American Journal of Engineering Research (AJER)
e-ISSN: 2320-0847 p-ISSN : 2320-0936
Volume-14, Issue-1, pp-99-106
www.ajer.org
Research Paper Open Access

Mobile SEO Application Development: Leveraging SwiftUI and Alamofire for Domain Insights

Ekaterine Papava
[1]*AOI, Tbilisi, Georgia*
Georgian Technical University, Faculty of Informatics and Control Systems, PhD Student, Progrem: Informatics

ABSTRACT: *In the modern digital landscape, tools that facilitate quick and effective SEO analysis are invaluable for businesses aiming to enhance their online presence. This research article details the development of a Domain Checker App using SwiftUI and Alamofire, designed to fetch and display SEO-related metrics such as Organic Keywords Count, Internal Link Quality, and Headings from a given domain. The application emphasizes user-friendly design, robust error handling, and efficient data fetching, showcasing its relevance for both developers and end-users. This paper explores the architecture, code implementation, and practical use cases, providing a blueprint for similar applications..*

KEYWORDS: *SwiftUI, Alamofire, Swift Programming, JSON Parsing, API Integration, SEO Analysis, Mobile App Development, Language-Specific SEO Solutions, Natural Language Processing, Data Visualization*

Date of Submission: 13-01-2025 Date of acceptance: 27-01-2025

I. INTRODUCTION

Search engine optimization (SEO) tools play a pivotal role in improving website visibility and driving organic traffic, which is essential for businesses operating in an increasingly competitive online ecosystem. The rapid evolution of mobile technologies demands SEO applications that are not only functional but also user-centric and adaptive to varying user needs. This study introduces a Domain Checker App built with SwiftUI and Alamofire, offering an intuitive platform to fetch, parse, and display critical SEO metrics for any given domain.

The app's standout features include dynamic data parsing, real-time API integration, and robust error-handling mechanisms, which ensure reliability and usability for end-users. By leveraging Alamofire's RESTful API capabilities and SwiftUI's declarative UI design, the application achieves seamless performance while providing actionable insights for SEO professionals. Additionally, this research explores the potential of integrating Natural Language Processing (NLP) to address challenges in language-specific SEO, particularly for languages with complex grammatical structures, such as Georgian. This novel approach extends the app's utility to a wider audience and opens avenues for future advancements in SEO-focused app development.

The following sections outline the architecture, core functionalities, and practical applications of the Domain Checker App. Furthermore, the study highlights opportunities for enhancing the tool's capabilities, including the integration of advanced analytics, NLP-based optimizations, and support for diverse user demographics, ensuring the app's relevance in an ever-changing digital landscape.

II. CHALLENGES OF SEO AND NLP SUPPORT FOR THE GEORGIAN LANGUAGE

Challenges of SEO and the Role of NLP in Emerging Languages:
While SEO tools are widely available for global languages like English and Spanish, challenges persist for less-supported languages, such as Georgian. The Georgian language, with its unique Mkhedruli alphabet and grammatical complexity, presents significant hurdles for indexing, keyword analysis, and content optimization. These limitations stem from inadequate support in widely-used SEO platforms, which struggle with non-Latin scripts and fail to account for Georgian linguistic structures.

Addressing these gaps involves integrating Natural Language Processing (NLP) techniques tailored to Georgian's unique features. By enhancing tokenization, stemming, and keyword extraction processes, businesses can improve SEO strategies and ensure content resonates with local audiences. Future developments in this area aim to create localized datasets, improve API encoding for Georgian script, and foster collaboration with global SEO platforms. These efforts will support the broader goal of developing accessible tools for emerging languages, ensuring inclusivity in the digital landscape.

III. OBJECTIVES

The primary objectives of this research include:

- Developing a user-friendly interface for domain input and data display.
- Implementing a reliable API integration using Alamofire.
- Handling errors gracefully to improve user experience.
- Showcasing the potential of SwiftUI in building responsive and interactive applications.
- Highlighting the potential application of NLP in handling complex linguistic structures, such as Georgian, as a future research direction.

