Journal of Harbin Engineering University
ISSN: 1006-7043

Vol 46 No. 05
May 2025

Quickli: A Serverless Cli And Web-Based Integrated Reminder System Using Nodejs & Supabase

[1]Dr. Tapsi Nagpal , [2]Ankit, [3]Yug Upadhyay, [4]Vibha Sharma

[1]Associate Professor

Department of Computer Science & Engineering

Lingaya's Viydapeeth Faridabad,India

[2]Student Department of Computer Science

& Engineering Lingaya's Viydapeeth Faridabad,India

[3]Student Department of computer science & engineering

Lingaya's Viydapeeth Faridabad,India

[4]Student Department of Computer Science & Engineering

Lingaya's Viydapeeth Faridabad,India

***Abstract*—** In today's hybrid work environments, productivity tools must offer both accessibility and flexibility across different platforms and user preferences. This paper introduces "Quickli," an innovative cross-platform productivity management system featuring seamless integration between a Command Line Interface (CLI) and a web application. Quickli enables users to efficiently manage tasks, reminders, and notes with real-time synchronization between both interfaces. The CLI component, distributed as a global NPM package, provides rapid task management directly from the terminal, while the web interface (available at quickli.snapstay.in) offers comprehensive visual organization of productivity data. Built with ReactJS for the frontend and Supabase [10] as the backend-asa-service solution, the system offers secure authentication, rowlevel security (RLS), and comprehensive data management capabilities. Our experimental evaluation demonstrates high performance with synchronization times under 250ms, strong data consistency across platforms, and positive user engagement scores averaging 4.5/5 in beta testing. Unique to this implementation is the perfect parity between CLI and web functionalities—users can seamlessly transition between interfaces while maintaining a consistent view and management of their productivity data. This paper presents the system architecture, implementation methodology, security considerations, and performance evaluation of Quickli as a viable open-source alternative in the productivity tool landscape.

***Keywords*—** Cross-platform applications, productivity tools, command line interface, web application, real-time synchronization, task management, Supabase, ReactJS, row-level security, open-source software, NPM package.

I. INTRODUCTION

The evolution of digital workspaces has created distinct preferences among users regarding interaction paradigms. While traditional knowledge workers often gravitate toward graphical interfaces with visual organization capabilities, technical professionals frequently prefer command-line environments for their efficiency and integration with development workflows. This dichotomy has led to a fragmented productivity tool landscape, where solutions typically excel in one paradigm while providing limited functionality in the other.

Quickli addresses this fundamental challenge by creating a unified productivity ecosystem that provides equal capabilities across both interface paradigms with seamless data synchronization. The system consists of two primary components: a web application built with ReactJS [11] that offers comprehensive visual organization of productivity data, and a command-line interface distributed as an NPM package that enables rapid terminal-based productivity management. These components share a common backend infrastructure built on Supabase, which provides authentication, database, storage, and real-time synchronization capabilities.

The architectural design ensures perfect feature parity between interfaces—any action performed in one interface is immediately reflected in the other. This approach enables users to leverage each interface's strengths contextually: rapid command-line interactions during development sessions and comprehensive visual organization during planning or review phases. The seamless transition eliminates the context-switching penalties traditionally associated with using multiple productivity tools.

This paper explores the implementation details, architectural decisions, security considerations, and performance optimizations that enable Quickli's dual-interface approach. We present evaluation results from both technical metrics and user experience perspectives, demonstrating that the system achieves its design goals of providing an efficient, secure, and flexible productivity management solution that adapts to diverse user preferences without compromising functionality or data consistency.

A comprehensive architecture for cross-platform productivity management with real-time data synchronization

Implementation of a secure, user-isolated data model using Supabase's Row-Level Security

Development and distribution of a CLI tool as a global NPM package for terminal-based productivity

Web application implementation with comprehensive dashboard for visual task management

Feature parity between CLI and web interfaces with bidirectional real-time updates

Performance evaluation and user experience assessment across both interface modalities

An open-source reference implementation demonstrating the viability of dual-interface productivity systems

II. RELATED WORK

The productivity tool landscape has evolved significantly in recent years, with various approaches attempting to bridge the gap between different interface paradigms. Understanding the strengths and limitations of existing solutions provides context for Quickli's design decisions and contributions.

