ANUDIP FOUNDATION

<u>Project Title</u>

" Speed Typing Tester"

By

Name	**Enrollment No**
Shubham Vitthal Pawar	**AF0481807**

Under Guidance
of

Rajshri Thete

ABSTRACT

The **Speed Typing Tester** is a web-based application designed to measure a user's typing speed and accuracy. It provides an interactive platform where users can practice typing, track their performance, and store their results for future reference. The system calculates metrics such as words per minute (WPM), accuracy percentage, and errors made during a typing test. Built using HTML, CSS, JavaScript, Python, and MySQL (via XAMPP), the project ensures a userfriendly interface, secure data storage, and reliable performance. This documentation outlines the project's objectives, system analysis, design, implementation, and testing phases.

The application lets users track their progress over time by saving their results. This helps users see where they need to improve and make their practice more focused. It also offers different difficulty levels and time options, so users can choose tests that match their skill level.

The Speed Typing Tester includes features like real-time error detection, speed tracking, and a leaderboard to motivate users to improve.

ACKNOWLEDGEMENT

The project **"Speed Typing Tester"** is the Project work carried out by

Name	**Enrollment No**
Shubham Vitthal Pawar	**AF0481807**

Under the Guidance.

We are thankful to my project guide for guiding me to complete the Project.

His suggestions and valuable information regarding the formation of the Project Report have provided me a lot of help in completing the Project and its related topics.

We are also thankful to my family member and friends who were always there to provide support and moral boost up.

CONTENT OF THE PROJECT

TOPIC OF THE PROJECT	*Page No*

TITLE OF THE PROJECT

SPEED TYPING TESTER

INTRODUCTION AND OBJECTIVES OF PROJECT

INTRODUCTION

A typing speed test measures how quickly and accurately you can type on a keyboard. It's a quick way to check your typing skills and see how many words you can type correctly per minute, also known as WPM.

Typing speed is an essential skill in the digital age. The Speed Typing Tester web application provides a fun and interactive way to assess and improve typing skills.

The **Speed Typing Tester** is a web application that allows users to test their typing skills by typing a given text within a set time. The system calculates the user's typing speed (in words per minute), accuracy, and errors, providing immediate feedback. It also stores user results in a MySQL database for tracking progress over time. The application is developed using HTML and CSS for the front-end, JavaScript for interactivity, Python for backend processing, and MySQL (via XAMPP) for data storage.

The project aims to provide a simple, engaging, and accessible tool for users to improve their typing skills, whether for professional development, academic purposes, or personal growth. It addresses the need for a reliable platform to practice typing in a digital environment.

OBJECTIVES OF PROJECT

The objective of a speed typing tester is to assess and improve a user's typing speed and accuracy. This can be achieved by providing a platform for users to practice typing, track their progress, and receive feedback on their performance. Additionally, the project aims to motivate users to improve their typing skills through gamification and progress tracking

The main objectives of the Speed Typing Tester project are:

- To create a user-friendly web application for testing typing speed and accuracy.
- To provide real-time feedback on typing performance, including WPM, accuracy, and errors.
- To store user results securely in a database for progress tracking.
- To ensure the system is reliable, easy to use, and scalable for future enhancements.
- To implement a responsive design that works on various devices (desktops, tablets, mobiles).
- To incorporate secure user registration and login features for personalized experiences.

SYSTEM ANALYSIS

✦ Problem Definition

Many individuals lack access to tools that help them practice and measure their typing skills effectively. Existing typing test applications may have limitations such as:

- ✓ Lack of user progress tracking.
- ✓ Inadequate security for user data.
- ✓ Complex interfaces that are not beginner-friendly.
- ✓ Limited feedback on typing performance.
- ✓ Inability to handle multiple users simultaneously.

The Speed Typing Tester addresses these issues by offering a simple interface, secure data storage, detailed performance metrics, and progress tracking.

✦ Preliminary Investigation

Purpose-

The project aims to provide an online platform for users to test and improve their typing skills anytime, anywhere. Users can take typing tests, view their results, and track their progress over time.

Benefits-

- Convenient access to typing tests via a web browser.
- Detailed performance metrics (WPM, accuracy, errors) to help users improve.
- Secure storage of user data and test results.
- User-friendly interface suitable for all skill levels.
- Progress tracking to motivate users to practice regularly.