IV. METHODOLOGY

Overview of the Application

The Domain Checker App consists of two main components:

- **ContentView:** The entry point for user interaction, where domains are entered, and data fetching is initiated.
- **DashboardView:** Displays fetched data, including Organic Keywords Count, Internal Links, and Headings, in an organized layout.

V. KEY FEATURES

This app demonstrates core functionalities vital for modern application development:

Fetching Data from an API:

- API integration is a critical component in modern apps that rely on external data. The Domain Checker App uses Alamofire to send a POST request to an SEO data API. Alamofire simplifies network requests by providing a concise syntax and built-in functionalities like parameter encoding and header management.
- The app sends a domain as input and expects JSON data in response, which is then parsed into a usable format.

Dynamic Parsing and Display:

- Parsing the JSON response is a multi-step process, where data is decoded into Swift structures using Codable. The app processes metrics such as Organic Keywords Count, Internal Links, and Headings dynamically. The parsed data is displayed in the user interface in real time, ensuring users receive actionable insights quickly.
- By leveraging SwiftUI’s declarative syntax, the app dynamically updates the interface based on the received data.

Error and Loading State Handling:

- User experience is enhanced by managing potential errors and providing feedback during loading. The app ensures that users are informed when data fetching fails or input is invalid.
- During data fetching, a ProgressView is displayed, giving users a clear visual indicator of the ongoing process. If an error occurs (e.g., invalid domain, API failure), a descriptive error message is displayed.

VI. DETAILED BREAKDOWN OF KEY FEATURES

User Input

TextField for Domain Input:

- The app includes a TextField component that allows users to enter the domain they wish to analyze. This field is styled for clarity and usability, featuring rounded borders and appropriate padding.
- Input validation is implemented to ensure that the domain field is not empty before the fetch operation can proceed. This reduces the likelihood of unnecessary API requests.

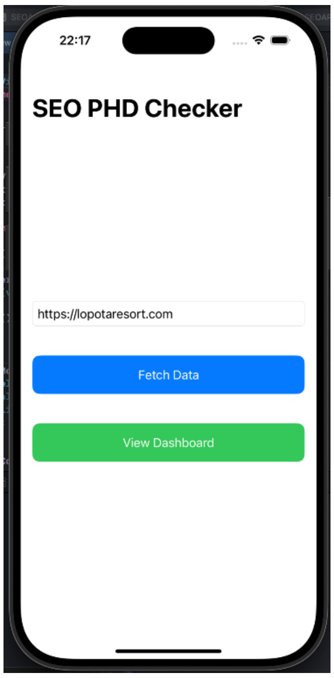


Fig.1 TextField for Domain Input on the Mobile App

Fetch Button:

- A button is provided to trigger the data-fetching operation. The button dynamically updates its state based on the validity of the user input and loading status. For example, the button is disabled if the domain field is empty or if data is being fetched.

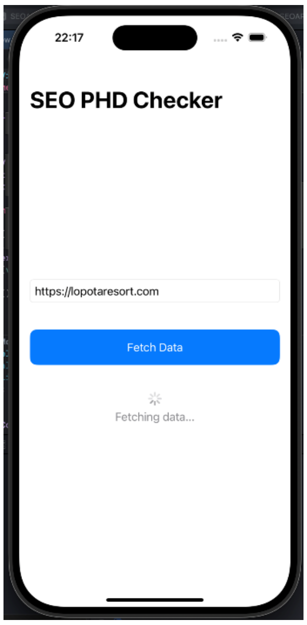


Fig.2 Data-fetching operation

Data Fetching

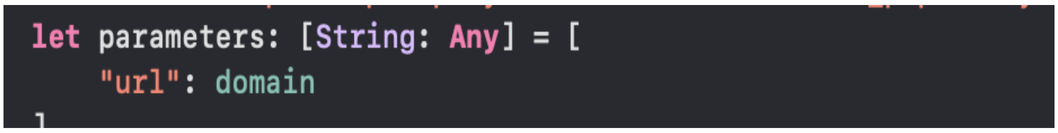
API Integration with Alamofire:

The core functionality of the app lies in its ability to fetch data from an API. Alamofire simplifies this process, providing a high-level abstraction for making HTTP requests. The fetchData function is responsible for sending a POST request to the SEO data API with the user-entered domain. Key aspects include:

1. **Parameter Passing:**

- The function sends the domain as a parameter within the request body. This ensures that the API can correctly identify and process the input domain.

Example parameter structure:

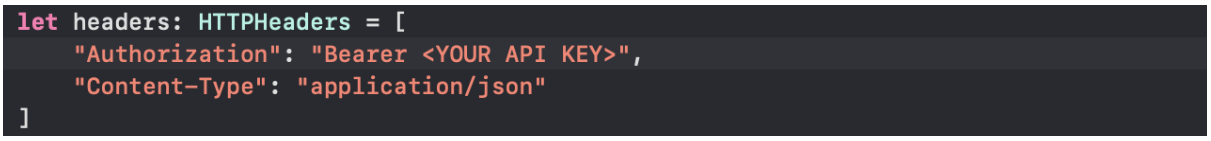


```
let parameters: [String: Any] = [
    "url": domain
]
```

Fig.3 Function sending the domain as a parameter within the request body

2. **Header Management:**
- Secure communication is ensured by including necessary headers such as authorization tokens and content type.

Example header structure:

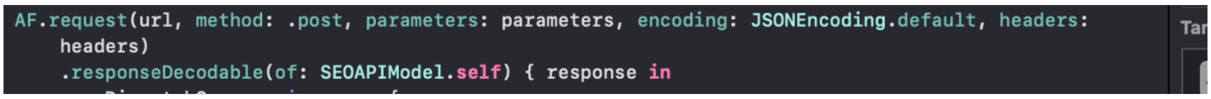


```
let headers: HTTPHeaders = [
    "Authorization": "Bearer <YOUR API KEY>",
    "Content-Type": "application/json"
]
```

Fig.4 Alamofire's AF.request for Headers

3. **POST Request:**
- Alamofire's AF.request method is used to send the request. The method takes the API endpoint, HTTP method, parameters, and headers as arguments.

Example:



```
AF.request(url, method: .post, parameters: parameters, encoding: JSONEncoding.default, headers:
    headers)
    .responseDecodable(of: SEOAPIModel.self) { response in
```

Fig.5 Alamofire's AF.request

4. **Error Handling**

Handling potential errors is critical for providing a smooth user experience. The app anticipates various failure scenarios, such as:

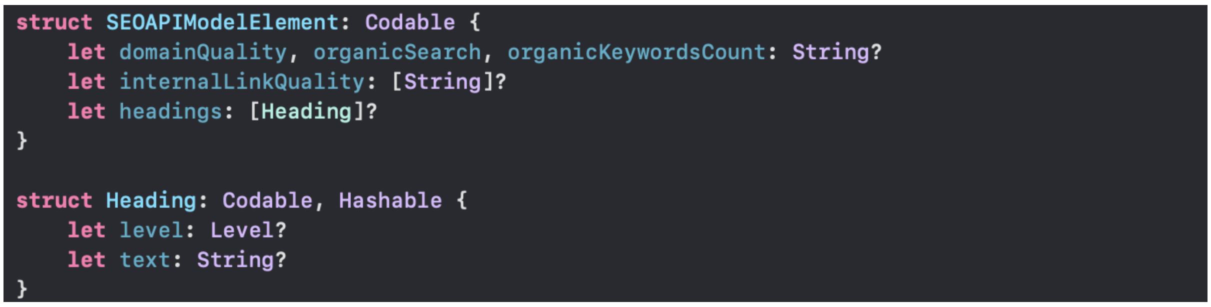
1. **Invalid Domain Input:**
- If the user enters an empty or incorrectly formatted domain, the app prevents the request from being sent and displays an appropriate error message.