A. Web-Based Productivity System

Web applications have dominated the mainstream productivity space due to their accessibility and visual interface capabilities. Systems like Trello [1], Asana[2], and Todoist [3] have established comprehensive ecosystems for task management, project organization, and collaboration. These platforms excel in providing intuitive visual representations of work through kanban boards, calendars, and interactive dashboards.

However, these web-centric approaches typically offer limited integration with command-line workflows, creating friction for technical users who operate primarily in terminal environments. While some provide APIs that enable custom CLI integrations, these are generally afterthoughts rather than first-class interfaces, resulting in feature disparities and synchronization delays.

B. CLI-Based Productivity Tools

In contrast, command-line productivity tools like TaskWarrior [4] , Todo.txt [5] , and various Git-workflow [6] utilities offer text-based interfaces optimized for keyboard-driven interaction. These tools provide rapid task entry, efficient filtering, and seamless integration with development environments, making them popular among technical professionals.

The primary limitation of these CLI-focused tools is their accessibility barrier for non-technical users who prefer visual organization and intuitive interaction models. Additionally, many lack comprehensive visualization capabilities for complex project structures or temporal relationships between tasks, limiting their effectiveness for planning and overview purposes.

C. Cross-Plateform Approaches

Few solutions have successfully implemented true cross-platform approaches with equal capabilities across interfaces. GitHub [7] and GitLab [8] provide both web and CLI interfaces for version control workflows, but their productivity management features remain secondary to their core development functions. Microsoft's PowerToys [9] offers productivity enhancements for Windows but lacks cross-platform support and comprehensive web integration.

Notion has attempted to bridge this gap through its official API and community-developed CLI tools, but these implementations suffer from feature disparities and synchronization challenges due to their distributed development approach. This highlights the difficulty in maintaining consistent functionality across drastically different interface paradigms.

Quickli builds upon these foundations while addressing their limitations through a unified architecture that ensures feature parity, real-time synchronization, and secure data management across both CLI and web interfaces, creating a truly integrated productivity ecosystem.

III. METHODOLOGY

The development of Quickli followed a structured methodology focused on creating a system with perfect parity between interfaces while maintaining security, performance, and usability. This section details the architectural approach, development process, database design, and security model implementation.

A. System Overview

Quickli employs a modern architecture centered around Supabase [10] as the backend-as-a-service platform (Fig. 1). This architecture enables real-time data synchronization between the CLI and web

interface while maintaining consistent security policies and data structures.

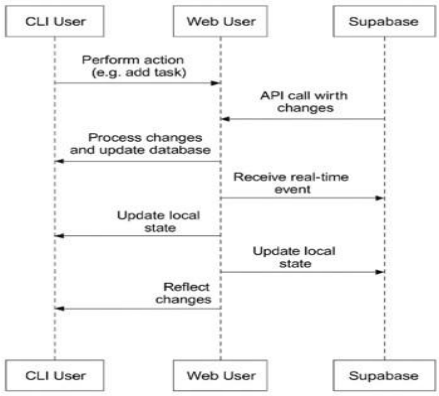

Fig. 1. High-level architecture of the Quickli system showing data flow between CLI, web interface, and Supabase backend.

The system consists of three primary components:

Web Interface: A ReactJS application hosted at quickli.snapstay.in providing visual organization and management of productivity data

CLI Tool: A Node.js [12] command-line application distributed globally via NPM (npm install -g quickli)

Backend Services: Supabase providing authentication, database, storage, and real-time capabilities

The web interface component is developed using ReactJS [11] with TailwindCSS for styling, providing a responsive and visually comprehensive productivity management environment. The interface implements a component-based architecture with reusable elements for task, reminder, and note management. State management is handled through React's Context API combined with reducers, creating predictable data flow throughout the application.

