International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 10 | Oct – 2025 DOI: 10.55041/ISJEM05096
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

Onehanded Coder:An Accessible Codind Environment for Single Handed Users

1st Ms.M.Jelcy M.Sc.,
Department of Computer Science
Sri Krishna Arts and Science College
Coimbatore,India
jelcym@skasc.ac.in

2nd V.Srimuki
Department of Computer
Sri Krishna Arts and Science College
Coimbatore,India
srimukivelmurugan0210@gmail.com

3rd K.Lathika Sarani
Department of Computer
Sri Krishna Arts and Science College
Coimbatore,India
lathikakumar667@gmail.com

ABSTRACT

The advancement of digital technologies and the growing dependence on coding platforms have made programming skills a necessity for students and professionals. The majority of available programming environments are, however, aimed at users with complete twohanded capabilities, and they pose severe restrictions for people with motor impairments or one-handed users. This paper presents Onehanded Coder, a web-based coding system of minimal weight specifically designed to increase software development accessibility and inclusivity. The system offers crucial features like syntax highlighting, autocompletion, inline error checking, shortcutbased workflows, and on-the-fly program execution while maintaining low mouse navigation dependency. Built on a current technology stack of HTML, CSS, JavaScript, React.js, Node.js, and MongoDB, the system balances functionality and usability. Systemic testing has validated its efficiency in safe authentication, file handling, program compiling, and running. The findings reveal Onehanded Coder minimizes physical exertion, eases coding activities, and provides a seamless user experience on devices. By merging accessibility with effectiveness, the system supports equal opportunities for students and developers and lays the groundwork for more inclusive technology tools in the future.

Keywords-*Accessibility, Code Editor, Inclusivity, Web Application, Onehanded Coder.*

INTRODUCTION

Programming has emerged as one of the most critical skills in the current digital economy, with thousands of tools to assist developers in writing, testing, and deploying code. Despite this plethora of platforms, most environments for coding assume that the user is completely in control of input devices using both hands. Integrated Development Environments (IDEs) and text editors tend to involve frequent interhands switching, complex multi-key shortcuts, and extensive coordination between the two hands. Although these design assumptions may not be problematic for the typical user, they pose obstacles for differently-abled learners, users with motor impairments, and the one-handed workflow only users.

This exclusion not only restricts the possibility of such users fully engaging in programming but also enhances educational and work-based inequalities in the domains of technology. The Onehanded Coder project overcomes these constraints by designing an accessible, webbased code editor

tailored for single-hand use. The system minimizes the unnecessary dependency on two-handed usage and minimizes fatigue by providing shortcut-based operations, autocompletion, inline error indication, and a consistent, lightweight interface. By emphasizing accessibility, personalization, and inclusivity, this project hopes to play a part in closing the gap between differently-abled people and general software development practices.

PROBLEM STATEMENT

Most current IDEs are not single-hand optimized and hence demand complicated two-handed tasks. It makes coding a laborious activity for motor-impaired users. Challenges are the reliance on constant hand switching between the mouse and the keyboard, difficulties with multikey shortcuts, and the absence of personalization. While there are accessibility tools like on-screen keyboards and voice-based environments, they only offer partial support and do not ease the entire workflow. Hence, a coding system that is specially intended to provide single-handed interaction has to be developed.

OBJECTIVES

In typical coding settings, developers are supposed to be navigating with both hands simultaneously, frequently executing pairs of keyboard shortcuts while at the same time using the mouse for navigation. For people who may not comfortably use both hands, this is a considerable impediment. The frequent handswitching doubles the amount of physical strain, makes typing and debugging more timeconsuming, and decreases productivity. Accessibility features like on-screen input and voice-input systems have been added as workarounds; however, these are merely surface fixes and do not address optimization of coding workflows. For instance, inputting lines of code with a voice command can be slow and errorprone, and on-screen keyboards don't have the speed and comfort necessary for extended programming sessions.

The lack of specialized tools developed for onehanded programming causes coding to become unnecessarily hard for differently-abled students. This restriction not only causes the rate of errors to go up but also makes learners less interested in embracing programming activities in a full-fledged way. Hence, there exists an urgent need for an inclusive approach with minimal effort and all the key features of a contemporary programming environment.