Proposed System-

The proposed system is designed to be reliable, user-friendly, and efficient:

- Users can register and log in to access personalized features.
- The system provides random text for typing tests to ensure variety.
- Results are calculated instantly and stored in a MySQL database.
- Users receive a mobile-friendly interface for accessibility.
- The backend (Python) ensures secure data processing and storage.

Technical Specifications-

The application uses:

- **HTML/CSS**: For creating a responsive and visually appealing front-end.
- **JavaScript**: For real-time interactivity and result calculations.
- **Python**: For backend logic and database interactions.
- **MySQL (via XAMPP)**: For storing user data and test results.
- **XAMPP**: As the local server environment for development and testing.

REQUIREMENTS SPECIFICATIONS

The Speed Typing Tester must fulfill the following criteria to ensure a robust, user-friendly, and scalable application:

1. **Reliability-** The system must perform typing speed calculations consistently without crashes or errors during tests.
2. **Good Organization-** The system should serve its core purpose: measuring typing speed (WPM/CPM) and accuracy (%) in a structured manner.
3. **Correctness-** Results (WPM, accuracy, errors) must be calculated precisely based on user input.
4. **Usability-** The interface should be intuitive for users of all skill levels (e.g., clear instructions, minimal buttons).
5. **Maintainability-** Code and database should be modular for easy debugging and updates (e.g., separate functions for WPM calculation and error detection).
6. **Expandability-** The system should allow future additions (e.g., new test difficulty levels, multiplayer modes) without major redesign.
7. **Portability-** The web application should deploy seamlessly on different servers (Apache, Nginx) and devices (desktop, mobile).
8. **Accurateness-** Ensure real-time metrics (WPM, errors) reflect the user's actual typing performance.
9. **Error Prevention-** Detect and handle edge cases (e.g., accidental keystrokes, backspace misuse) without skewing results.
10. **Communication-** Optionally, integrate email notifications for test results or progress reports (future scope).

FEASIBILITY STUDY

A feasibility study evaluates the practicality of the proposed system. The Speed Typing Tester was analyzed across several feasibility types:

Technical Feasibility

- The project uses widely available technologies (HTML, CSS, JavaScript, Python, MySQL).
- XAMPP provides a stable server environment for development.
- The system is compatible with modern web browsers and operating systems (Windows, macOS, Linux).

Economic Feasibility

- The project requires minimal hardware (a standard PC with 128 MB RAM and 20 GB storage).
- Software tools (XAMPP, MySQL, Python) are open-source and free.
- Development costs are low, as the project is built by a single developer.

Schedule Feasibility

- The project was completed within three months, making it time-feasible.
- Each phase (analysis, design, coding, testing) was allocated sufficient time.

Operational Feasibility

- The interface is intuitive, requiring no prior technical knowledge.
- Users can easily navigate and perform typing tests.
- Features such as **typing tests** are simple and user-friendly, allowing users to navigate and utilize the system effectively without training or guidance. The focus is on accessibility and ease of use.

Social Feasibility

- The project promotes skill development and is socially acceptable.
- It does not involve any illegal or unethical operations.

Advantages of Feasibility Study

- Identifies risks and mitigation strategies.
- Ensures cost-effective development.
- Confirms the system meets user needs and technical requirements.

PROJECT PLANNING AND SCHEDULING :

PROJECT PLANNING

- **Project Planning**

The project is divided into the following phases:

1. Preliminary Investigation
2. System Analysis
3. System Design
4. Coding
5. Security Implementation
6. Testing
7. Implementation

Activity Diagram

Preliminary Investigation

- Identification of Need (Automated Typing Speed Assessment)
- Requirement Specifications (Functional/Non-functional)
- Feasibility Study (Technical, Economic, Operational)

System Analysis

- Project Planning & Scheduling
- SRS (Software Requirement Specification) Preparation
- Choice of Software Paradigm (Agile/Waterfall)

System Design

- **Module Description**:
    - User Authentication
    - Typing Test Engine (WPM/CPM Calculation)
    - Results Dashboard

