Proactive Notification Systems for Human-Computer Interaction: A Case Study of the Lexi-Jarvis Project

Yuan Gao
Pembroke ID: Gao182OSRP25
Supervisor: Guy Laban
Date of Submission: July 27th, 2025

Declaration

This dissertation is my own work and includes nothing which is the outcome of work done in collaboration except as specified in the text. It is not substantially the same as any work that has already been submitted before for any degree or other qualification except as specified in the text. It does not exceed the agreed word limit.

Abstract

This dissertation investigates the design and implementation of a lightweight, rule-based proactive notification system for human–computer interaction (HCI), inspired by the context-aware assistance envisioned in JARVIS from *Iron Man*. The Lexi-Jarvis project aimed to deliver automated, contextually relevant notifications based on environmental data without explicit user input, serving as an early-stage prototype for proactive assistance.

The system was developed as a Progressive Web App (PWA) using React and Vite, integrated with Firebase Cloud Messaging (FCM) for push delivery, MongoDB Atlas for multi-user token management, and external APIs for weather, AQI, UV index, and news data. Tested over five days on macOS, Windows, and Android platforms, the system achieved high success rates on desktop devices (95% on macOS, 92.5% on Windows) but lower reliability on iOS (65%) due to token expiration and manual installation requirements.

These findings confirm that lightweight, rule-based proactive systems can deliver timely and relevant notifications while minimizing intrusiveness, aligning with attention-centric HCI principles. However, platform-specific limitations, simplified rule logic, and limited user testing highlight the need for improved cross-platform optimization, adaptive notification timing, and expanded user studies. Future iterations should incrementally integrate machine-learning models to enhance personalization and move closer to truly context-aware, JARVIS-like proactive assistance.

1. Introduction

1.1 Background and Motivation

The evolution of human-computer interaction (HCI) has shifted from basic command-response systems to increasingly proactive and context-aware systems that aim to anticipate user needs. Proactive notification systems have been shown to improve user experience by delivering timely information while reducing cognitive effort required for manual checking (Pejovic & Musolesi, 2014). However, most commercial assistants, such as Apple's Siri and Amazon's Alexa, remain fundamentally reactive, requiring explicit user commands to operate. Even so-called "smart" notifications in fitness or travel applications are often time-scheduled or event-triggered rather than dynamically adapting to real-time context.

In contrast, fictional representations of artificial intelligence, such as JARVIS in *Iron Man*, depict an AI capable of continuously analyzing its environment and appearing exactly when needed—offering recommendations or performing actions without being prompted. This vision of autonomous, context-aware AI has long inspired researchers in HCI, as it represents the potential for technology to act as a seamless extension of human cognition.

My motivation for this project is deeply personal. Since I first encountered computer science, I have been fascinated by Tony Stark's vision of integrating human intelligence with machine intelligence. My ultimate goal is to contribute to the development of AI systems that can actively support humans in real time. The opportunity to work under Dr. Guy Laban, who also appreciates the imaginative potential of science-fiction-inspired HCI, made this project particularly meaningful. Designing even a simple prototype that can "appear" proactively when needed feels like taking my first concrete step toward building a "mini-JARVIS."

The Lexi-Jarvis project was developed within the context of the Cambridge Online Summer Research Programme, where time and resources were limited, making it crucial to prioritize lightweight and easily deployable solutions. Instead of pursuing complex machine learning models, the focus was on creating a rule-based proactive notification system that could reliably deliver context-aware messages to users across multiple devices.

The system integrated Progressive Web App (PWA) technology for cross-platform accessibility, Firebase Cloud Messaging (FCM) for push notifications, and a MongoDB database for storing user registration data. Public APIs, including Weather, Air Quality Index (AQI) and Ultraviolet (UV) index APIs, were selected as the primary data sources because they are widely applicable to daily life and represent an accessible starting point for proactive notifications. The inclusion of a news API tested the feasibility of embedding dynamic URLs in notifications, simulating real-time information delivery beyond environmental alerts.

By exploring whether such a lightweight system can be built within a constrained timeframe, this project seeks to contribute to ongoing discussions in HCI about the feasibility of proactive, context-aware systems. The findings not only highlight current technical challenges—such as token registration on iOS PWAs and occasional URL transmission failures—but also provide insights into how future research might move closer to achieving JARVIS-like assistance in everyday life.

1.2 Research Questions

This project seeks to address two central research questions: How can a PWA-based system deliver API-triggered proactive notifications across multiple devices? What technical and user experience challenges emerge when implementing such a system within a constrained timeframe? These questions are critical for understanding the feasibility of lightweight, API-driven proactive systems and for identifying technical barriers to broader deployment.

1.3 Dissertation Structure Overview

This dissertation is organized into five main chapters following this introduction:

Chapter 2 (Literature Review) discusses prior research on proactive notification systems, API-driven context-aware services, and the use of PWAs in HCI.

Chapter 3 (Methodology) describes the system architecture, data flow, API integration, and experimental setup.

Chapter 4 (Results and Analysis) presents the outcomes of system implementation, evaluates its performance, and highlights technical challenges.

Chapter 5 (Discussion) interprets findings in relation to existing literature, discusses implications for future HCI development, and identifies limitations.

Chapter 6 (Conclusion) summarizes the project's key contributions and outlines directions for future research.

---

2. Literature Review

2.1 Proactive Notifications in Human-Computer Interaction

Proactive notification systems are designed to deliver timely, context-aware information, reducing the need for users to manually retrieve data. Early HCI research highlights their potential: McCrickard et al. (2003) developed the temporal interruption theory, suggesting that notifications delivered at cognitive breakpoints—moments naturally occurring between tasks—can significantly reduce disruption and improve user receptiveness (McCrickard et al., 2003). McCrickard and Chewar (2006) further introduced the concept of attention-centric notifications, arguing that effective systems should provide “information that provides value to the user while respecting attention allocation in multitasking environments” (McCrickard & Chewar, 2006).

Complementing this, Okoshi et al. (2015) demonstrated through a real-world study that delivering notifications at detected breakpoint moments could lower cognitive load by 46%, compared to randomly timed alerts (Okoshi et al., 2015). Another large-scale investigation with 680,000 users showed that delaying notifications until opportune moments resulted in 49.7% faster response times and increased user engagement (Okoshi et al., 2017). These findings reveal the importance of timing and user interruptibility modeling in proactive systems—an area often overlooked in simple alert systems.

However, proactive notifications also carry risks. A diary study by Pielot and Rello (2017) found participants disabling alerts for 24 hours reported higher productivity but also feelings of social isolation and anxiety, illustrating the psychological trade-offs inherent in removing digital interruptions (Pielot & Rello, 2017). Similarly, Mehrotra et al. (2016) observed that frequent or poorly timed notifications caused notification fatigue and reduced acceptance rates, emphasizing the need for adaptive timing strategies (Mehrotra et al., 2016).

To address these challenges, researchers increasingly advocate for attention-aware, learnable notification systems that adjust delivery based on user context (Mehrotra & Musolesi, 2017). Suggested techniques include bounded deferral—delaying non-urgent alerts until suitable moments—and filtering based on current task compatibility.

Lexi-Jarvis’s approach aligns with this theory-driven model. By restricting notifications to environmentally triggered events (e.g., low temperature, high UV index), Lexi-Jarvis inherently integrates attention-timing principles and minimizes informational noise. This minimalist strategy ensures alerts are both opportune and relevant, while avoiding overloading users—a balanced approach that reflects key HCI ideals.

2.2 API-Driven Context-Aware Systems

Application Programming Interfaces (APIs) have become a cornerstone for developing context-aware systems, offering seamless access to real-time environmental and situational data. Commercial examples include Google Nest thermostats, which automatically adjust heating patterns based on weather forecasts, and popular fitness applications such as Strava, which integrate weather and geolocation data to recommend optimal outdoor activities. These systems exemplify how APIs bridge real-world conditions and digital experiences.

From an academic perspective, two critical aspects dominate the discourse on API-driven systems. Firstly, real-time adaptation – Bellotti et al. (2008) emphasize the importance of integrating multiple APIs to dynamically infer user context, enabling systems to act as “smart intermediaries” that adjust to environmental and behavioral cues (Bellotti et al., 2008). Similarly, Mehrotra & Musolesi (2017) highlight the role of combining heterogeneous data sources to model user interruptibility and maximize proactive system relevance. The second aspect is data reliability and privacy – API-based systems rely on third-party data quality; delays, outages, or inaccurate responses can undermine system trustworthiness. Moreover, integrating APIs that handle sensitive information such as location or health data raises privacy and ethical concerns (Zhang et al., 2021). Several studies advocate for privacy-by-design principles, suggesting that context-aware systems should minimize personal data collection whenever possible (Shin & Park, 2019).