SYSTEM ANALYSIS

A. EXISTING SYSTEM

The programming environment today is dependent on full-fledged IDEs and editors like Eclipse, Visual Studio, or Sublime Text. These are powerful and full-featured but are assumed to be used two-handed. Users must remember hardto-remember multi-key shortcuts and constantly use the mouse. The process not only raises the cognitive burden but also results in tiredness and diminished performance when performed by single-handed users. Accessibility features like speech-to-text or virtual keyboards try to make up for it, but they do not organize workflows in such a manner as to

facilitate continuous coding. It leads to reduced productivity, frustration, and overall poor learning experience due to this nonoptimization for differently-abled learners.

B. PROPOSED SYSTEM

Onehanded Coder system has a novel solution by adopting single-handed interaction as its exclusive focus. It supports keyboard-controlled shortcuts eliminating the hassles of multiple mouse clicks, inline error messages to cut down on debugging time, and auto-completion to reduce typing effort. The system offers a uniform and customized environment by saving user choices like themes and frequently used shortcuts. Unlike current solutions to accessibility, this system is tailored for coding, which makes workflows easier and lessens the physical burden of programming activities. Its lightweight nature allows it to provide performance on devices without necessarily demanding high-end specs, thus finding applicability in learning for a variety of learners.

SYSTEM DESIGN

A. MODULES

Authentication & User Management Module:

It manages secure login and registration of users. It checks credentials on login and maintains that only authenticated users can use the platform. It also takes care of password encryption, fetching, and user accounts to ensure a safe and trustworthy coding environment.

Code Editor Module:

The code editor is the main workspace where users write and edit their programs. It is enhanced with features such as syntax highlighting, auto-completion, template snippets, and inline error prompts. The editor is optimized for one-hand usage, allowing shortcut-driven commands that reduce dependence on mouse operations.

Compiler & Execution Module:

This module takes up the task of compiling and running the code entered by the user. It executes the program, detects errors, and shows the output in real-time. Errors are marked with straightforward messages so that users have no problem finding and correcting them easily without changing to external compilers.

File Management Module:

The file management system enables users to save, read, and delete coding files. This provides session continuity, making it possible for users to pick up their saved work at any point. Streamlined file storage avoids data loss and allows for flexibility in dealing with multiple projects.

User Preferences & Settings Module:

This module individualizes the coding space for every user. It stores preferences like editor schemes, font types, and personalized hotkeys. By storing these preferences between logins, the system provides a unified and familiar experience for the user.

Administration Module

The admin module is meant for system administrators to track user accounts, logs, and the overall system stability. Administrators can authorize modifications, resolve problems, and make sure that the coding platform is secure as well as scalable.

Help & Support Module:

This module helps users understand and operate the system. It gives users access to FAQs, usage instructions, and troubleshooting assistance. Through provision of instant assistance, it minimizes confusion, particularly for novice users.

B. INPUT DESIGN

The input design is effort-minimal while being accurate and secure. All inputs from the user, including login information, registration information, and program code, are collected via easy-to-use forms and a responsive editor interface. Validation tests are applied to stop improper or incomplete submissions, like crosschecking for correct email formats and checking for password strength. In the code editor, syntax checking and auto-indentation make errors during program entry less likely. Through keyboard-first models and shortcutbased commands, the system minimizes mouse interactions, thereby making the system accessible and efficient for one-handed use.

C. OUTPUT DESIGN

Output design guarantees that results are meaningfully and clearly presented to users. The system's primary output is the result of program execution, which is displayed directly in the editor interface. Error outputs, including syntax or runtime errors, are clearly indicated with highlight and line numbers, which lead users to correct errors easily. Successful outputs are presented in a structured display area to differentiate them from error messages. Also, non-interactive operations like registration, login, and saving files give feedback outputs to ensure users that their operations have been successfully executed. The outputs are presented using readable fonts, proper spacing, and accessible color coding, which facilitates singlehanded interaction.

C. DATABASE DESIGN

Database structure is crucial in storing user data, coding files, and system logs in an efficient manner. Onehanded Coder relies on MongoDB as its native database, organized into collections that map to various entities in the system. The Users collection has one-off user credentials, hashed passwords, and profile settings. The Files collection holds saved code files by users, with timestamps for versioning. The Tickets and Messages collections hold user support requests and respective communications threads. The Audit Logs collection tracks administrative activity and system-level events to maintain accountability. Through a normalized and secure design, the database removes redundancy, provides data integrity, and facilitates quick information retrieval for an uneventful user experience.