- **Database Design**:
    - Users Table (UserID, Username, PasswordHash)
    - TestResults Table (TestID, UserID, WPM, Accuracy, Timestamp)
- **UI Design**:
    - Typing Interface (Timer, Input Field, Live Stats)
    - Results History Page

Coding

- Frontend (HTML/CSS/JavaScript for real-time typing test)
- Backend (Python + MySQL for data storage)
- Debugging and Optimization

Security

- User Authentication (Password Hashing)
- Database Security (SQL Injection Prevention)

Testing

- **Unit Testing**: WPM calculation, error detection
- **Integration Testing**: User login → Test → Results flow
- **System Testing**: End-to-end performance validation

Implementation

- Deployment on XAMPP/local server
- Documentation (User Guide, Admin Manual)

Milestone Achieved

Phase	**Milestone**
Preliminary Investigation	Identified need for a typing speed assessment tool
System Analysis	SRS finalized and approved
System Design	UI mockups and database schema completed
Coding	Fully functional prototype ready
Testing	All test cases passed (100% coverage)
Implementation	System deployed with user documentation

SOFTWARE REQUIREMENT SPECIFICATION (SRS)

1. Introduction

Requirements analysis is conducted to define the scope of the **Speed Typing Tester**, a web-based application that measures typing speed (WPM/CPM) and accuracy. This phase produces:

- **Data Requirements**: For database design (user profiles, test results).
- **Functional Requirements**: For application logic (real-time calculations, user flows).

2. Project Overview

Project: **Speed Typing Tester** This system allows users to:

- Take timed typing tests with random text passages.
- View real-time metrics (WPM, accuracy, errors).
- Save results to track progress over time.
- **Admin** can manage test passages and view aggregated user statistics.

3. Data Requirements User Module

User Login/Authentication

- Users log in with:
  - **Username**
  - **Password** (hashed in the database).
- **Optional**: Guest mode for quick tests without registration.

User Actions

- Start/pause/reset typing tests.
- View live WPM, CPM, and accuracy during tests.
- Access historical test results in a dashboard.
- Update profile (email, password).

Self-Registration

- Users can create accounts (unlike the voting system example).

Admin Module

Admin Login

- Admins log in with **username** and **password**.

Test Passage Management

- Add/edit/delete typing passages (easy/medium/hard difficulty).
- View system usage statistics (average WPM, user count).

User Management

- Delete inactive accounts.
- Reset passwords (if needed).

4. Functional Requirements

1. **Typing Test Execution**
   - Display random text passages for typing.
   - Timer starts automatically on first keystroke.
2. **Real-Time Metrics** – Calculate and display:
   - ✦ **WPM** (Words Per Minute).
   - ✦ **CPM** (Characters Per Minute).
   - ✦ **Accuracy** (%).
   - ✦ **Error count** (misspelled words).
3. **User Authentication**
   - Secure login with session management.
   - Password recovery via email (optional).
4. **Data Persistence**
   - Save test results to the database (MySQL via XAMPP).
5. **Input Validation**
   - Ignore non-alphanumeric keys during tests.

- Prevent backspace abuse from inflating accuracy.

6. **Admin Controls**

- CRUD operations for test passages.
- Export user statistics (CSV/PDF).

5. **Additional Features**

User Experience

- **Color-coded feedback**: Correct (green) vs. incorrect (red) keystrokes.
- **Test history graph**: Visualize WPM progress over time.

Security

- **Password hashing** (bcrypt/scrypt).
- **Session timeout**: 30 minutes of inactivity.

Performance

- Real-time calculations with **<1s latency**.
- Support **50+ concurrent users**.

Compatibility

- Responsive design (desktop, tablet, mobile).
- Cross-browser support (Chrome, Firefox, Edge).

6. Database Schema

Tables

1. `users` ○ `user_id` (PK), `username`, `email`, `password_hash`, `created_at`
2. `test_results` ○ `test_id` (PK), `user_id` (FK), `wpm`, `accuracy`, `errors`, `timestamp`

3. `passages` (Admin-managed) ○ `passage_id` (PK), `text`, `difficulty_level`, `added_by_admin`

SOFTWARE REQUIREMENTS :

For Development & Testing -

1. Operating System

- **Windows**: Windows 7, 8, 10, or 11 (64-bit recommended).
- **Linux**: Ubuntu 20.04+, Fedora, or similar distributions.
- **macOS**: macOS 10.15 (Catalina) or newer.

