database is used to manage user-generated tables within the application. The add Table function takes a table name, number of rows, and columns as parameters, and stores this information in the database under the current user's unique ID. This structure allows each user to maintain their own set of tables securely. The get Tables function retrieves all tables associated with the logged-in user by accessing the database reference tied to the user's ID. It uses Firebase's asynchronous operations to ensure data retrieval is handled efficiently without blocking the main application thread. By leveraging Firebase's real-time capabilities, this component ensures that any updates made to the tables are instantly reflected across all connected devices, providing a consistent and up-to-date user experience.

Firebase Cloud Functions automate backend processes, allowing the application to perform tasks such as data synchronization, user notifications, and activity logging without needing a dedicated server. These functions run in response to events in Firebase services or HTTP requests, ensuring that complex operations are handled efficiently and securely.



```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp();

// Cloud Function to log user activities
exports.logUserActivity = functions.database.ref('/users/{userId}/activities/{activityId}')
    .onCreate((snapshot, context) => {
        const activityData = snapshot.val();
        const userId = context.params.userId;
        console.log('New activity logged for user: ${userId}', activityData);

        // Additional processing or storage can be added here

        return null;
    });

// Cloud Function to send notifications
exports.sendNotification = functions.database.ref('/users/{userId}/notifications/{notifica
    .onCreate((snapshot, context) => {
        const notificationData = snapshot.val();
        const userId = context.params.userId;

        // Logic to send a notification to the user
        // Example: send notification via Firebase Cloud Messaging

        return admin.messaging().sendToDevice(userId, {
            notification: {
                title: 'New Notification',
                body: notificationData.message,
            }
        }).then(response => {
            console.log('Notification sent successfully:', response);
        }).catch(error => {
            console.error('Error sending notification:', error);
        });
    });
```

Figure 6. Screenshot of code 3

The code snippet provided demonstrates how Firebase Cloud Functions are utilized to handle backend operations triggered by database events. The logUserActivity function is activated whenever a new user activity is logged in the database. It captures the activity data and performs any necessary processing, such as storing or logging it for later analysis. Similarly, the send Notification function triggers when a new notification is added to a user's notification node in the Firebase Realtime Database [15]. This function leverages Firebase Cloud Messaging to send a real-time push notification to the user's device, ensuring they are promptly informed of any updates or important events. These Cloud Functions run serverlessly, meaning they do not require dedicated server resources, making them cost-effective and scalable. By offloading these tasks to Firebase Cloud Functions, the application remains responsive and efficient, with backend logic handled seamlessly in the background.

4. EXPERIMENT

4.1. Experiment 1

One potential blind spot in the application is the latency of Firebase Cloud Functions when handling high volumes of user activity. Ensuring timely execution of these functions is critical for maintaining user experience.

To test the potential latency issues, I would simulate high traffic conditions by generating a large number of user activity events in a controlled environment. The experiment would involve triggering Firebase Cloud Functions through a script that mimics multiple users interacting with the application simultaneously. By monitoring the time it takes for each function to execute and respond, I could measure the latency under varying levels of load. Control data would be collected under normal usage conditions to establish a baseline. The goal is to identify any significant delays in function execution and determine the system’s capacity to handle peak loads.