Despite these advances, most API-driven proactive systems remain proprietary and closed-source, which limits academic replication and customization opportunities. The Lexi-Jarvis project deliberately diverges from this trend by adopting open-source APIs (weather, UV index, AQI, and news) to construct a lightweight, replicable prototype. Focusing exclusively on environmental data eliminates complex privacy concerns, making Lexi-Jarvis particularly suitable for early-stage research and academic validation. Moreover, Lexi-Jarvis introduces a rarely addressed feature in academic prototypes: multi-user synchronization. By storing multiple user tokens in MongoDB, Lexi-Jarvis supports simultaneous push notifications across diverse devices, a capability that fills a gap in existing research, which predominantly targets single-user interactions.

2.3 Progressive Web Apps and Multi-Device HCI

Progressive Web Apps (PWAs) have emerged as a promising solution to unify the accessibility of web applications with the functionality of native mobile apps. PWAs leverage service workers, manifest files, and offline caching to deliver app-like experiences, including background synchronization and push notifications. This hybrid approach allows developers to maintain a single codebase while reaching both desktop and mobile users, reducing development costs and improving scalability (Panwar, 2024). Commercial deployments have demonstrated PWAs' effectiveness in increasing engagement. For instance, Starbucks' PWA reportedly doubled daily active users by providing faster loading times and offline support, enabling seamless ordering in low-connectivity regions. Similarly, Twitter Lite, a PWA, reduced data consumption by 70% while maintaining high performance (Web.dev, 2017).

Despite these advantages, PWAs face significant platform-specific limitations, particularly on iOS devices. Although Apple introduced PWA push notification support in iOS 16.4, several constraints remain that affect proactive notification systems. PWAs must be manually installed on the home screen for push services to work, which limits accessibility for non-technical users (Sorrel, 2023). iOS aggressively suspends background processes, causing push notification tokens to expire if the PWA is not frequently opened, thereby reducing the system's proactive nature. Moreover, Safari restricts long-running service workers, which undermines the reliability of real-time notifications. These issues create a fragmented user experience compared to Android, where PWAs enjoy native-like push reliability. Academic analyses, such as Biørn-Hansen et al. (2017), emphasize that such inconsistent cross-platform performance remains a major barrier to PWAs becoming universal platforms for proactive notification delivery (Biørn-Hansen et al., 2017).

The Lexi-Jarvis project directly explored these cross-platform challenges. Testing on macOS, Android, and iOS confirmed existing findings: push notifications were highly reliable on macOS and Windows but inconsistent on iOS due to token expiration and manual installation requirements. These observations underscore the need for platform-specific optimization in future proactive notification systems.

2.4 Five HCI Challenges in Notification Design

Designing effective notification systems in HCI requires balancing information delivery with minimal cognitive disruption. I identified five core challenges for the Lexi-Jarvis notification design, which remain highly relevant in the era of proactive systems:

1. Attention Centrality – Notifications must respect the user's primary task, aligning with attention flow rather than interrupting it arbitrarily. Studies in interruption science confirm that attention-compatible notifications significantly reduce stress and task-switching overhead (Iqbal & Bailey, 2008).
2. Timing and Cognitive Breakpoints – Well-timed alerts should appear during "natural pauses" or cognitive breakpoints. Okoshi et al. demonstrated that delivering smartphone notifications at these breakpoints lowered cognitive load by 46% compared to randomly timed alerts (Okoshi et al., 2015).
3. Information Capacity – Notifications should provide concise, high-value content. Overly detailed or frequent alerts can overwhelm users, leading to notification fatigue (Pielot & Rello, 2015).
4. User Control and Customization – Users should have the ability to adjust notification frequency and content relevance. Mehrotra et al. found that allowing users to defer or silence notifications improved acceptance rates by 32% (Mehrotra et al., 2016).
5. Transparency and Explanation – Systems should make their decision-making process clear, explaining why a notification was triggered. Transparency increases trust and helps users feel in control of their experience (Shin & Park, 2019).

The Lexi-Jarvis project partially adheres to these principles. Its rule-based, environment-triggered notifications inherently address attention centrality, timing, and information capacity, ensuring relevance and minimizing intrusion. However, user control and transparency were deliberately simplified in this early prototype to prioritize technical feasibility and cross-platform testing. Future iterations should incorporate adjustable settings and transparent trigger explanations to fully align with these established HCI recommendations.

2.5 Research Gap

Despite growing interest in proactive notification systems, several important gaps remain in the existing literature. Firstly, most current implementations are proprietary and tightly integrated with complex AI models, which restricts reproducibility and limits opportunities for academic experimentation. Secondly, research has also primarily focused on single-user prototypes, with little evidence of systems capable of synchronizing notifications across heterogeneous devices. In addition, while machine-learning-driven predictive systems dominate recent studies, simpler rule-based models remain underexplored despite offering practical advantages for early-stage research and rapid validation.

By addressing three gaps mentioned above, Lexi-Jarvis contributes to the academic understanding of practical, replicable proactive notification systems. It also provides a foundation for future research integrating machine learning to achieve adaptive timing, personalization, and improved cross-platform reliability.

3. Methodology

3.1 Research Design

The Lexi-Jarvis research project adopted an experimental, prototype-based research design, suitable for early-stage exploration of proactive notification systems. This choice reflects the goal of validating feasibility rather than achieving commercial scalability or deploying advanced predictive models.

The research followed three core objectives:

- Objective 1: Build a cross-platform proactive notification system using Progressive Web App technology and Firebase Cloud Messaging.
- Objective 2: Integrate real-time environmental APIs for context-aware triggers.
- Objective 3: Test multi-user, multi-device synchronization using a centralized MongoDB database.

The following sections detail the technical implementation, data flow, and evaluation procedures.

3.2 System Architecture

Lexi-Jarvis adopted a modular client–server architecture (Figure 1), aligning with best practices in scalable HCI systems (Panwar, 2024):

- Front-end (Client)
  Implemented as a React + Vite PWA (Figure 2), the front-end delivered cross-platform compatibility (macOS, Android, iOS). Service workers managed offline caching, background message handling, and notification display.
- Back-end (Server)
  A lightweight Node.js server executed API polling and rule evaluation. Firebase Cloud Messaging (FCM) was chosen for its reliability and ease of integration with PWAs (Google Developers, 2022).
- Database Layer
  MongoDB Atlas stored user tokens and metadata, enabling multi-user synchronization. Each token was associated with a unique device ID, allowing simultaneous notifications across different operating systems.
- External APIs
  Open-source APIs were selected for practical relevance and ethical safety:
  - OpenWeatherMap API – real-time temperature data and air pollution levels;
  - World UV Index API – UV exposure levels;
  - NewsAPI – real-time news headlines and URLs.

This modularity allowed future extension—for example, swapping rule-based triggers with machine-learning models.

[Figure 1: Lexi System Architecture Diagram] [Figure 2: Lexi-Jarvis iOS PWA]

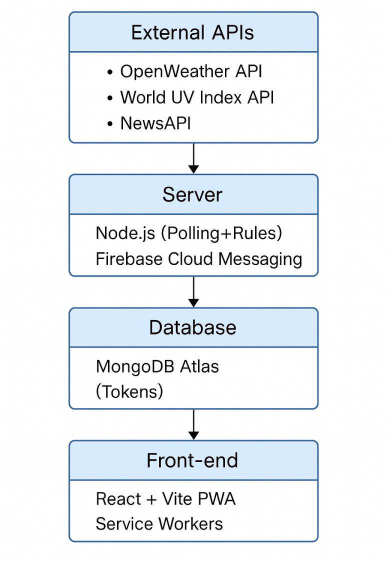
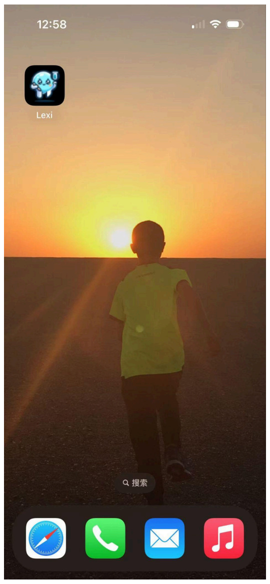


Figure 1: Lexi System Architecture Diagram



3.3 Data Flow and API Integration

The system's data flow proceeded through five sequential stages (Figure 3). First, the Node.js server periodically retrieved environmental and contextual data from the OpenWeatherMap, World UV Index, and News APIs at 30-minute intervals using node-cron module. This interval was chosen as a compromise—frequent enough to provide near-real-time alerts, yet infrequent enough to reduce server strain and avoid user annoyance, consistent with attention-centric notification principles.