2. Web Server

- **Apache** or **Nginx** (for local development).
- **Python built-in server** (for testing Flask/Django backend, if used).

3. Database

- **XAMPP** (includes MySQL and phpMyAdmin for local database management).
- **Alternative**: Online MySQL/PostgreSQL (e.g., AWS RDS, PlanetScale).

4. Web Development Tools

- **Frontend**:
  - **VS Code** / **Sublime Text** (for HTML/CSS/JavaScript).
  - **Bootstrap** (optional for responsive design).
- **Backend** (if using Python): ○ **Python 3.8+** with libraries: Flask/Django, SQLAlchemy. ○ **Postman** (API testing).

5. Database Management

- **phpMyAdmin** (included in XAMPP).
- **MySQL Workbench** (for advanced queries).

6. Web Browsers for Testing

- **Chrome** (latest version), **Firefox**, **Edge**.
- **Mobile**: Chrome on Android, Safari on iOS (for responsive checks).

7. Additional Tools

- **Git** (version control).
- **Figma** / **Adobe XD** (UI prototyping, optional).

For Running the Project After Implementation

1. Operating System

- Same as development OS (Windows/Linux/macOS).

2. Web Server

- **Production**: Apache/Nginx (with WSGI for Python backend).
- **Local**: XAMPP (for small-scale deployments).

3. Database

- **MySQL** (via XAMPP or cloud-hosted).

4. Web Browsers

- Any modern browser (Chrome/Firefox/Safari/Edge).

Functional Requirements:

1. User Registration & Authentication

Description: Users should be able to create accounts to save their typing test results and track progress.

Key Points:

- Users provide:
    - **Name**
    - **Username**

○ **Password** (hashed)

○ **Email** (optional for password recovery)

- Guests can take tests without registration (results not saved).
- One-click login via **Google/GitHub** (optional).

2. Typing Test Interface

Description: The system should display a random text passage for users to type.

Key Points:

- **Text Passages**:
  - Pull from a database of pre-loaded sentences/paragraphs.
  - Categorized by difficulty (Easy/Medium/Hard).
- **Timer**: ○ Starts automatically on first keystroke. ○ Configurable durations (1/5/10/15 minutes).
- **Live Feedback**:

  ○ Color-coded text (green = correct, red = errors). ○ Real-time WPM/CPM/accuracy updates.

3. Results Calculation

Description: The system must calculate and display typing metrics accurately.

Key Points:

- **Metrics**:
  - **WPM** (Words Per Minute): (Correct characters / 5) / (Time in minutes).
  - **CPM** (Characters Per Minute): Total characters typed / time. ○ **Accuracy**: (Correct keystrokes / Total keystrokes) × 100.
- **Error Analysis**:

  ○ Highlight missed/extra characters. ○ Count repeated errors (e.g., consistent misspelling).

4. Test History & Analytics

Description: Users should view past test results to track improvement.

Key Points:

- **Dashboard**:
  - ○ Graph of WPM/accuracy over time. ○ Average scores by difficulty level.
- **Export Data**:
  - ○ Download results as CSV/PDF (optional).

5. Admin Features

Description: Admins manage test content and user data.

Key Points:

- **Passage Management**:
  - ○ Add/edit/delete typing passages. ○ Tag passages by difficulty/language.
- **User Management**:
  - ○ View/delete user accounts. ○ Reset passwords (if needed).

6. System Configuration

Description: Customize the typing test experience.

Key Points:

- **Themes**: Light/dark mode toggle.
- **Sound Effects**: Enable/disable keystroke sounds.
- **Keyboard Layout**: QWERTY/AZERTY/Dvorak support.

Software Engineering Paradigm: Modified Waterfall Model for Speed Typing Tester

1. Introduction

The **Speed Typing Tester** project will follow a **modified Waterfall Model with feedback mechanisms** to balance structure and flexibility. While maintaining the linear progression of traditional Waterfall, we incorporate iterative feedback loops to accommodate real-world adjustments during development.