The CLI component is built with Node.js [12] and structured as a global NPM package (installed via npm install -g quickli). It implements the Commander.js pattern for command parsing and organization, allowing for intuitive command structures that mirror the functionality available in the web interface. The CLI [14] follows UNIX philosophy principles with focused commands and composable operations for efficient terminal-based productivity.

The backend services are provided by Supabase, which offers a comprehensive suite of tools including PostgreSQL [15] database, authentication services, storage capabilities, and real-time data synchronization through PostgreSQL's logical replication features. This infrastructure enables the bidirectional real-time updates that form the core of Quickli's synchronization capabilities.

The architecture ensures that all operations performed through either interface interact with the same database, maintaining perfect data consistency. When a user performs an action in one interface (e.g., completing a task via CLI), the change is immediately propagated to the backend and then reflected in real-time on any open web interface sessions. This bidirectional synchronization is achieved using Supabase's real-time capabilities, which leverage PostgreSQL's logical replication feature.

B. Development Methodology

Quickli was developed using an Agile methodology with iterative sprints focusing on key components. The development process began with extensive user research to understand the different workflow patterns and interface preferences among potential users. This research informed the feature prioritization and interface design decisions throughout the development cycle.

The database schema and Supabase configuration were established first, creating the foundation for all subsequent development. This included designing the data structures, relationships, and access patterns that would support efficient querying and real-time updates. Security policies were defined at this stage to ensure data isolation between users from the beginning.

The core web interface was developed next, implementing the visual components and interaction patterns for task, reminder, and note management. The interface was designed with a mobile-first approach, ensuring responsiveness across device sizes while maintaining full functionality. Authentication flows, dashboard layouts, and data visualization components were prioritized to establish the core user experience.

In parallel, the CLI tool development focused on creating an intuitive command structure that would provide equivalent functionality to the web interface while adhering to command-line interaction patterns. Special attention was paid to command naming, parameter handling, and output formatting to create a natural terminal experience that would feel familiar to CLI power users.

The real-time synchronization layer was implemented once both interfaces reached functional stability. This involved configuring Supabase's real-time channels, implementing event handlers in both interfaces, and ensuring consistent data transformation between backend and frontend representations. Extensive testing was conducted to verify synchronization behavior under various network conditions and concurrency scenarios.

Performance optimization and user feedback integration formed the final development phase. This included identifying and resolving performance

bottlenecks, implementing caching strategies, and refining the user experience based on feedback from beta testers across technical and non-technical backgrounds.

C. Database Schema Design

The database schema (Fig. 2) was designed to support comprehensive productivity management while enabling efficient queries and real-time updates.

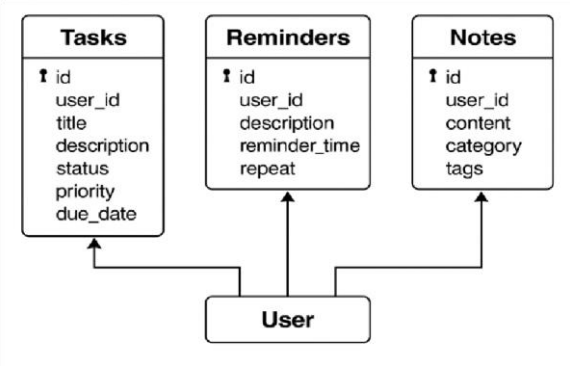

Fig. 2. Database schema diagram showing relationships between core entities and their attributes.

The schema includes four primary tables:

Users: Stores user authentication information and preferences

Tasks: Manages task data including title, status, priority, and due dates

Reminders: Stores time-based notifications with recurrence patterns

Notes: Contains textual information with categorization and tagging

Additional indices were created to optimize common query patterns, particularly for filtering and sorting operations. The database design incorporates soft deletion for all data types, maintaining record history while allowing users to recover accidentally deleted items.