2. **Network Errors**:
- Failures such as timeouts or lack of internet connectivity are captured and communicated to the user via descriptive error messages.

3. **API Errors:**
- Errors returned by the API (e.g., invalid domain or server-side issues) are parsed and displayed to the user.
- Example error message display:
- errorMessage = error.localizedDescription

5. **JSON Parsing**

The JSON response from the API is decoded into a predefined Swift structure using the Codable protocol. This ensures seamless parsing and usage of the response data within the app. For example:



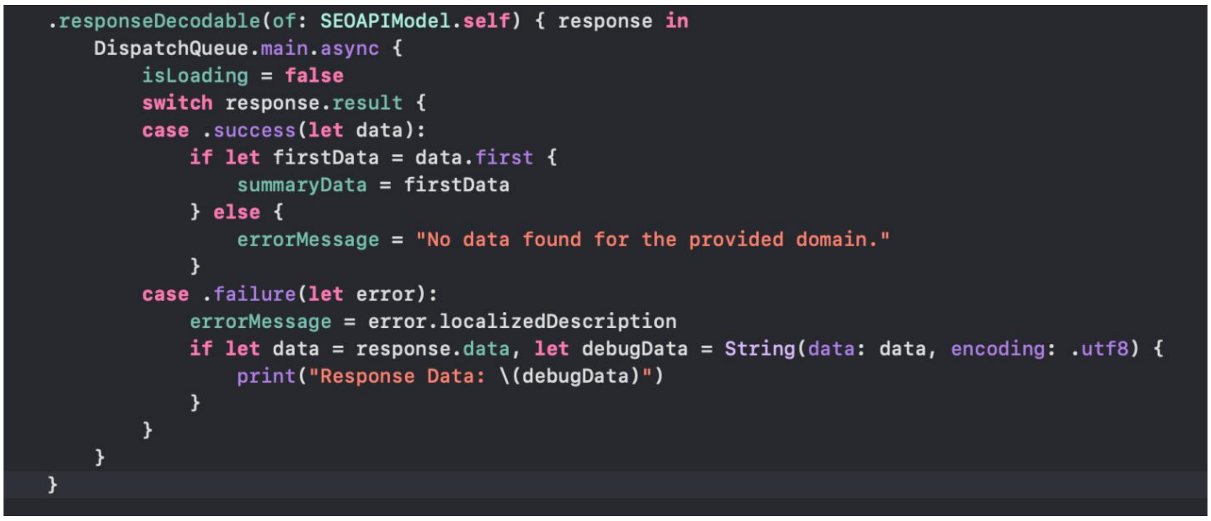
```
struct SEOAPIModelElement: Codable {
    let domainQuality, organicSearch, organicKeywordsCount: String?
    let internalLinkQuality: [String]?
    let headings: [Heading]?
}

struct Heading: Codable, Hashable {
    let level: Level?
    let text: String?
}
```

Fig.6 JSON Parsing

6. **Response Structure:**

Decoding Process:



```
    .responseDecodable(of: SEOAPIModel.self) { response in
        DispatchQueue.main.async {
            isLoading = false
            switch response.result {
            case .success(let data):
                if let firstData = data.first {
                    summaryData = firstData
                } else {
                    errorMessage = "No data found for the provided domain."
                }
            case .failure(let error):
                errorMessage = error.localizedDescription
                if let data = response.data, let debugData = String(data: data, encoding: .utf8) {
                    print("Response Data: \(debugData)")
                }
            }
        }
    }
```

Fig.7 Decoding Process

7. **Loading State Management**

During data fetching, a ProgressView is displayed to indicate that the app is working on the request. This visual cue improves user experience by providing immediate feedback.