2. Key Characteristics for This Project

2.1 Linearity with Feedback Loops

- **Traditional Approach**: Strict sequential phases (Analysis → Design → Coding → Testing → Deployment).
- **Our Adaptation**:

  ○ Retains phase order but allows revisiting prior phases based on feedback. ○ Example: If usability issues arise during testing, the team can refine the UI design before final deployment.

2.2 Flexible Phase Transitions

- **Traditional Rigidity**: No overlap between phases.
- **Our Adaptation**:
  - **Design** and **Coding** phases may overlap once core UI/backend designs are approved.
  - **Testing** begins during coding for critical modules (e.g., WPM calculation).

2.3 Incremental Deliverables

- **Traditional Monolithic Delivery**: Single end-product release.
- **Our Adaptation**:

  ○ Deliver **minimum viable product (MVP)** early (e.g., basic typing test without auth). ○ Add features (user accounts, analytics) in subsequent iterations.

3. Phases with Feedback Mechanisms

Phase 1: Preliminary Investigation

- **Activities**:
  - Define project scope (e.g., "A web app to measure typing speed").
  - Feasibility study (technical, economic, operational).
- **Feedback Loop**:
  - Stakeholders may request additional features (e.g., multilingual support), prompting scope adjustments.

Phase 2: System Analysis

- **Activities**:
  - Finalize **SRS** (Software Requirements Specification).
  - Prioritize features (e.g., real-time WPM vs. saved history).
- **Feedback Loop**:
  - Users/testers review requirements for clarity → refine SRS.

Phase 3: System Design

- **Activities**:
  - **UI/UX**: Wireframes for typing interface and results dashboard.
  - **Database**: Schema for users and test_results tables. ○ **Architecture**: Frontend (HTML/CSS/JS) + Backend (Python/Flask).
- **Feedback Loop**:
  - Developers may suggest simplifying the database design during coding.

Phase 4: Coding

- **Activities**:
  - Frontend: Typing test interface with live metrics. ○ Backend: WPM algorithm, user authentication.
- **Feedback Loop**: ○ Unit tests reveal flaws in WPM logic → revisit design/code.

Phase 5: Testing

- **Activities**:
  - Unit Testing: Validate WPM/accuracy calculations. ○ Integration Testing: User login → Test → Save results.
- **Feedback Loop**: ○ Users report mobile layout issues → tweak responsive design.