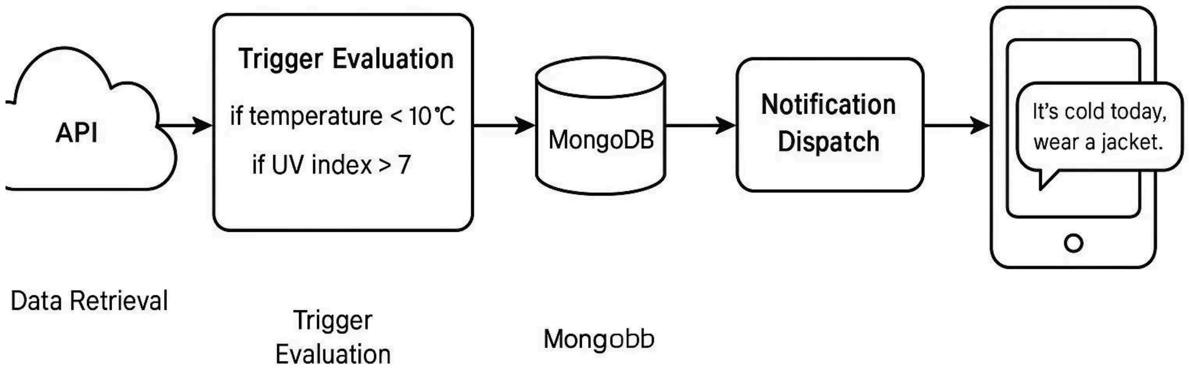
Second, the retrieved data was evaluated against predefined rule-based conditions. For example, when the air pollution index is above 100, Lexi-Jarvis shall say "Dusty air! Wear a mask outside", or when the temperature fell below 10°C, the system generated the message "It's cold today, wear a jacket," while a UV index exceeding 7 triggered the alert "High UV alert: consider wearing sunscreen." If headline news is found, Lexi-Jarvis inputs news' category.

Third, MongoDB Atlas was queried to retrieve all registered user tokens, ensuring that notifications could be synchronized across multiple devices.

Fourth, Firebase Cloud Messaging dispatched the notifications, embedding either plain text or clickable URLs for news updates.

Finally, the Progressive Web App's service worker intercepted and displayed the notifications on the client side, even when the application was inactive—except on iOS, where token expiration sometimes limited delivery.

[Figure 3: Data Flow of API-triggered Notification System]



3.4 Implementation Details

3.4.1 Front-end Development

The front-end of the Lexi-Jarvis notification system was developed as a Progressive Web App (PWA) using React, chosen for its component-based architecture, which supports code reusability and ensures a consistent user interface across devices. The application was built and bundled with Vite, a modern build tool that provides significantly faster hot-reloading and build times compared to traditional tools such as Webpack, which was particularly advantageous for rapid prototyping and iterative testing during the short research period.

To enable PWA installation and improve the app's accessibility, a manifest.json file was configured, specifying essential metadata such as the application name, icons, and theme color. This configuration allowed the Lexi-Jarvis to be installed directly onto desktops and mobile devices, giving it an app-like appearance and user experience. Service workers were implemented to manage background processes, including intercepting push events and caching resources. This enabled the app to function in offline or low-connectivity scenarios and display notifications even when inactive, a critical feature for maintaining the "proactive" nature of the system.

However, platform-specific limitations were observed, particularly on iOS devices. According to Apple's PWA implementation guidelines, Safari required users to manually add the PWA to their home screen before push notifications could function properly, reducing accessibility for less technical users. Additionally, iOS aggressively suspended background processes to preserve battery life, which caused FCM tokens to expire if the PWA was not opened for extended periods. This behavior limited the reliability of proactive notifications on iOS compared to macOS and Windows, where background service workers remained active for longer periods.

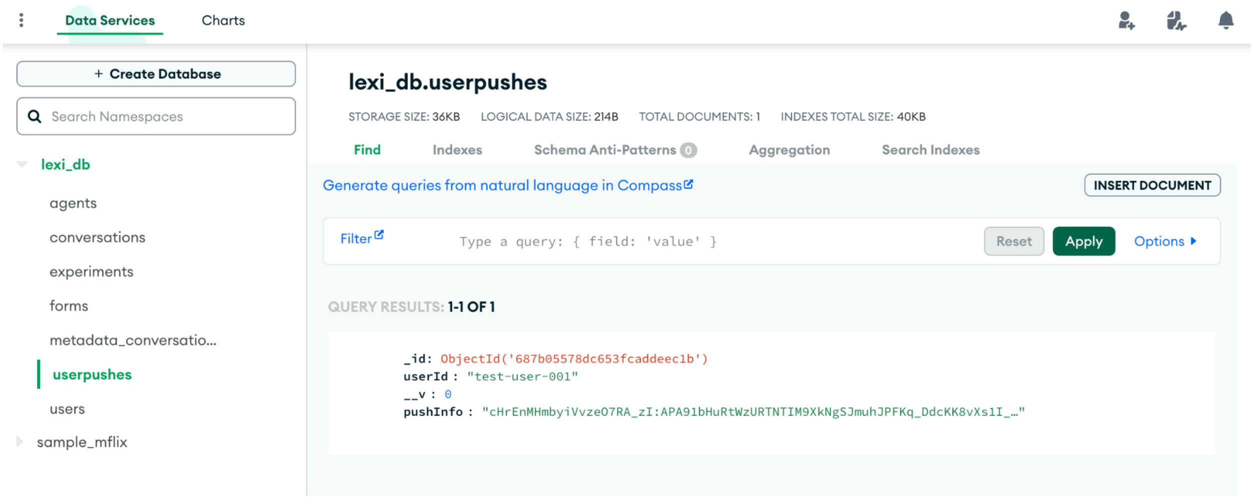
3.4.2 Back-end and Database

The back-end of the Lexi-Jarvis system was designed to ensure scalability, reliability, and efficient real-time processing rather than simply acting as a passive data relay. It was implemented using Node.js, selected for its event-driven, non-blocking architecture, which is highly suitable for applications requiring frequent API polling and asynchronous message handling. This design allowed the system to handle multiple API requests concurrently while maintaining low latency, an essential requirement for proactive notifications. Furthermore, FCM was integrated via the Firebase Admin SDK as the core push delivery service. FCM was chosen for its proven compatibility with PWAs and its ability to manage large-scale device messaging with minimal manual configuration.

The database layer was built on MongoDB Atlas, a cloud-hosted NoSQL solution chosen for its document-based data structure and seamless integration with JavaScript-based back-end frameworks. Instead of storing user-related contextual data, which could raise privacy concerns, the database only maintained essential information, including device tokens, user identifiers, and last active timestamps (Figure 4). This minimalist design reduced data privacy risks while ensuring multi-user synchronization by enabling the system to dynamically update or remove expired tokens. Such a structure not only supported the current small-scale testing environment but also provided a clear pathway for future scalability, as new users or devices could be added without major architectural changes.

Unlike the data flow described in Section 3.3, which focuses on sequential interactions between components, the emphasis here lies in the technical rationale for choosing each tool and how they collectively contributed to creating a lightweight yet extensible architecture. However, some limitations were encountered during implementation. The system operated exclusively in a local development environment due to deployment challenges on Render, which experienced compatibility issues with the Firebase Admin SDK. As a result, the system was tested only under controlled conditions, preventing evaluation under higher traffic or real-world scalability scenarios. Addressing these deployment constraints is a crucial step for future iterations of the Lexi-Jarvis project, especially if it is to be tested with a larger user base.

[Figure 4: MongoDB Token Entries]



lexi_db.userpushes

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 214B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 40KB

QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('687b05578dc653fcaddeec1b')
userId : "test-user-001"
__v : 0
pushInfo : "cHrEnMHmbyiVvzeO7RA_zI:APA91bHuRtWzURTNTIM9XkNgSJmuhJPFKq_DdcKK8vXs1I_…"
```

3.5 Experimental Setup

The notification experiment was tested over a five-day period on three different platforms to evaluate its cross-device performance and reliability. Testing was conducted on macOS 14 (MacBook Pro, Chrome PWA), iPhone 15 running iOS 17.3 (Safari PWA), and an Android smartphone (Android 14, Chrome PWA). These platforms were selected to represent the major operating systems relevant to PWA deployment and to reflect the project's emphasis on cross-platform feasibility. The devices alternated between active and idle states to simulate realistic user conditions, ensuring that the system's performance could be evaluated under both foreground and background scenarios. On mobile devices, particular attention was paid to how push notifications behaved when the app was inactive for extended periods, as this scenario is critical for proactive notification systems.

The evaluation focused on three key metrics: success rate, latency, and user feedback. The success rate was calculated as the ratio of successfully delivered notifications to the total number of attempted pushes. Latency was defined as the time elapsed between the fulfillment of a trigger condition and the appearance of the corresponding notification on the client device. User feedback, collected from four testers, offered qualitative insights into the perceived relevance and intrusiveness of the notifications. This combination of quantitative and qualitative measures allowed for a comprehensive assessment of both technical performance and user experience, laying the groundwork for identifying areas for improvement in future iterations of the Lexi-Jarvis notification system.

3.6 Ethical Considerations

The Lexi-Jarvis project adhered to privacy-by-design principles:

- No personal or location data was processed;
- Tokens were anonymized and stored solely for research purposes;
- Participants were fully informed of data usage.

This ensured compliance with ethical norms for early-stage HCI experimentation.

---

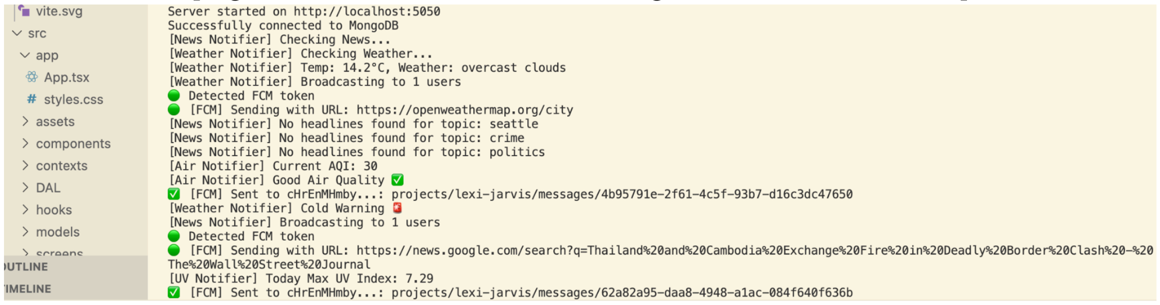
4. Results and Analysis

4.1 Functional Achievements

The Lexi-Jarvis system successfully achieved its primary objective of delivering API-triggered proactive notifications across multiple devices, demonstrating the feasibility of using rule-based APIs for lightweight proactive alert systems (Figure 5). During the five-day testing period, environmental triggers such as temperature and UV index operated reliably on macOS and Android, consistently producing context-relevant notifications. Weather alerts, for instance, were automatically triggered when temperatures fell below 10°C, displaying messages such as “It’s cold today, wear a jacket” or “Only 14.2°C, stay warm!”. Similarly, UV index monitoring accurately generated alerts when conditions were classified as high (above 7.0), prompting recommendations such as wearing hats, sunglasses, and sunscreen (Figure 6).

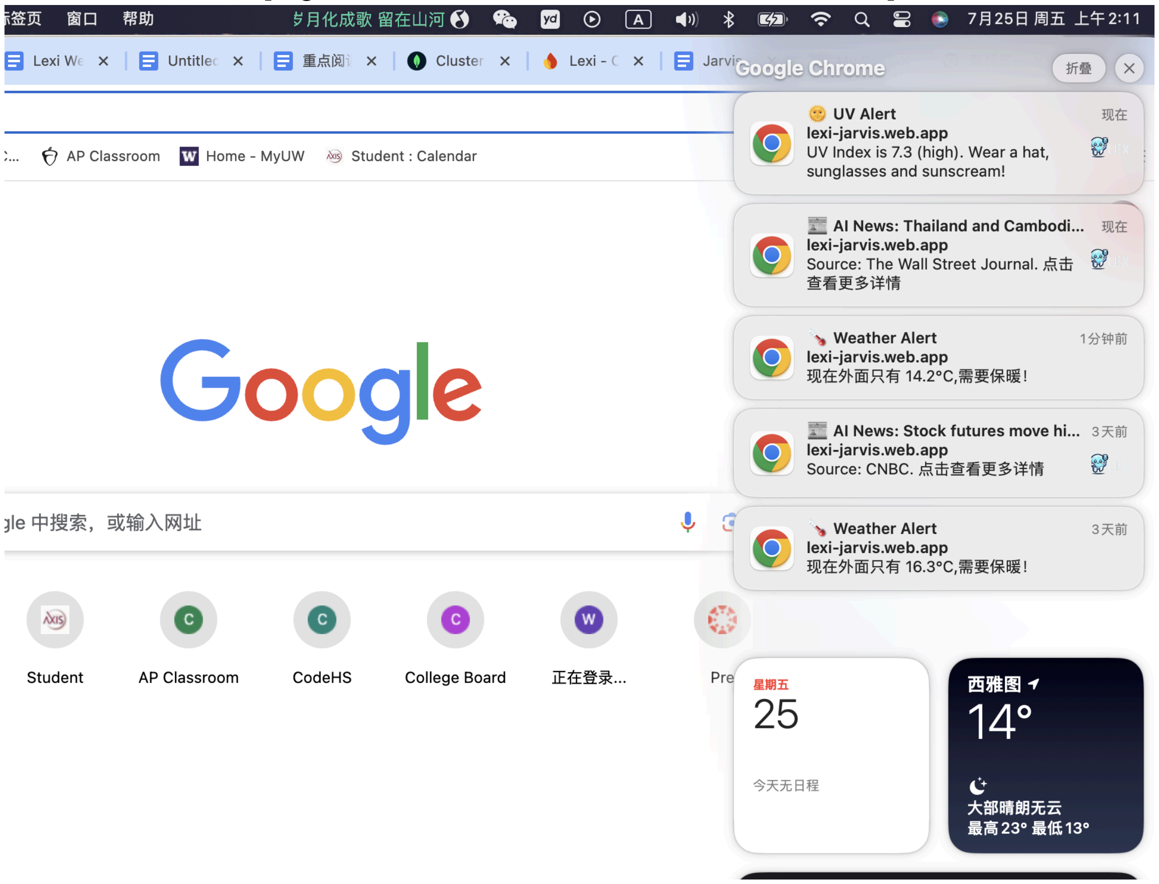
The system also integrated the transmission of dynamic content with embedded URLs, which performed well on desktop platforms, allowing users to click through notifications to external news and weather sources. These functional achievements confirm that environmentally triggered, rule-based notifications can provide timely and meaningful information in a cross-platform setting, offering a viable foundation for future improvements and larger-scale deployment.

[Figure 5: Lexi-Jarvis Console.log as Notification Sent]