8. **Data Display**

DashboardView: Displaying SEO Data

The DashboardView serves as the visual representation of the fetched SEO metrics. It organizes data into structured sections, ensuring clarity and accessibility for the user. The key features and design elements include:

1. **List Display:**
- Data is displayed in a list format, making it easy for users to navigate through different metrics such as:
- Organic Keywords Count.
- Internal Links.
- Headings grouped by their levels (e.g., H1, H2, H3).

2. **Dynamic Sections:**
- The list is broken into distinct sections, with each section dynamically populated based on the fetched data.

3. **Internal Links:**
- Internal links are displayed with truncation to manage long URLs, ensuring a clean and organized presentation.
- For example:

- Text(link).lineLimit(1).truncationMode(.tail)

This approach ensures that even extensive link lists remain visually tidy without sacrificing usability.

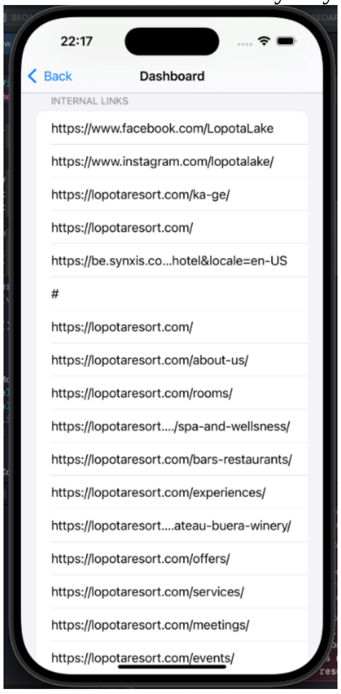

22:17
Back Dashboard
INTERNAL LINKS
https://www.facebook.com/LopotaLake
https://www.instagram.com/lopotalake/
https://lopotaresort.com/ka-ge/
https://lopotaresort.com/
https://be.synxis.co...hotel&locale=en-US
#
https://lopotaresort.com/
https://lopotaresort.com/about-us/
https://lopotaresort.com/rooms/
https://lopotaresort..../spa-and-wellsness/
https://lopotaresort.com/bars-restaurants/
https://lopotaresort.com/experiences/
https://lopotaresort....ateau-buera-winery/
https://lopotaresort.com/offers/
https://lopotaresort.com/services/
https://lopotaresort.com/meetings/
https://lopotaresort.com/events/

Fig.8 Real time data display

9. **Headings Display:**

- Headings are categorized by their levels (e.g., H1, H2, H3) and displayed with accompanying text for better context.

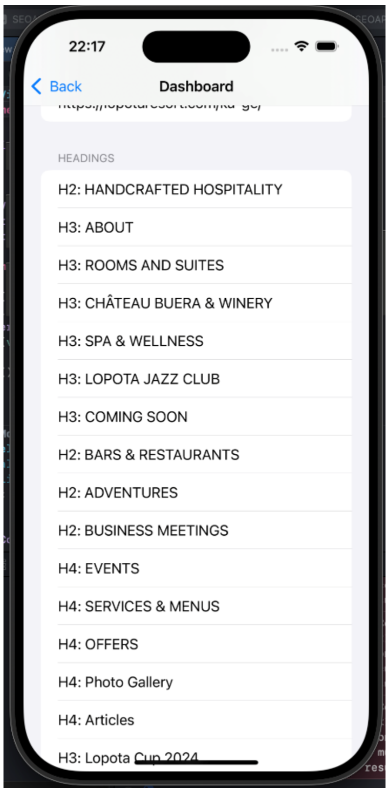

22:17
Back Dashboard
HEADINGS
H2: HANDCRAFTED HOSPITALITY
H3: ABOUT
H3: ROOMS AND SUITES
H3: CHÂTEAU BUERA & WINERY
H3: SPA & WELLNESS
H3: LOPOTA JAZZ CLUB
H3: COMING SOON
H2: BARS & RESTAURANTS
H2: ADVENTURES
H2: BUSINESS MEETINGS
H4: EVENTS
H4: SERVICES & MENUS
H4: OFFERS
H4: Photo Gallery
H4: Articles
H3: Lopota Cup 2024