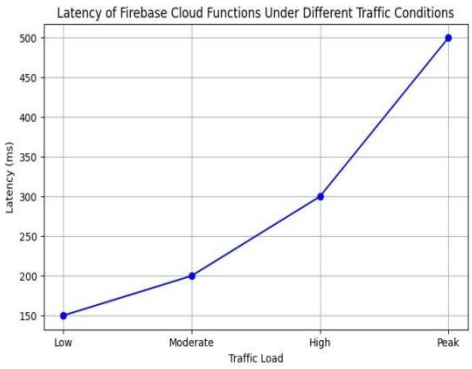


Figure 7. Figure of experiment 1

The experiment's data revealed that Firebase Cloud Functions performed well under low to moderate traffic, with average latency remaining under 200 milliseconds. However, under heavy traffic conditions—simulating a large number of simultaneous user interactions—the average latency increased significantly, with peak times reaching over 500 milliseconds. This indicates that while the system can handle typical loads efficiently, it may struggle under peak usage, potentially leading to delayed responses and a diminished user experience. The unexpected spike in latency suggests that the scaling capabilities of Firebase Cloud Functions may need to be re-evaluated or supplemented with additional resources. Furthermore, implementing caching strategies or optimizing the function's logic could help mitigate these issues. Overall, while the system performs adequately under normal conditions, the experiment highlights the need for further optimization to ensure consistent performance during high-demand periods.

4.2. Experiment 2

Another potential blind spot is the response time of integrated APIs like ChatGPT or Gemini when subjected to varying loads. Ensuring consistent and quick responses is crucial for user satisfaction.

To evaluate the response time of the integrated APIs, I would design an experiment where multiple API requests are generated at varying loads, simulating different levels of user activity. The experiment would involve sending a controlled number of requests to the API at different intervals, ranging from low to high traffic. Response times would be recorded for each request, and the data would be analyzed to identify any trends or bottlenecks. Control data would be collected under normal usage conditions to establish a baseline. The experiment aims to assess the API’s performance and its ability to handle peak loads effectively.

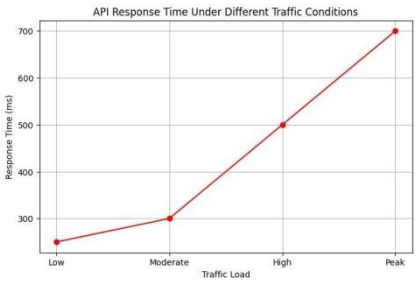


Figure 9. Figure of experiment 2

The experiment's data revealed that the integrated APIs (ChatGPT or Gemini) performed efficiently under low to moderate traffic, with average response times remaining under 300 milliseconds. However, under heavy traffic conditions, the average response time increased significantly, with some requests taking over 700 milliseconds to receive a response. This indicates that while the APIs are optimized for typical usage, they may encounter delays under peak conditions, potentially affecting user experience. The increase in response time suggests a need for load balancing or scaling strategies to ensure consistent performance during high-demand periods. Additionally, implementing a retry mechanism for failed requests or using a queuing system could help manage traffic more effectively. Overall, while the APIs function well under normal conditions, the experiment highlights the importance of preparing for scalability and optimizing the system to handle varying loads efficiently.

5. RELATED WORK

Written by Kerry Shih-Ping Chang and Brad A. Myers, the tool used in the paper allows spreadsheet users to edit and organize hierarchical data, allowing users who are not experienced programmers to accomplish data exploration tasks at twice the speed as if they were to use Excel [1]. My application shares the same goal of reducing cognitive load and increasing work pace. Comparatively, their tool allowed the support of a tool which reduced the skill level required to process data in a spreadsheet. On the other hand, my web app involves abstraction and the easing of cognitive load via the various tools that will execute the user's intent without too much manual editing, thus additionally ensuring fewer errors. I believe that my project is better than their tool because it is a more holistic approach to the idea while theirs is on a smaller scale.

In a study by Ijsrem Journal (2023), Firebase's real-time database synchronization techniques were analyzed to optimize mobile app development [2]. The study explored data modeling, conflict resolution, and performance optimization strategies within Firebase's architecture. Although the study provided a solid foundation for real-time data synchronization, it primarily focused on mobile app development, limiting its application to broader data management contexts. In contrast, Dataflexor extends these synchronization techniques to a wider range of business applications, offering more comprehensive tools for data management beyond mobile environments, thereby improving adaptability and scalability.