SYSTEM IMPLEMENTATION

The system is developed with a modern technology stack. The front-end is created with HTML, CSS, and JavaScript, and React.js handles dynamic user interfaces using its component-based design. The backend is created with Node.js and Express.js, which handles routing and API handling for functionalities like login, file saving, and code execution. MongoDB is the main database, holding user credentials, code files, preferences, and execution history.

During deployment, every module was initially unit-tested for functionality. For instance, the login module was tested for secure authentication and input validation. Integration testing was done next, checking communication between modules like the compiler and editor. Validation testing was done to ensure that there were error messages for incorrect input. Last but not least, acceptance testing was carried out with emulation of real-world environments, ensuring that the platform behaved correctly under actual use. To aid new users, training was offered through guides, help screens, and live demonstrations that acquainted them with major features such as coding, saving, and running.

PHASES OF IMPLEMENTATION

The deployment of Onehanded Coder was done in a series of properly defined phases so that every part was written, tested, and installed in a verifiable and controlled manner. The first phase, Requirement Analysis & Planning, consisted of completing functional and nonfunctional requirements according to user demands for single-handed use. At this phase, indepth specifications were created for the editor features (auto-completion, inline error detection, keyboard-first shortcuts), backend services (execution API, authentication), and database schema (users, files, tickets, logs). Project scope, module responsibilities, and success criteria were recorded so that ongoing development stayed true to accessibility and performance objectives.

The Design Phase built requirements into tangible system blueprints. UI/UX wireframes and the editor interaction model were built with keyboard-first flows and reduced mouse dependency. Architectural diagrams established the way the React frontend, Express/Node.js API, execution/compile service, and MongoDB collections would communicate. Data models, API contracts, and security procedures (password hashing, role-based access) were established. This phase also generated the input/output formats, DFD and ER diagrams that would inform implementation and testing.

In the Development Phase, the modules were developed iteratively following a componentbased development. The frontend team developed reusable React components for the editor, preferences, file manager, and support pages with accessibility features (tab order, ARIA attributes, large target areas for shortcut mappings). The backend team implemented RESTful endpoints for authentication, file operations, execution requests, and admin functions. The execution module was developed as a sandboxed service (or an API mocked up for initial development) to compile/interpret provided code and send back

results. Database collections and indexes in MongoDB were established to keep user profiles, code files, tickets, and audit logs. Each of the features was committed piecemeal to support constant review and integration.

Unit Testing was done in parallel with development. Individual elements—like the login API, editor auto-completion engine, file save/retrieve handlers, and execution endpoint—were tested with positive and negative examples to check for correctness and input validation. Automated tests were written by developers for key logic (e.g., password hashing, token validation, save/delete file flows) and manual test scripts for UI behavior individual to onehand usability (shortcut activation, focus management, and keyboard navigation).

After unit tests, Integration Testing ensured interactions between modules. Integration tests ensured that code typed in the editor could be sent to the execution service and outputs reflected back in the UI. Integration also ensured end-to-end flows: registration → login → create file → run code → save execution history. API responses were verified for proper status codes, right error messages, and security (JWT token management, secure admin routes).

Performance tests were involved to identify delays in the execution pipeline and database queries.

System Testing & Validation tested the entire application in the target environment and on supported browsers. Tests comprised functional testing (all user cases function as expected), compatibility tests (Chrome, Firefox, Edge), and accessibility tests (keyboard navigation, screenreader naming). Usability tests emphasized single-hand usage: quantifying keystrokes to perform typical tasks, ensuring shortcuts were within one-hand reach, and verifying minimal mouse usage. Security validation comprised ensuring encrypted password storage, role-based access controls, and defense against typical web threats (e.g., injection, XSS).

Acceptance Testing was done with key end users, such as those who depended on singlehand use, to ensure that the system met actual user requirements. The feedback gathered during acceptance testing informed final tweaks: tuning default shortcuts, enhancing inline error messages, and adjusting UI contrast for readability. All defects or usability problems found in this phase were fixed prior to deployment.