Fig.9 Heading Tags

4. **Presentation:**

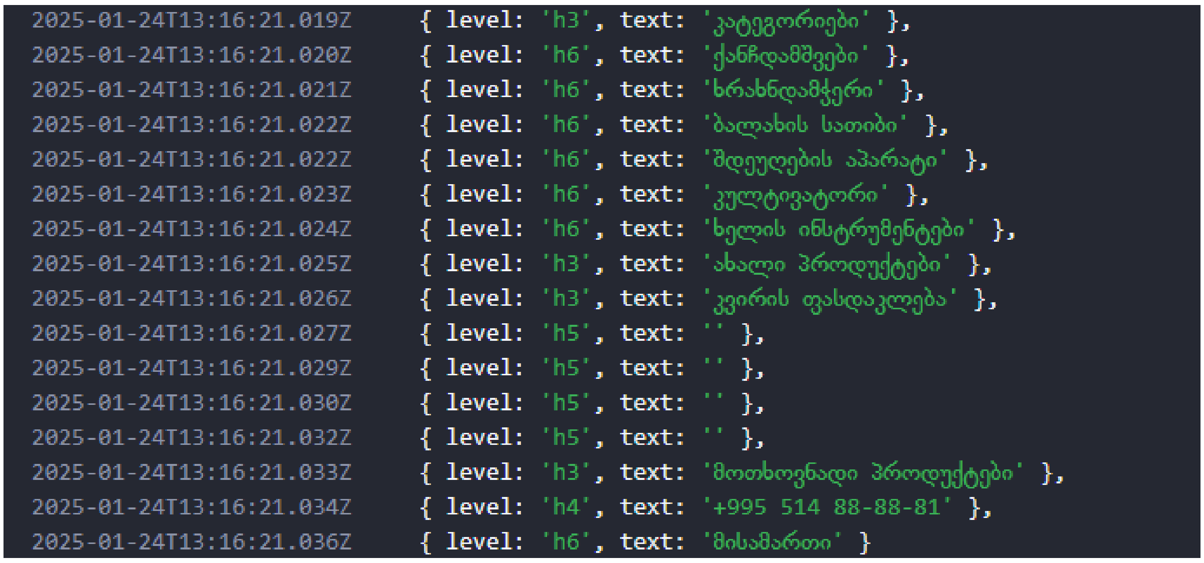
- SwiftUI's declarative syntax ensures smooth animations and transitions between different states, providing an enhanced user experience.

5. **Error States in Display:**
- If no data is available for any section (e.g., no internal links found), the app displays a friendly placeholder message like "No internal links detected for this domain."
6. **Styling and Accessibility:**
- The DashboardView incorporates accessible design principles by ensuring legible font sizes, clear contrast, and VoiceOver support.

Visual and Functional Enhancements
- **Custom Icons:** Icons are added alongside metrics to visually distinguish between sections.
- **Expandable Views:** Sections can be collapsible to allow users to focus on specific metrics without overwhelming the interface.
- **Search or Filter:** Additional functionality could be added to allow users to search or filter internal links or headings based on keywords.