D. Security Model and Data Isolation

Quickli implements a comprehensive authentication system using Supabase Auth, which provides secure user management capabilities. The system supports email/password authentication (traditional credentials-based login), third-party OAuth (support for Google, GitHub, and other OAuth providers), magic link authentication (passwordless email-based authentication), and multi-factor authentication (additional security layer for sensitive accounts).

The authentication flow is consistent across both interfaces: the user provides credentials via web form or CLI command; credentials are securely transmitted to Supabase Auth; upon successful validation, a JWT token is issued; the token is securely stored (browser storage for web, system keychain for CLI); and subsequent requests include the token for authorization

A critical feature of Quickli is its strong data isolation between users. This is achieved through PostgreSQL's Row-Level Security (RLS) policies configured in Supabase. The RLS policies enforce security constraints ensuring users can only read their own data, users can only write/modify their own data, system administrators have configurable access to audit logs, and deleted data is soft-deleted with retention policies.

This implementation ensures that even if API endpoints are compromised, the database layer provides strong isolation between user data. The policies are automatically enforced for all database operations, regardless of whether they originate from the web interface or the CLI.

The CLI tool implements secure token management to maintain authentication state between sessions. Authentication tokens are stored in the system keychain using the keytar library, which leverages platform-specific secure storage mechanisms: Windows Credential Manager (Windows), Keychain (macOS), and libsecret (Linux). The CLI automatically refreshes tokens when they expire and detects and handles revoked tokens appropriately when a token is revoked (e.g., after logout from web interface), when a password is changed, or when explicit logout is performed.

The system supports multiple authenticated devices per user with each device maintaining its own secure token storage, login state managed independently across devices, and session revocation targeted to specific devices when needed. This comprehensive token management ensures a seamless authentication experience for CLI users while maintaining strong security practices.

IV. IMPLEMENTATION

The implementation of Quickli focused on creating a seamless user experience across both web and CLI interfaces while maintaining consistency, security, and performance. This section details the specific implementation approaches for each interface and the synchronization mechanism that connects them.

A. Web Application Interface

The Quickli web application (Fig. 4) provides a comprehensive visual interface for productivity management. Developed using ReactJS with TailwindCSS for styling, the web interface follows a responsive design philosophy that adapts to different screen sizes and devices.

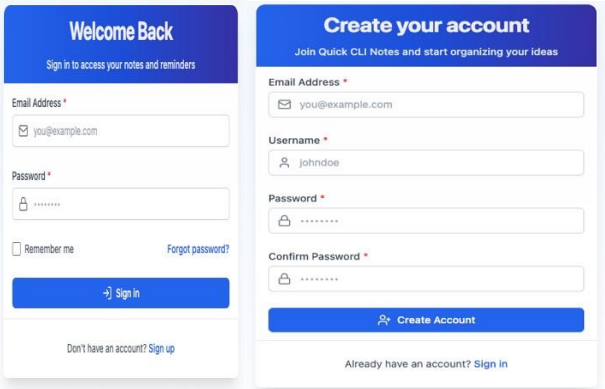

Fig. 4. Quickli web interface showing (a) login/signup screen, (b) dashboard with recent activities

1) Authentication Flow

The web application implements a secure user authentication flow:

Landing page with login/signup options

Email/password authentication

Social login options (Google, GitHub)

Password reset functionality

Session persistence using secure cookies

After successful authentication, users are redirected to the dashboard, which serves as the central hub for all productivity activities.

