IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

DIGITAL CALCULATOR WEB APPLICATION

Amruta Dhanaji Yadav,Srushti Gangaram Kale, Akshata Surykant Khamkar, Priti Asaram Gore]

University/College : [Pharate Patil Management Institutes Mandavgan Pharata, Tal-Shirur, Dist-Pune]

Guide/Supervisor : [Bhise sir and Akshay Jagtap]

ABSTRACT

This project presents a web-based digital calculator developed using the Flask framework, incorporating user authentication, calculation history tracking, and a modern user interface. The application allows users to perform basic arithmetic operations securely, with their calculations stored in a database for later review. It demonstrates key concepts in web development, database management, security, and client-server architecture. The system uses SQLite for data persistence, Flask-Login for session management, and AJAX for seamless interactions.

INTRODUCTION

1.1 Background

Calculators have evolved from mechanical devices to digital applications. With the rise of web technologies, creating interactive tools like calculators as web apps provides accessibility across devices. This project builds a calculator that not only computes expressions but also ensures user privacy through authentication and maintains a personal history log.

1.2 Problem Statement

Traditional calculators lack user-specific features like history tracking and security. This app addresses that by requiring login, storing calculations per user, and preventing unauthorized access.

1.3 Objectives

- Develop a responsive web calculator for basic operations.
- Implement user registration, login, and session management.
- Store and retrieve calculation history using a database.
- Ensure secure evaluation of expressions and data handling.

- Design an intuitive UI with modern styling.

1.4 Scope

The app supports basic arithmetic (+, -, *, /) with parentheses. Future expansions could include scientific functions. It is designed for educational purposes, demonstrating Flask's capabilities.

LITERATURE REVIEW

2.1 Web Frameworks

Flask is a lightweight Python micro-framework ideal for small to medium projects. Compared to Django (full-stack), Flask offers flexibility without overhead. Studies (e.g., from Stack Overflow surveys 2025) show Flask's popularity for APIs and simple apps due to its minimalism.

2.2 Authentication in Web Apps

User authentication prevents unauthorized access. Flask-Login handles sessions securely, using cookies and CSRF protection. Literature on web security (OWASP Top 10, 2025) emphasizes hashing passwords (using Werkzeug) and protecting against injection attacks.

2.3 Database Integration

SQLAlchemy ORM abstracts database operations, supporting SQLite for development. Research on NoSQL vs. SQL (e.g., ACM papers) highlights SQL's suitability for relational data like user-calculations.

2.4 Expression Evaluation

Python's eval() is used with restrictions to avoid code injection. Alternatives like sympy for symbolic math were considered but kept simple for this scope.

2.5 UI/UX Trends

Modern web apps use CSS gradients, shadows, and fonts like Orbitron for a futuristic look. AJAX/Fetch enables real-time updates without reloads, improving user experience (as per Nielsen Norman Group UX studies).

METHODOLOGY

3.1 System Architecture

- Client-side: HTML, CSS, JS for UI and interactions.
- Server-side: Flask handles routes, auth, and calculations.
- Database: SQLite stores users and history.
- Communication: JSON via AJAX for calculations.

3.2 Development Process

Agile methodology: Iterative development with sprints for auth, calculator, history, and testing.

3.3 Tools and Technologies

- Backend: Python 3.12, Flask, Flask-SQLAlchemy, Flask-Login, Werkzeug.
- Frontend: HTML5, CSS3, Vanilla JS.
- Database: SQLite.
- IDE: VS Code.
- Version Control: Git.

IMPLEMENTATION DETAILS

4.1 Database Schema

- User: id, username, password (hashed).
- Calculation: id, expression, result, timestamp, user_id (foreign key).

4.2 Authentication Flow

- Register: Hash password, store user.
- Login: Verify credentials, start session.
- Protected Routes: Use @login_required decorator.

4.3 Calculator Logic

- Client: Build expression, send via Fetch.
- Server: Sanitize input, eval(), store in DB if success.
- History: Query DB, display in reverse chronological order.

4.4 Security Measures

- Input sanitization to allowed chars.
- Password hashing.
- Session protection.
- Error handling for invalid expressions.

4.5 UI Implementation

- Neon-themed CSS with grid layout for buttons.
- Keyboard support for accessibility.
- Flash messages for feedback.

TESTING AND RESULTS

5.1 Unit Testing

- Tested auth: Register duplicate user (fails), login invalid (fails).
- Calculator: 2+2=4 (success), /0 (error).
- History: Added entries, verified display.

5.2 Integration Testing

- End-to-end: Register → Login → Calculate → View History → Logout.

5.3 Performance

- Response time <100ms for calculations.
- Scalable for small user base (SQLite limit).

5.4 Results

The app functions as intended, with secure auth and persistent history.

FEATURES

- User registration and login with password hashing.
- Secure session management and logout.
- Basic arithmetic operations (+, -, ×, ÷) with parentheses.
- Real-time calculation via AJAX (no page reload).
- Personal calculation history with timestamps.
- Clear (C) and Delete last character (DEL).
- Keyboard support.
- Error handling (e.g., division by zero).
- Responsive & modern neon-style UI.

FUTURE SCOPE / POSSIBLE ENHANCEMENTS

- Advanced auth: OAuth (Google login), 2FA.
- Scientific calculator: Add sin, cos, log, etc., using sympy.
- History features: Search, delete entries, export to CSV.
- Multi-user sharing: Share calculations.
- Deployment: Host on Heroku/AWS with PostgreSQL.
- Mobile app integration via Progressive Web App (PWA).
- Analytics: Track usage patterns.
- Accessibility: ARIA labels, voice commands.

CONCLUSION

This project successfully implements a secure, user-centric digital calculator using Flask, demonstrating proficiency in web development, database integration, and security practices. It provides a foundation for more complex applications and highlights the importance of user data protection in modern software.

Key Learnings:

- Flask's routing and extension ecosystem.
- ORM for efficient DB interactions.
- Client-side scripting for dynamic UIs.
- Security best practices (hashing, sanitization).
- Project management from ideation to testing.

References:

- Flask Documentation (flask.palletsprojects.com)
- OWASP Security Guidelines (owasp.org)
- Python eval() Safety (docs.python.org)
- UX Design Principles (Nielsen Norman Group) Thank you for your

valuable guidance.

Date: January 2026
Place: [Your City]