VII. ONGOING RESEARCH AND API DEPLOYMENT FOR NLP IN GEORGIAN

Developing robust NLP tools for languages with unique characteristics, like Georgian, remains an area of interest for exploration on the next phase. Although this project primarily addresses SEO analysis, expanding its scope to include NLP capabilities can unlock broader functionalities. For example, tools enabling automatic keyword extraction, semantic parsing, or sentiment analysis for Georgian text could significantly improve the application's utility.



```
2025-01-24T13:16:21.019Z     { level: 'h3', text: 'კატეგორიები' },
2025-01-24T13:16:21.020Z     { level: 'h6', text: 'ქანჩდამშვები' },
2025-01-24T13:16:21.021Z     { level: 'h6', text: 'ხრახნდამჭერი' },
2025-01-24T13:16:21.022Z     { level: 'h6', text: 'ბალახის სათიბი' },
2025-01-24T13:16:21.022Z     { level: 'h6', text: 'შედუღების აპარატი' },
2025-01-24T13:16:21.023Z     { level: 'h6', text: 'კულტივატორი' },
2025-01-24T13:16:21.024Z     { level: 'h6', text: 'ხელის ინსტრუმენტები' },
2025-01-24T13:16:21.025Z     { level: 'h3', text: 'ახალი პროდუქტები' },
2025-01-24T13:16:21.026Z     { level: 'h3', text: 'კვირის ფასდაკლება' },
2025-01-24T13:16:21.027Z     { level: 'h5', text: '' },
2025-01-24T13:16:21.029Z     { level: 'h5', text: '' },
2025-01-24T13:16:21.030Z     { level: 'h5', text: '' },
2025-01-24T13:16:21.032Z     { level: 'h5', text: '' },
2025-01-24T13:16:21.033Z     { level: 'h3', text: 'მოთხოვნადი პროდუქტები' },
2025-01-24T13:16:21.034Z     { level: 'h4', text: '+995 514 88-88-81' },
2025-01-24T13:16:21.036Z     { level: 'h6', text: 'მისამართი' }
```

Fig.10 Getting Data in Georgian Language

Future research topics include:

1. **Morpho-Syntactic Analysis:** Handling Georgian's complex grammar, such as verb conjugations and compound structures, to improve linguistic insights.
2. **Dataset Creation:** Building datasets to include regional dialects, formal writing, and domain-specific vocabularies for tasks like keyword generation.
3. **NLP Model Integration:** Deploying Georgian-specific NLP models for semantic parsing or content classification within the app.
4. **Collaborative Platforms:** looking for the solutions of integration these developments into global frameworks, aiming compatibility and broader accessibility.

This research trajectory ensures that the app can evolve beyond its initial scope—not only as a mobile application providing essential SEO analysis tools but also as an innovative platform available on smartphones. This advancement could drive groundbreaking solutions for Georgian-language SEO and NLP challenges.

VIII. CONCLUSION

The Domain Checker App effectively demonstrates the integration of SwiftUI and Alamofire to build a responsive, user-friendly mobile application. Key functionalities, such as dynamic data parsing, error handling, and real-time UI updates, ensure a robust user experience, making it a practical tool for SEO professionals on the go.

Future iterations could expand the app's capabilities by integrating additional SEO metrics, enhancing data visualization, and exploring the inclusion of NLP features to support complex linguistic structures, further increasing its versatility and global applicability. This app offers a solid foundation for mobile-first SEO analysis, with the potential to evolve and meet the growing needs of users in an increasingly digital world.

REFERENCES

[1]. Build Mobile Apps with SwiftUI and Firebase: Learn SwiftUI and Firebase by Building Real-World Applications Communicating with a Backend. 2023. doi: 10.1007/978-1-4842-9452-9.
[2]. Meurer, P., "The Morphosyntactic analysis of Georgian," 2023.
[3]. L. Liu, M. Bahrami, J. Park, and C. Wei-Peng, "Web API Search: Discover Web API and Its Endpoint with Natural Language Queries," Springer, Cham, 2020, pp. 96–113. doi: 10.1007/978-3-030-59618-7_7.
[4]. Solberg Söilen, K. (2024). The E-Commerce Website and Mobile App. In: Digital Marketing. Springer Texts in Business and Economics. Springer, Cham. https://doi.org/10.1007/978-3-031-69518-6_10
[5]. Kim, J., & Lee, J. (2002). Critical design factors for successful e-commerce systems. Behaviour & Information Technology, 21(3), 185–199. https://doi.org/10.1080/0144929021000009054