2) Dashboard Interface

The dashboard provides a comprehensive overview of the user's productivity data:

Recent tasks with status indicators

Upcoming reminders with timing information

Latest notes with category tags

Quick-action buttons for common operations

Activity summary with completion metrics

3) Task Management Interface

The task management section enables comprehensive CRUD operations:

Task creation with title, description, priority, and due date

List view with filtering and sorting options

Kanban board view for visual task organization

Batch operations for efficient task management

Detailed task view with progress tracking

4) Reminder System

The reminder interface provides time-based notification management:

Reminder creation with flexible recurrence patterns

Calendar view for temporal organization

Notification preferences and delivery methods

Integration with external calendar systems

Time zone awareness for global users

5) Notes Interface

The notes section offers robust information management:

Rich text editor with formatting options

Category and tag management

Search functionality with relevance sorting

Export options in various formats

Attachment support for comprehensive documentation

6) Import/Export Functionality

The web interface includes comprehensive data portability features:

Export of all data types (tasks, reminders, notes)

Multiple format options (JSON, CSV, PDF)

Import functionality with validation and conflict resolution

Backup scheduling for data protection

B. CLI Tool Implementaion

The Command Line Interface component of Quickli provides efficient terminal-based productivity management following UNIX design principles. Implemented as a global NPM package using Node.js and the Commander.js library, the CLI offers equivalent functionality to the web interface through a text-based interaction model optimized for keyboard efficiency.

The CLI structure follows a command-subcommand pattern with consistent naming conventions:

quickli tasks - Task management commands

quickli reminders - Reminder management commands

quickli notes - Note management commands

quickli auth - Authentication commands

The implementation uses structured command parsing with type validation and intelligent defaults to minimize required typing while maintaining precision. Date parsing uses the Chrono.js library to support natural language date inputs like "tomorrow," "next Monday 2pm," or "in 3 days," which are converted to precise timestamps for storage.

Output formatting receives special attention with table-based displays for list operations, color coding for status and priority indicators, and configurable

verbosity levels. The implementation uses the Chalk library for ANSI color support and the CLI-Table3 library for structured data presentation, with graceful fallbacks for terminals without color capabilities.

Authentication in the CLI follows the same security model as the web interface but adapted for terminal interaction. The login command prompts for credentials if not provided as arguments, with password input masked for security. Authentication tokens are securely stored in the system keychain using platform-specific mechanisms, enabling persistent authentication across terminal sessions without compromising security.

The CLI implementation includes interactive modes for complex operations, using the Inquirer.js library to provide form-like interfaces when appropriate. For example, the quickli tasks add --interactive command presents a series of prompts for task details, while maintaining the option for single-command operation when desired.

Performance optimization in the CLI focuses on startup time and command responsiveness. The implementation uses lazy loading for dependencies and connection pooling for database operations to minimize latency. Command execution follows an asynchronous pattern with appropriate error handling and helpful error messages when operations fail.

C. Real-Time Synronization

A key innovation in Quickli is the seamless real-time synchronization between the CLI and web interface (Fig. 6). This was implemented using Supabase's real-time capabilities based on PostgreSQL's [15] logical replication feature.

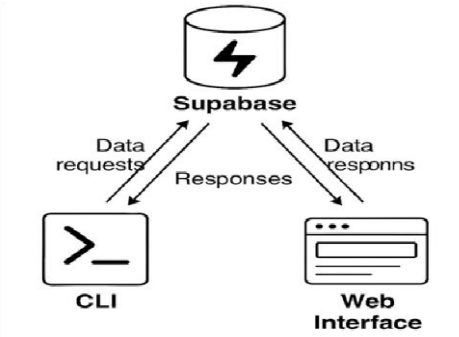

Fig. 6. Sequence diagram showing real-time data synchronization between CLI, web interface, and Supabase backend.

The synchronization process follows these steps:

User performs an action in either interface (CLI or web)

The respective interface sends the change to Supabase via API

Supabase processes the change and applies it to the PostgreSQL database

Logical replication triggers a real-time event

Both interfaces receive the event and update their local state

The user sees the change reflected across all active interfaces

This bidirectional synchronization ensures that users can seamlessly switch between interfaces based on their current context and workflow needs. For example, a user might quickly add tasks via the CLI while coding, then organize them into projects using the web interface's drag-and-drop capabilities later in the day.