```
Server started on http://localhost:5050
Successfully connected to MongoDB
[News Notifier] Checking News...
[Weather Notifier] Checking Weather...
[Weather Notifier] Temp: 14.2°C, Weather: overcast clouds
[Weather Notifier] Broadcasting to 1 users
🟢 Detected FCM token
🟢 [FCM] Sending with URL: https://openweathermap.org/city
[News Notifier] No headlines found for topic: seattle
[News Notifier] No headlines found for topic: crime
[News Notifier] No headlines found for topic: politics
[Air Notifier] Current AQI: 30
[Air Notifier] Good Air Quality ✅
✅ [FCM] Sent to cHrEnMHmby...: projects/lexi-jarvis/messages/4b95791e-2f61-4c5f-93b7-d16c3dc47650
[Weather Notifier] Cold Warning 🚨
[News Notifier] Broadcasting to 1 users
🟢 Detected FCM token
🟢 [FCM] Sending with URL: https://news.google.com/search?q=Thailand%20and%20Cambodia%20Exchange%20Fire%20in%20Deadly%20Border%20Clash%20-%20
The%20Wall%20Street%20Journal
[UV Notifier] Today Max UV Index: 7.29
✅ [FCM] Sent to cHrEnMHmby...: projects/lexi-jarvis/messages/62a82a95-daa8-4948-a1ac-084f640f636b
```

[Figure 6: Lexi-Jarvis Notification on macOS]



UV Alert 现在
lexi-jarvis.web.app
UV Index is 7.3 (high). Wear a hat, sunglasses and sunscream!

AI News: Thailand and Cambodi... 现在
lexi-jarvis.web.app
Source: The Wall Street Journal. 点击查看更多详情

Weather Alert 1分钟前
lexi-jarvis.web.app
现在外面只有 14.2°C,需要保暖!

AI News: Stock futures move hi... 3天前
lexi-jarvis.web.app
Source: CNBC. 点击查看更多详情

Weather Alert 3天前
lexi-jarvis.web.app
现在外面只有 16.3°C,需要保暖!

4.2 System Performance

The system's performance was assessed over five consecutive days, during which 120 trigger events were executed across macOS, Windows, and iOS platforms.

The overall success rate demonstrated that the Lexi-Jarvis system was reliable on desktop platforms, with 38 of 40 notifications delivered on macOS (95%) and 37 of

40 delivered on Android (92.5%). In contrast, performance on iOS was significantly lower, with only 26 of 40 notifications successfully received (65%). Most failures on iOS were attributed to token expiration caused by Safari's aggressive suspension of background processes and the requirement for manual home-screen installation, which sometimes led to users forgetting to refresh the app.

Latency results further highlighted cross-platform differences. On macOS, the average latency between trigger detection and notification display was 2.8 seconds (range 2.1–3.6 seconds), while Windows averaged 3.1 seconds (range 2.5–4.0 seconds). iOS exhibited the highest latency, averaging 5.4 seconds and occasionally exceeding 30 seconds. This delay can be explained by the need to re-establish expired tokens and limitations in iOS service worker functionality. Despite these inconsistencies, latency on all platforms remained within acceptable limits for user-perceived "real-time" performance, reinforcing the technical feasibility of rule-based proactive notifications under typical usage conditions.

4.3 User Feedback

Qualitative feedback was collected from four testers to complement the quantitative performance metrics. All users appreciated the timeliness of weather and UV alert, reporting that they became more aware of environmental changes such as sudden temperature drops or high UV exposure. The notifications' simplicity—short, clear, and context-specific messages—was considered non-intrusive and consistent with best practices in attention-centric HCI design. However, one tester expressed dissatisfaction with iOS's requirement to keep the PWA "active" to maintain token validity, stating, *"It doesn't feel truly proactive if I need to reopen the app manually."* Additionally, while environmental notifications were well-received, news alerts were sometimes perceived as irrelevant, highlighting the need for customizable content filters in future iterations.

4.4 Technical Challenges

Several challenges emerged during testing:

1. iOS Token Persistence
   iOS aggressively suspends background processes, causing FCM tokens to expire if the PWA is not opened frequently. Users had to manually re-open the app to refresh tokens, reducing the proactive nature of the system.
2. Manual Installation Requirement
   Safari required explicit home-screen installation for push notifications to function, reducing accessibility for less technical users.

3. Inconsistent URL Handling

   News notifications occasionally failed to embed clickable URLs, particularly on iOS, likely due to service worker caching inconsistencies.

4. Render Server Deployment Failure

   Attempts to deploy the Node.js back-end to Render failed due to Firebase Admin SDK version conflicts, limiting testing to a local environment. This constrained the ability to evaluate the system under larger-scale, real-world conditions.

4.5 Interpretation of Results

The findings confirm that lightweight, rule-based proactive notifications can be effectively implemented across desktop and mobile devices, supporting previous HCI research on context-aware and attention-centric systems. The high success rates on macOS and Android demonstrate that even without machine learning, environmentally triggered rules can deliver timely and relevant alerts. These results align with McCrickard et al.'s (2003) principle that notifications should only appear when they provide clear value and respect users' attention flow. The positive user feedback, particularly regarding the low frequency and concise messaging, suggests that rule-based timing can reduce the cognitive burden typically associated with proactive systems. Furthermore, Lexi-Jarvis's reliance on environmental triggers reflects Okoshi et al.'s (2015) findings that aligning notifications with user context decreases perceived intrusiveness.

However, the inconsistencies on iOS reveal significant platform-dependent constraints, echoing broader concerns in the PWA literature about cross-platform reliability. These results indicate that while rule-based systems are feasible as transitional prototypes, improvements in token persistence, installation accessibility, and content personalization will be necessary before such systems can achieve fully seamless, JARVIS-like proactive assistance.

---

5. Discussion

5.1 Contribution to Existing Literature and Prior Research