The Deployment Phase staged the application for production. Server configuration, environment variables, database provisioning, and secure access policies were set. Continuous integration/continuous deployment (CI/CD) pipelines were set up to build and deploy automatically. The deployment involved configuring logging and monitoring to monitor application health and usage metrics, and setting up backup routines for the MongoDB instance to protect user data.

User Documentation and Training were done to facilitate easy adoption. This involved a user guide explaining how to register, alter preferences, utilize keyboard shortcuts, save files, and execute code.

In-app help screens and brief tutorial walkthroughs were implemented to quickly onboard new users. Administrators had unique documentation detailing user management, logs monitoring, and maintenance processes.

Lastly, Maintenance & Iterative Improvement started post-deployment. This process encompasses regular preventive maintenance (security patches, performance tuning, database backup) as well as perfective maintenance (adding features like multi-language support, collaborative editing, or AI-enhanced suggestions). A feedback loop was created to gather user reports and analytics, rank enhancements, and plan periodic updates, keeping the platform stable, secure, and increasingly more usable for one-handed developers.

MAINTENANCE OF THE SYSTEM

System maintenance is perhaps the most important phase of the software life cycle because it guarantees that the application remains effective after it is deployed. For Onehanded Coder, maintenance is essential to deal with user comments, correct defects, optimize performance, and change the system according to new needs. As actual use tends to reveal situations not completely envisioned during development, the developers and administrators need to closely monitor and improve the system constantly.

Maintenance operations in Onehanded Coder can be divided into two categories: Perfective Maintenance and Preventive Maintenance.

Perfective Maintenance entails developing and enhancing the system by adding new functionality or enhancing current functions. For instance, in subsequent releases, the platform could accommodate more programming languages, have collaborative coding functionalities, or provide cloud storage for files.

User experience improvements, including additional themes, shortcut mapping customizations, and better debugging recommendations, fall under perfective maintenance. These enhancements make the system improve in step with user demands and advancements in technology.

Preventive Maintenance is concerned with reducing potential future issues and ensuring long-term stability. This involves applying routine security patches, ensuring logs for abnormal activity, ensuring database queries are optimized, and ensuring server environments are upgraded. Preventative action also ensures regular data backups are scheduled to safeguard user code and settings from unintentional loss. Furthermore, load testing and system monitoring are performed to ensure the application will still function efficiently even when users increase.

In combination, perfective and preventive maintenance ensure that Onehanded Coder is stable, secure, and scalable. Through proactive maintenance, the system will remain an inclusive and efficient coding space, able to evolve according to both existing and new requirements.

RESULTS AND DISCUSSION

Testing and analysis of the Onehanded Coder system indicated that it achieves its goals of accessibility, usability, and performance. It was tested and found to be secure for authentication. The code editor was smooth and responsive. Auto-completion and error messages saved considerable typing effort and debugging time. Saving and loading files were tested successfully, maintaining continuity from session to session. The system was also stable under the use of multiple users at the same time, responding well to load. In general, the findings indicate that Onehanded Coder reduces the difficulties of single-handed users effectively and provides a sustainable and accessible solution for coding exercises.

CONCLUSION

The Onehanded Coder project proves that technology can be used effectively in overcoming accessibility issues when it comes to programming. Through the delivery of a light, accessible, and extensible environment, the platform minimizes physical work and allows differently-abled programmers to code more effectively. The system achieves its goals of usability and inclusivity while preserving critical aspects of contemporary IDEs. Through constant updates and upgrades, Onehanded Coder can grow to become a globally used platform that facilitates equal opportunities in software development.

FUTURE ENHANCEMENT

While the system currently fulfills its purpose, there is still room for expansion. Subsequent releases can add more programming languages to offer users more choice. Features for collaborative coding can enable multiple users to code in unison in real time. Integration of cloudbased storage can enhance access by allowing users to access their projects on any device. The integration of artificial intelligence can offer adaptive suggestion of code and individualized learning pathways. Lastly, creating mobile apps for Android and iOS would make the platform's use available everywhere and allow coding anywhere, anytime.

REFERENCES

1. Pressman, R. S. (2014). Software Engineering: A Practitioner’s Approach. McGraw-Hill Education.
2. Sommerville,I.(2015).Software Engineering(10th ed.).Pearson Education.
2. Balagurusamy, E. (2019). Programming in ANSI C. McGraw-Hill Education.
3. Fowler, M. (2004). UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison Wesley.