VI. RESULT & DISCUSSION

Extensive testing and evaluation were conducted to assess Quickli's performance, scalability, and user experience. This section presents the results of these evaluations and their implications for real-world usage scenarios.

A. Performance Metrics

Performance testing was conducted across various operation types and network conditions to ensure responsive behavior in both interfaces. Each operation was tested 100 times under controlled network conditions (5Mbps bandwidth, 50ms latency) to obtain reliable averages.

TABLE I: Performance Metrics for Key Operations

Operation Type	*Web Interface (ms)*	*CLI (ms)*	*Sync Delay (ms)*
Task Creation	*187*	*213*	*247*
Task Update	*156*	*203*	*233*
Task Listing	*142*	*119*	*N/A*
Reminder Creation	*201*	*238*	*251*
Note Creation	*233*	*265*	*278*
Authentication	*378*	*402*	*N/A*

The results demonstrate that both interfaces provide responsive performance, with the CLI showing slight advantages in listing operations due to its simpler rendering requirements. Web interface operations typically completed within 150-250ms, while CLI operations ranged from 120-270ms depending on the complexity. Synchronization delays remained consistently under 300ms, which is imperceptible to users in typical productivity workflows.

B. Scalability Testing

To evaluate system scalability, we conducted load testing with simulated user accounts and data volumes (Fig. 8). The testing used a progressive approach, starting with 10 concurrent users and scaling up to 2,000 users, with each user performing a standardized sequence of operations.

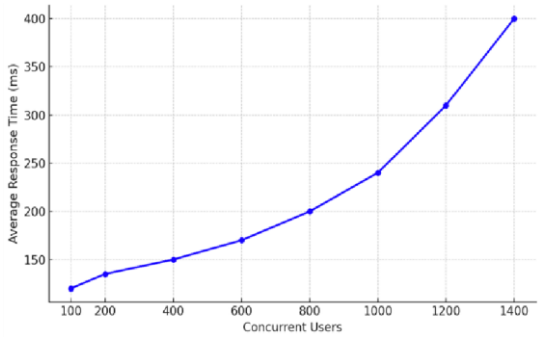

Fig. 8. System performance under increasing load showing response time versus concurrent users.

The system maintained linear scaling up to 1,000 concurrent users with proper database indexing and connection pooling. Response times increased by approximately 15% when scaling from 100 to 1,000 users, which remains within acceptable performance parameters. Beyond 1,000 users, response time degradation became more pronounced, indicating the need for horizontal scaling or additional optimization for very large deployments.

Database query performance remained consistent throughout scaling tests, with properly indexed queries showing minimal degradation even at higher user counts. The implementation of row-level security policies added approximately 5-8% overhead to query times, which was deemed an acceptable tradeoff for the security benefits provided.

These results indicate that Quickli's architecture can support small to medium-sized organizations without significant performance concerns, while larger deployments would benefit from horizontal scaling strategies.

C. User Feedback and Usability

We conducted a beta testing program with 15 participants of varying technical backgrounds. Participants used Quickli for a four-week period and provided feedback through structured surveys and interviews. Key findings include:

Overall satisfaction rating: 4.5/5

Web interface usability rating: 4.7/5

CLI usability rating: 4.2/5 (4.8/5 among technical users)

93% of participants reported that the dual-interface approach improved their productivity

87% indicated they would continue using the system after the study

VII. CONCLUSION

This paper presented Quickli, a novel cross-platform productivity system that bridges the gap between commandline and web interfaces. By providing seamless real-time synchronization, comprehensive security, and intuitive interfaces across both paradigms, Quickli demonstrates the viability of dual-interface productivity tools for modern workflows.

The performance evaluation and user feedback confirm that the system achieves its design goals of providing efficient, secure, and flexible productivity management across interface preferences. The open-source implementation provides a foundation for future research and development in cross-platform productivity systems.