The Lexi-Jarvis project contributes to several areas identified as research gaps in Section 2.5, while also reinforcing and extending findings from prior studies. First, it demonstrates that a functional, replicable proactive notification system can be developed using open-source APIs and publicly available tools, contrasting sharply with proprietary ecosystems such as Google Assistant and Alexa, mentioned by Mehrotra and Musolesi (2017), which remain largely inaccessible for academic experimentation.

By storing multiple user tokens in MongoDB and synchronizing notifications across devices, Lexi-Jarvis also addresses the lack of academic prototypes focusing on multi-user, cross-platform delivery. Although its success rate on iOS was relatively low, the architecture proves conceptually scalable and adaptable for larger user bases.

Furthermore, Lexi-Jarvis aligns with Bellotti et al. 's (2008) argument that early context-aware systems, such as Magitti, can serve as effective transitional prototypes toward fully intelligent, AI-driven assistants. Its performance shows that incremental enhancements—such as expanding rule sets or integrating machine-learning models—offer a practical and resource-efficient path forward for early-stage research.

The results are consistent with findings from previous studies on cross-platform inconsistencies in PWAs. Similar to observations by Panwar (2024), which highlight iOS's restrictive handling of service workers, Lexi-Jarvis's 65% success rate on iOS underscores the challenge of achieving uniform performance across heterogeneous platforms. User feedback also echoes established theories in interruption science. Testers found timely weather and UV alerts valuable, whereas news notifications were sometimes perceived as irrelevant, confirming Mehrotra et al.'s (2016) warning that reduced content relevance increases the risk of notification fatigue. These consistencies indicate that Lexi-Jarvis is not only technically feasible but also aligned with existing theoretical frameworks, further validating its potential as a research tool.

5.2 Research Implications

The implications of this research extend beyond the scope of this prototype. From a human–computer interaction perspective, Lexi-Jarvis demonstrates how attention-aware design principles can be implemented with minimal computational resources, offering a baseline for future studies exploring proactive user assistance. In practical terms, similar lightweight systems could be adapted for applications such as healthcare alerts, disaster early-warning systems, or environmental monitoring, particularly in resource-constrained settings where deploying full-scale AI models is impractical. Finally, Lexi-Jarvis provides a foundation for bridging toward more advanced AI-driven systems. By incrementally integrating adaptive machine-learning algorithms to predict optimal notification timing and personalize content, future iterations could move closer to realizing the "mini-JARVIS" vision described in Chapter 1, where proactive assistants appear exactly when needed.

5.3 Limitations and Future Work

5.3.1 Limitations

Despite its contributions, Lexi-Jarvis faced several limitations:

1. iOS Performance Issues – Token expiration and manual installation requirements significantly reduced reliability, limiting its feasibility as a universal cross-platform solution.
2. Limited User Testing – The experiment involved only four testers, restricting the generalizability of user feedback and making it difficult to draw statistically significant conclusions.
3. Local Deployment – Server deployment failures on Render prevented large-scale, real-world testing, constraining the evaluation to controlled conditions.
4. Simplified Rules – Lexi-Jarvis used only four basic triggers; more complex scenarios, such as combining multiple APIs for richer context inference, were not explored.

5.4.2 Future work

Future iterations of Lexi-Jarvis could address these limitations through:

- Improved Cross-Platform Support – Investigating native app wrappers (e.g., Capacitor or React Native) or optimizing service workers for iOS to achieve more stable token persistence.
- Adaptive Notification Timing – Incorporating machine-learning models to learn user preferences and predict ideal notification moments, following methods such as those proposed by Mehrotra and Musolesi (2017).
- Expanded User Studies – Deploying the server and testing with larger, more diverse participant groups to evaluate user experience and acceptance statistically.
- Enhanced Content Personalization – Allowing users to customize notification categories to reduce notification fatigue and improve perceived relevance (Mehrotra et al., 2016).

---

6. Conclusion

This dissertation explored the feasibility of developing a lightweight, rule-based proactive notification system to deliver API-triggered alerts across multiple devices, with the broader vision of moving toward context-aware AI assistants inspired by JARVIS in *Iron Man*. The Lexi-Jarvis project was designed as a proof-of-concept notification prototype, emphasizing technical feasibility, replicability, and multi-user scalability.

The project successfully implemented a Progressive Web App integrated with Firebase Cloud Messaging, MongoDB, and open-source APIs, including weather, UV index, AQI, and news data sources. The system achieved high notification success rates on macOS (95%) and Windows (92.5%), validating that even simple, rule-based triggers can reliably provide timely and relevant notifications. These results align with established principles in HCI research, which emphasize that notifications must deliver clear value while respecting user attention flow. User feedback further supported this, as testers described the weather and UV alerts as timely and useful, and the minimalist messaging as non-intrusive.

In addition to confirming the feasibility of this approach, Lexi-Jarvis contributes to the academic understanding of proactive notification systems in several ways. First, it demonstrates that an open-source, replicable prototype can serve as a valuable academic tool in a field dominated by proprietary, closed-source systems such as Google Assistant and Alexa. Second, it introduces multi-user, cross-platform functionality, which is rarely addressed in academic prototypes, by enabling simultaneous notifications through a centralized token management system in MongoDB. Third, it highlights the viability of rule-based systems as transitional prototypes, supporting arguments in prior literature that such lightweight approaches can act as a practical stepping stone before adopting complex machine-learning models.