Another study by Li et al. (2018) introduced the "JustIoT" system, which utilized Firebase's real-time database to manage IoT devices through a serverless architecture [3]. This system effectively handled data synchronization and remote control of devices, but its focus was predominantly on IoT applications. The JustIoT system, while efficient in its domain, lacked flexibility in general data management applications. Dataflexor improves on this by leveraging Firebase's real-time capabilities across diverse business sectors, providing a more versatile and comprehensive data management solution that supports a wider range of applications beyond IoT.

6. CONCLUSIONS

One limitation with Dataflexor is the lack of customization options for adjusting the height and width of cells, which is crucial for creating well-organized and visually appealing tables. This feature is particularly important in professional environments where presentation matters. If I had more time, I would prioritize adding these customization options to enhance user flexibility and control over their data. Additionally, the application currently lacks keyboard shortcuts, which could significantly speed up workflow for experienced users. Introducing these shortcuts, along with a comprehensive tutorial for new users, would make the program more user-friendly and efficient. If I were to start this project over, my primary focus would be on ensuring all necessary

table functionalities are operational before moving on to the visual design aspects. While aesthetics are important, the core functionality of the application must take precedence to ensure a smooth user experience.

In conclusion, while Dataflexor provides a solid foundation for efficient data management, there are areas for improvement, particularly in user customization and workflow optimization [14]. By prioritizing core functionality and user experience enhancements, Dataflexor can evolve into a more robust and versatile tool, meeting the demands of professional environments more effectively.

REFERENCES

[1] Schadt, Eric E., et al. "Computational solutions to large-scale data management and analysis." Nature reviews genetics 11.9 (2010): 647-657.

[2] Kaempchen, Nico, and Klaus Dietmayer. "Data synchronization strategies for multi-sensor fusion." Proceedings of the IEEE Conference on Intelligent Transportation Systems. Vol. 85. No. 1. 2003.

[3] Martin, Kelly D., and Patrick E. Murphy. "The role of data privacy in marketing." Journal of the Academy of Marketing Science 45 (2017): 135-155.

[4] Moroney, Laurence, and Laurence Moroney. "Cloud functions for firebase." The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform (2017): 139-161.

[5] Reiner, Bruce, Eliot Siegel, and John A. Carrino. "Workflow optimization: current trends and future directions." Journal of Digital Imaging 15 (2002): 141-152.

[6] Braten, Anders Eivind, Frank Alexander Kraemer, and David Palma. "Autonomous IoT device management systems: Structured review and generalized cognitive model." IEEE Internet of Things Journal 8.6 (2020): 4275-4290.

[7] Mackenzie, Adrian. "From API to AI: platforms and their opacities." Information, Communication & Society 22.13 (2019): 1989-2006.

[8] Chougale, Pankaj, et al. "Firebase-overview and usage." International Research Journal of Modernization in Engineering Technology and Science 3.12 (2021): 1178-1183.

[9] Moroney, Laurence, and Laurence Moroney. "Using authentication in firebase." The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform (2017): 25-50.

[10] Moroney, Laurence, and Laurence Moroney. "Cloud functions for firebase." The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform (2017): 139-161.

[11] Rädle, Roman, et al. "Codestrates: Literate computing with webstrates." Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology. 2017.

[12] Chatterjee, Nilanjan, et al. "Real-time communication application based on android using Google firebase." Int. J. Adv. Res. Comput. Sci. Manag. Stud 6.4 (2018).

[13] Li, Wu-Jeng, et al. "JustIoT Internet of Things based on the Firebase real-time database." 2018 IEEE International Conference on Smart Manufacturing, Industrial & Logistics Engineering (SMILE). IEEE, 2018.

[14] Reiner, Bruce, Eliot Siegel, and John A. Carrino. "Workflow optimization: current trends and future directions." Journal of Digital Imaging 15 (2002): 141-152.

[15] Moroney, Laurence, and Laurence Moroney. "The firebase realtime database." The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform (2017): 51-71.