The integration of Supabase as a backend-as-a-service platform with comprehensive security features demonstrates how modern serverless architectures can support sophisticated applications with complex synchronization requirements. The row-level security implementation provides strong data isolation without compromising performance, addressing a critical concern in multi-user productivity systems. This security model, combined with comprehensive token management across platforms, establishes a foundation for trust that is essential in productivity tools handling potentially sensitive information

As work environments continue to evolve toward hybrid models, tools that accommodate diverse user preferences and workflows will become increasingly important. Quickli represents a step toward more inclusive productivity systems that respect user autonomy while maintaining consistency and security.

VIII. FUTURE SCOPE

While Quickli provides significant advancements in cross-platform productivity management, there remain several promising areas for future development. One key direction is the implementation of an offline-first architecture using Conflict-Free Replicated Data Types (CRDTs) to ensure seamless data synchronization even without network connectivity. This would enable users to maintain productivity in disconnected environments, with changes synchronizing automatically when connectivity is restored.

Additionally, the development of native mobile applications for Android and iOS would enhance accessibility and user engagement, allowing users to

interact with the system on the go. Mobile interfaces could leverage platform-specific capabilities while maintaining data consistency with existing web and CLI interfaces.

The integration of machine learning models could further elevate productivity by enabling intelligent task prioritization and automated scheduling based on user behavior. By analyzing patterns in task completion, deadline adherence, and work habits, the system could offer personalized recommendations to optimize workflow efficiency.

Quickli could also evolve into a collaborative platform by introducing team workspaces, with support for role-based access and granular permission controls. This would extend the productivity benefits to team environments while maintaining the flexibility of interface choice for individual team members.

Lastly, building a modular extensibility framework would allow third-party developers to contribute plugins and integrations, significantly expanding the system's capabilities and adaptability to diverse workflows. Through an open API and extension marketplace, Quickli could grow into a comprehensive productivity ecosystem that adapts to specialized needs across various professional domains.

ACKNOWLEDGMENT

The authors would like to thank the beta testing participants for their valuable feedback and the open-source community for their contributions to the technologies that made this work possible.

REFERENCE

1. Atlassian, "Trello," [Online]. Available: https://trello.com, Accessed: 2024.
2. Asana, "Asana: Work management platform for teams," [Online]. Available: https://asana.com, Accessed: 2024.
3. Doist, "Todoist: The to-do list to organize work & life," [Online]. Available: https://todoist.com, Accessed: 2024.
4. TaskWarrior, "Taskwarrior: Free and open source software that manages your TODO list," [Online]. Available: https://taskwarrior.org, Accessed: 2024.
5. G. Baddeley, "Todo.txt: Future-proof task tracking in a file you control," [Online]. Available: http://todotxt.org, Accessed: 2024.
6. S. Yang, "Notion-CLI: Command line interface for Notion," GitHub repository, 2022.
7. GitLab, "GitLab: DevOps platform," [Online]. Available: https://gitlab.com, Accessed: 2024.
8. GitHub, "GitHub: Where the world builds software," [Online]. Available: https://github.com, Accessed: 2024.
9. Microsoft, "PowerToys: Windows system utilities," [Online]. Available: https://github.com/microsoft/PowerToys, Accessed: 2024.
10. Supabase, "Supabase Documentation," [Online]. Available:
11. https://supabase.com/docs, Accessed: 2024.
12. ReactJS, "React - A JavaScript library for building user interfaces," [Online]. Available: https://reactjs.org, Accessed: 2024.
13. Node.js Foundation, "Node.js," [Online]. Available: https://nodejs.org, Accessed: 2024.
14. J. Nielsen, "Usability Engineering," Morgan Kaufmann Publishers Inc., San Francisco, CA, 1993.

A. Cockburn, "Using Both an Integrated Development Environment and Command Line Interface for Development," International Conference on Software Engineering, pp. 52-61, 2020.

15. PostgreSQL Global Development Group, "PostgreSQL: The world's most advanced open source database," [Online]. Available: https://postgresql.org, Accessed: 2024.