However, several technical and methodological challenges emerged during testing. iOS presented significant limitations, including token expiration, manual home-screen installation requirements, and inconsistent URL handling, which reduced its proactive nature. Deployment constraints also restricted the system to a local testing environment, preventing large-scale real-world evaluation. Furthermore, the system’s reliance on a small set of environmental rules limited its ability to deliver more personalized or adaptive notifications, and user testing was confined to a very small sample size.

These findings nevertheless offer clear directions for future work. Enhancing cross-platform reliability—potentially through native wrappers or hybrid frameworks—would be essential for achieving consistent multi-device performance. Integrating adaptive machine-learning algorithms to optimize notification timing and personalize content could significantly increase user acceptance and reduce notification fatigue. Expanding testing to larger, more diverse user groups would allow for more robust evaluation of user experience, while deploying the system on cloud servers would test its scalability in real-world contexts.

Overall, the Lexi-Jarvis project demonstrates that even within limited time and resources, a lightweight, rule-based proactive notification system can make meaningful progress toward more autonomous, context-aware HCI systems. By validating the technical feasibility and identifying key challenges, this research provides a foundation for future developments in proactive notification design. More importantly, it represents an incremental but concrete step toward realizing the vision of an intelligent assistant that appears exactly when needed—a small but significant stride toward the long-envisioned “mini-JARVIS.”

References

Bellotti, Victoria, Bo Begole, Ed H. Chi, Nicolas Ducheneaut, Ji Fang, Ellen Isaacs, Tracy King, et al. “Activity-Based Serendipitous Recommendations with the MAGITTI Mobile Leisure Guide.” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, April 6, 2008, 1157–66. https://doi.org/10.1145/1357054.1357237.

Biørn-Hansen, Andreas, Tim A. Majchrzak, and Tor-Morten Grønli. “Progressive Web Apps: The Possible Web-Native Unifier for Mobile Development.” *Proceedings of the 13th International Conference on Web Information Systems and Technologies*, 2017. https://doi.org/10.5220/0006353703440351.

Iqbal, Shamsi T, and Brian P Bailey. Effects of intelligent notification management on users and their tasks , April 5, 2008. https://dl.acm.org/doi/10.1145/1357054.1357070.

McCrickard, D. Scott, C. M. Chewar, Jacob P. Somervell, and Ali Ndiwalana. “A Model for Notification Systems Evaluation—Assessing User Goals for Multitasking Activity.” *ACM Transactions on Computer-Human Interaction* 10, no. 4 (December 2003): 312–38. https://doi.org/10.1145/966930.966933.

McCrickard, D Scott, and Christa M Chewar. “Designing Attention-Centric Notification Systems: Five HCI Challenges.” *Cognitive Systems*, August 15, 2006, 77–100. https://doi.org/10.4324/9781410617088-10.

Mehrotra, Abhinav, and Mirco Musolesi. *Intelligent Notification Systems: A Survey of the State of the Art and Research Challenges*, 2017. https://doi.org/10.48550/arXiv.1711.10171 .

Mehrotra, Abhinav, Veljko Pejovic, Jo Vermeulen, Robert Hendley, and Mirco Musolesi. “My Phone and Me.” *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, May 7, 2016, 1021–32. https://doi.org/10.1145/2858036.2858566.

Okoshi, Tadashi, Julian Ramos, Hiroki Nozaki, Jin Nakazawa, Anind K. Dey, and Hideyuki Tokuda. “Attelia: Reducing User’s Cognitive Load Due to Interruptive Notifications on Smart Phones.” *2015 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, March 2015, 96–104. https://doi.org/10.1109/percom.2015.7146515.

Okoshi, Tadashi, Kota Tsubouchi, Masaya Taji, Takanori Ichikawa, and Hideyuki Tokuda. “Attention and Engagement-Awareness in the Wild: A Large-Scale Study with Adaptive Notifications.” *2017 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, March 2017, 100–110. https://doi.org/10.1109/percom.2017.7917856.

Panwar, Vijay. “Leveraging Progressive Web Apps (PWAs) for Enhanced User Experience and Performance: A Comprehensive Analysis.” *International Journal of Management IT and Engineering* 14 (April 4, 2024). https://doi.org/ISSN: 2249-0558.

Pejovic, Veljko, and Mirco Musolesi. “InterruptMe.” *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, September 13, 2014, 897–908. https://doi.org/10.1145/2632048.2632062.

Pielot, Martin, and Luz Rello. “Productive, Anxious, Lonely.” *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services*, September 4, 2017, 1–11. https://doi.org/10.1145/3098279.3098526.

Pielot, Martin, and Luz Rello. “The Do Not Disturb Challenge.” *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, April 18, 2015, 1761–66. https://doi.org/10.1145/2702613.2732704.

Shin, Donghee, and Yong Jin Park. “Role of Fairness, Accountability, and Transparency in Algorithmic Affordance.” *Computers in Human Behavior* 98 (September 2019): 277–84. https://doi.org/10.1016/j.chb.2019.04.019.

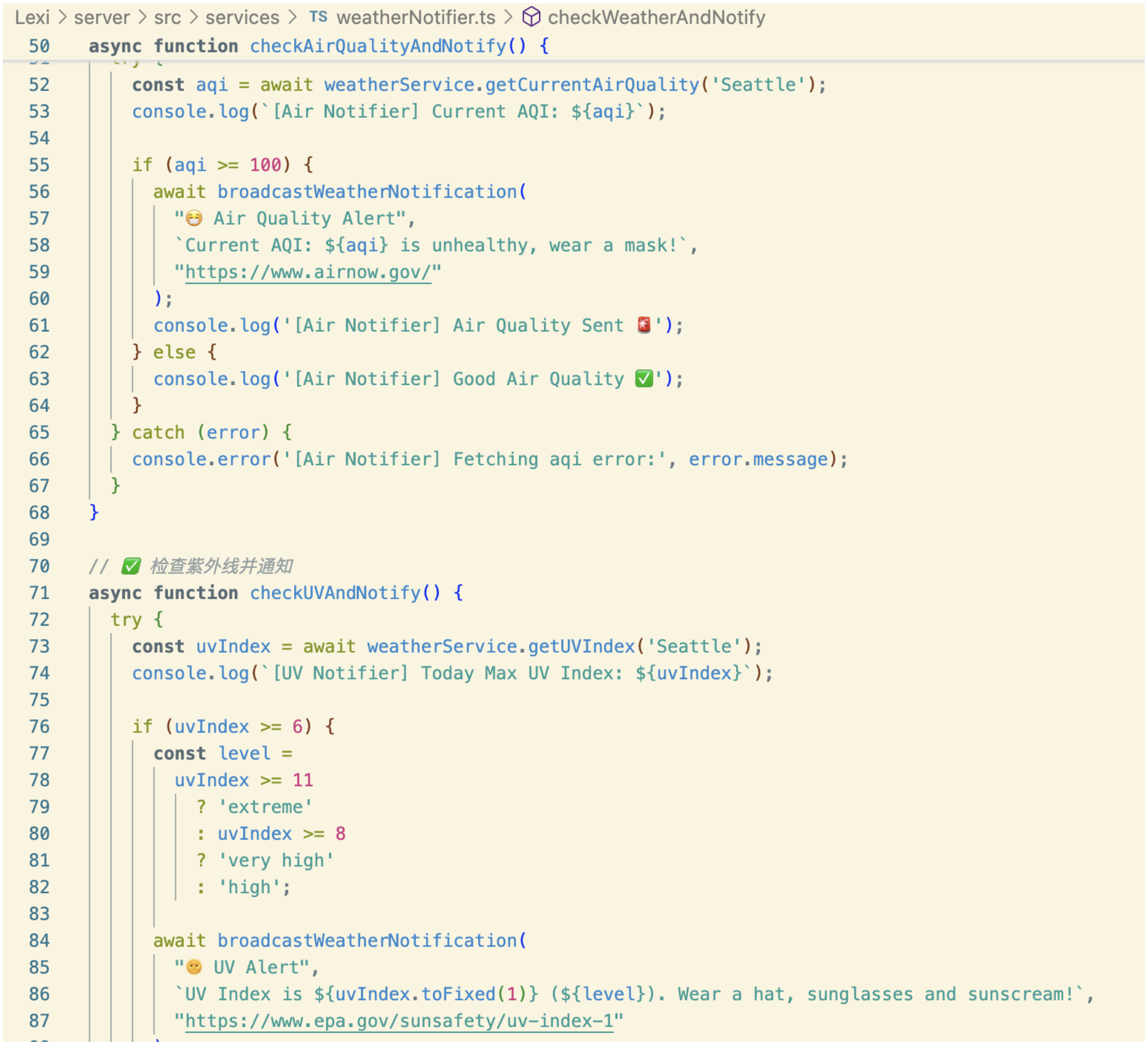
Sorrel, Charlie. “Notifications Finally Make iPhone and iPad Web Apps Worth Using.” Lifewire, February 22, 2023. https://www.lifewire.com/notifications-finally-make-iphone-and-ipad-web-apps-worth-using-7112381.

“Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage : Web.Dev.” web.dev, 2017. https://web.dev/case-studies/twitter.

Zhang, Mengyuan, Lei Yang, Shibo He, Ming Li, and Junshan Zhang. “Privacy-Preserving Data Aggregation for Mobile Crowdsensing with Externality: An Auction Approach.” *IEEE/ACM Transactions on Networking* 29, no. 3 (June 2021): 1046–59. https://doi.org/10.1109/tnet.2021.3056490.

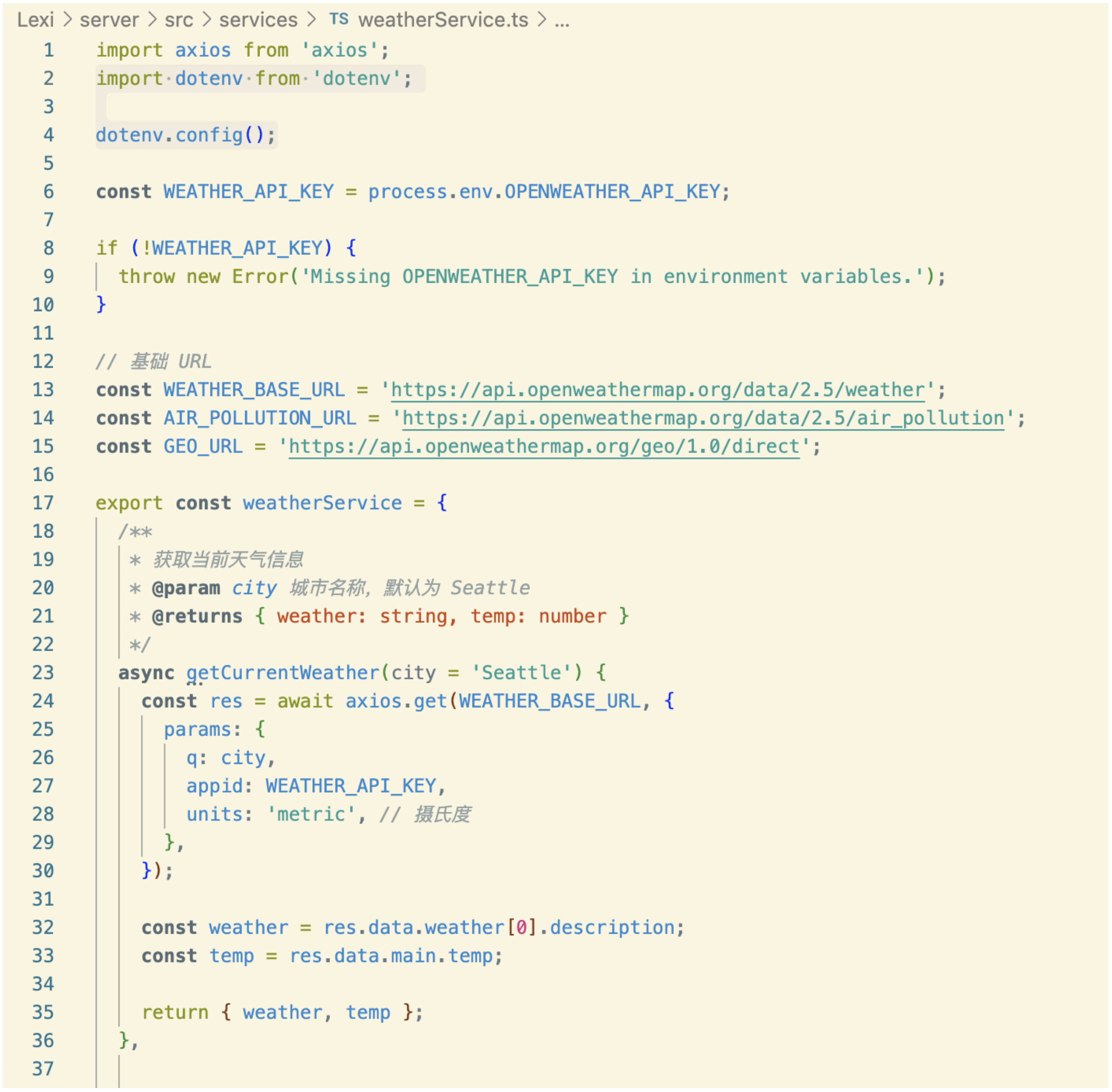
Appendices

Appendix A: Key Code Snippets (Notification Trigger Function)



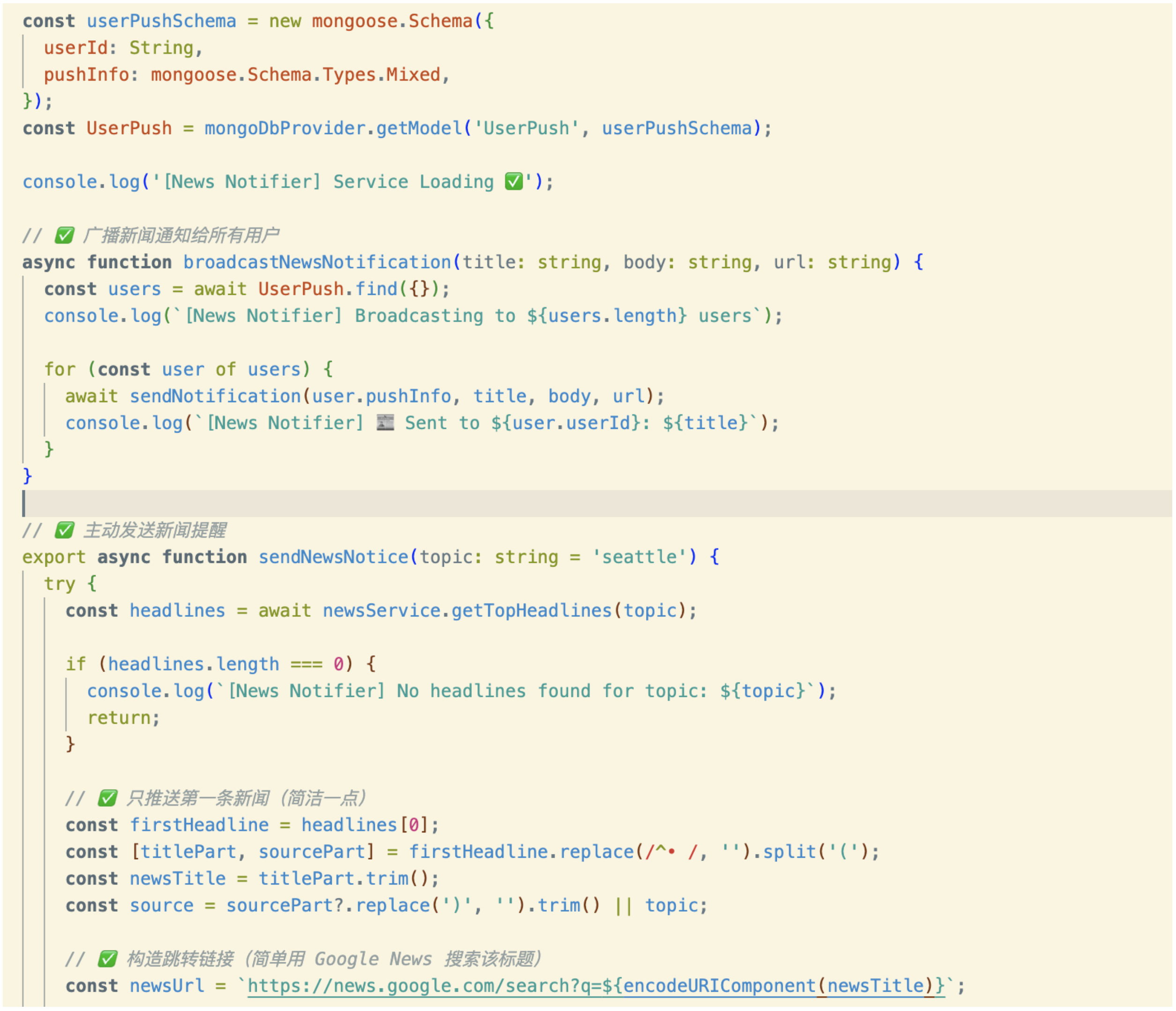
Lexi > server > src > services > TS weatherNotifier.ts > checkWeatherAndNotify

```typescript
async function checkAirQualityAndNotify() {
        const aqi = await weatherService.getCurrentAirQuality('Seattle');
        console.log(`[Air Notifier] Current AQI: ${aqi}`);

        if (aqi >= 100) {
            await broadcastWeatherNotification(
                "😷 Air Quality Alert",
                `Current AQI: ${aqi} is unhealthy, wear a mask!`,
                "https://www.airnow.gov/"
            );
            console.log('[Air Notifier] Air Quality Sent 🚨');
        } else {
            console.log('[Air Notifier] Good Air Quality ✅');
        }
    } catch (error) {
        console.error('[Air Notifier] Fetching aqi error:', error.message);
    }
}

// ✅ 检查紫外线并通知
async function checkUVAndNotify() {
    try {
        const uvIndex = await weatherService.getUVIndex('Seattle');
        console.log(`[UV Notifier] Today Max UV Index: ${uvIndex}`);

        if (uvIndex >= 6) {
            const level =
                uvIndex >= 11
                    ? 'extreme'
                    : uvIndex >= 8
                    ? 'very high'
                    : 'high';

            await broadcastWeatherNotification(
                "🌞 UV Alert",
                `UV Index is ${uvIndex.toFixed(1)} (${level}). Wear a hat, sunglasses and sunscream!`,
                "https://www.epa.gov/sunsafety/uv-index-1"
```

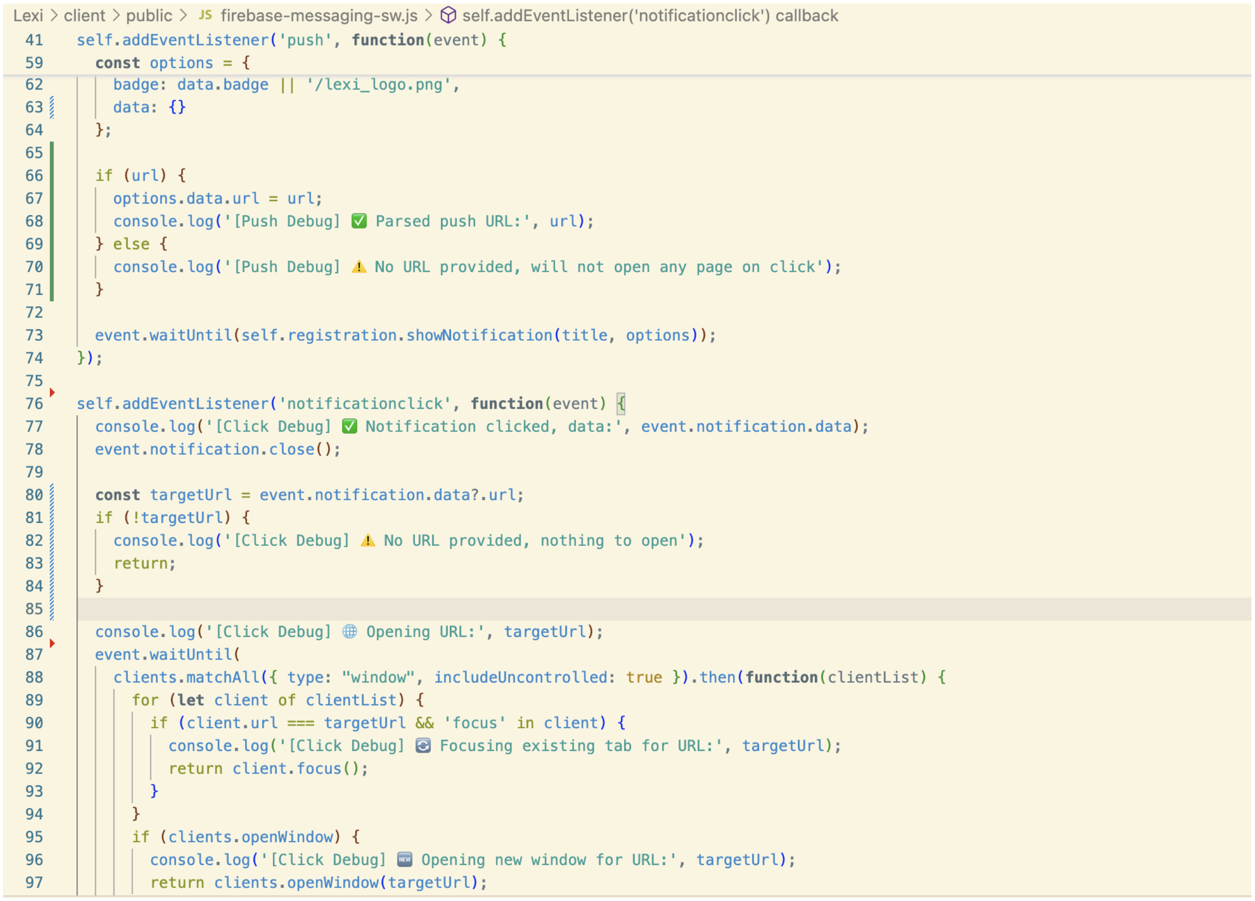

Lexi > server > src > services > TS weatherService.ts > ...

```typescript
import axios from 'axios';
import dotenv from 'dotenv';

dotenv.config();

const WEATHER_API_KEY = process.env.OPENWEATHER_API_KEY;

if (!WEATHER_API_KEY) {
  throw new Error('Missing OPENWEATHER_API_KEY in environment variables.');
}

// 基础 URL
const WEATHER_BASE_URL = 'https://api.openweathermap.org/data/2.5/weather';
const AIR_POLLUTION_URL = 'https://api.openweathermap.org/data/2.5/air_pollution';
const GEO_URL = 'https://api.openweathermap.org/geo/1.0/direct';

export const weatherService = {
  /**
   * 获取当前天气信息
   * @param city 城市名称，默认为 Seattle
   * @returns { weather: string, temp: number }
   */
  async getCurrentWeather(city = 'Seattle') {
    const res = await axios.get(WEATHER_BASE_URL, {
      params: {
        q: city,
        appid: WEATHER_API_KEY,
        units: 'metric', // 摄氏度
      },
    });

    const weather = res.data.weather[0].description;
    const temp = res.data.main.temp;

    return { weather, temp };
  },

```

Appendix B: Key Code Snippets (News Trigger and URL Transmission)



```
const userPushSchema = new mongoose.Schema({
  userId: String,
  pushInfo: mongoose.Schema.Types.Mixed,
});
const UserPush = mongoDbProvider.getModel('UserPush', userPushSchema);

console.log('[News Notifier] Service Loading ✅');

// ✅ 广播新闻通知给所有用户
async function broadcastNewsNotification(title: string, body: string, url: string) {
  const users = await UserPush.find({});
  console.log(`[News Notifier] Broadcasting to ${users.length} users`);

  for (const user of users) {
    await sendNotification(user.pushInfo, title, body, url);
    console.log(`[News Notifier] 📰 Sent to ${user.userId}: ${title}`);
  }
}

// ✅ 主动发送新闻提醒
export async function sendNewsNotice(topic: string = 'seattle') {
  try {
    const headlines = await newsService.getTopHeadlines(topic);

    if (headlines.length === 0) {
      console.log(`[News Notifier] No headlines found for topic: ${topic}`);
      return;
    }

    // ✅ 只推送第一条新闻（简洁一点）
    const firstHeadline = headlines[0];
    const [titlePart, sourcePart] = firstHeadline.replace(/^• /, '').split('(');
    const newsTitle = titlePart.trim();
    const source = sourcePart?.replace(')', '').trim() || topic;

    // ✅ 构造跳转链接（简单用 Google News 搜索该标题）
    const newsUrl = `https://news.google.com/search?q=${encodeURIComponent(newsTitle)}`;
```


Lexi > client > public > JS firebase-messaging-sw.js > self.addEventListener('notificationclick') callback

```
self.addEventListener('push', function(event) {
  const options = {
    badge: data.badge || '/lexi_logo.png',
    data: {}
  };

  if (url) {
    options.data.url = url;
    console.log('[Push Debug] ✅ Parsed push URL:', url);
  } else {
    console.log('[Push Debug] ⚠️ No URL provided, will not open any page on click');
  }

  event.waitUntil(self.registration.showNotification(title, options));
});

self.addEventListener('notificationclick', function(event) {
  console.log('[Click Debug] ✅ Notification clicked, data:', event.notification.data);
  event.notification.close();

  const targetUrl = event.notification.data?.url;
  if (!targetUrl) {
    console.log('[Click Debug] ⚠️ No URL provided, nothing to open');
    return;
  }

  console.log('[Click Debug] 🌐 Opening URL:', targetUrl);
  event.waitUntil(
    clients.matchAll({ type: "window", includeUncontrolled: true }).then(function(clientList) {
      for (let client of clientList) {
        if (client.url === targetUrl && 'focus' in client) {
          console.log('[Click Debug] 🔄 Focusing existing tab for URL:', targetUrl);
          return client.focus();
        }
      }
      if (clients.openWindow) {
        console.log('[Click Debug] 🆕 Opening new window for URL:', targetUrl);
        return clients.openWindow(targetUrl);
```