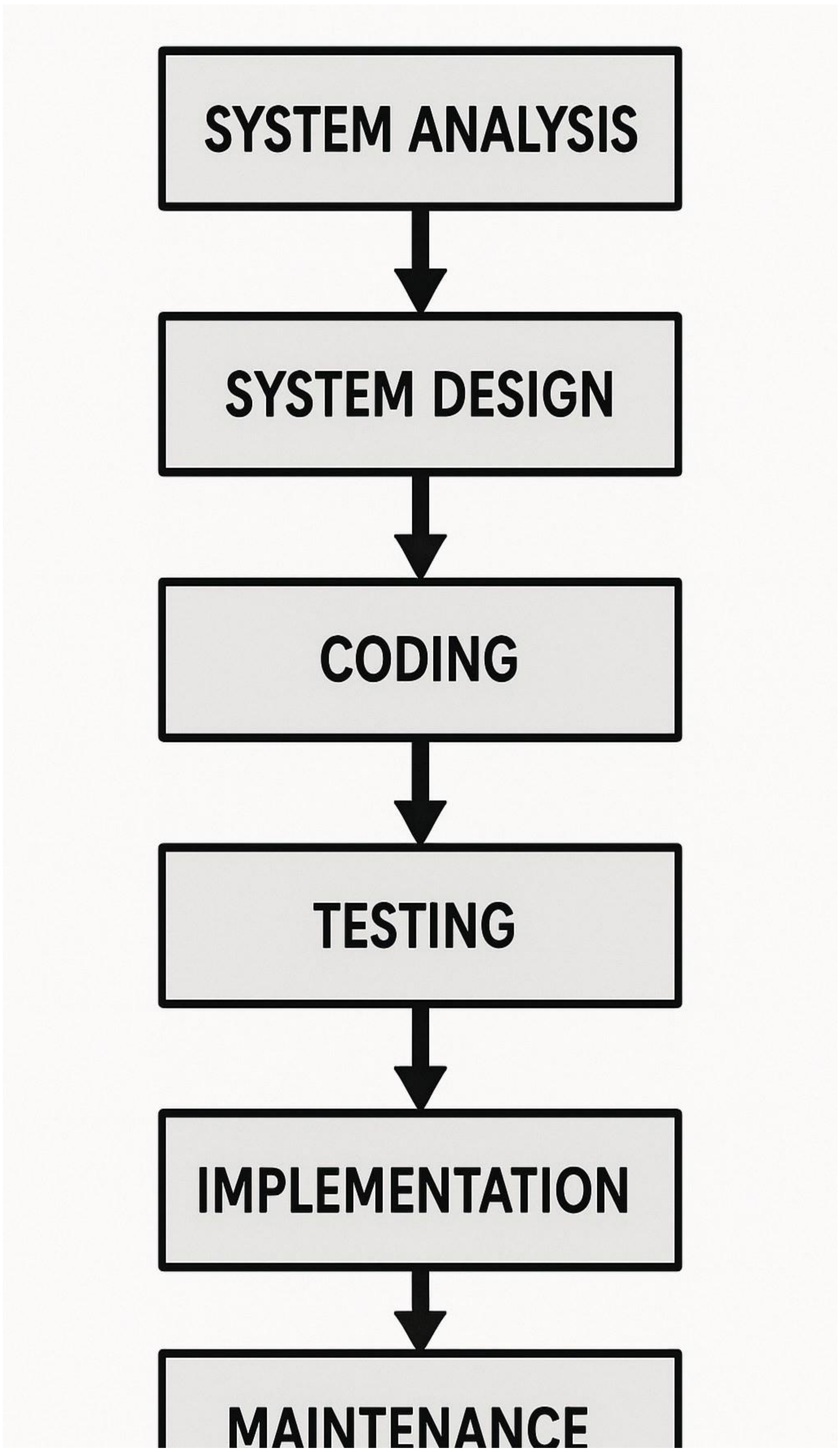
Phase 6: Deployment

- **Activities**:
  - Deploy on XAMPP/local server. o Create user/admin documentation.
- **Feedback Loop**:
  - Post-launch feedback may prompt minor updates (e.g., timer tweaks).

Benefits for Speed Typing Tester

- **Improved Flexibility:**
  Traditional Waterfall models follow a rigid, phase-by-phase approach, making it difficult to implement changes once a stage is completed. In contrast, our modified approach was highly flexible and adaptive. This allowed us to easily respond to user experience (UX) and technical feedback throughout the development. For example, when users pointed out layout issues or inconsistencies in the typing accuracy logic, we could promptly make changes without affecting the overall timeline.
- **Early Risk Detection and Management:**
  In the Waterfall model, most bugs and issues are detected only in the final testing phase, which can lead to significant delays and higher costs. Our approach ensured early and continuous testing, which helped us identify and resolve problems at an early stage. A key example was the detection of a formula error in the WPM (Words Per Minute) calculation. Instead of discovering it late, we were able to fix it during development, resulting in a more stable and accurate application.
- **Continuous Stakeholder Input:**
  Waterfall restricts stakeholder involvement mostly to the early requirement-gathering phase, often leading to a final product that doesn't fully meet user expectations. In our development process, stakeholders such as project guides, peers, and potential users were involved continuously. Their feedback played a crucial role in refining features like dark mode, real-time error highlighting, and typing sound effects, ensuring the application was tailored to actual user needs.
- **Enhanced User Experience (UX):**
  Our approach supported real-time user feedback and iterative UI improvements. As a result, the final application offered a much smoother, more interactive, and user-friendly experience. Additions such as paragraph difficulty levels, a responsive interface, customizable timer options, and typing sound effects all contributed to an engaging typing environment.
- **Efficient Time Management:**
  In Waterfall, late discovery of issues can lead to rework and timeline extensions. Our method ensured continuous integration and testing, which helped resolve issues as they occurred. This proactive approach saved time, reduced rework, and helped us stick to our development schedule.
- **Ease of Feature Expansion:**
  Adding new features in the Waterfall model requires revisiting earlier stages, making the process slow and resource-heavy. Our modified approach allowed

us to expand the feature set incrementally. Functionalities like CSV export of history, real-time leaderboard, custom test durations, and accuracy breakdown graphs were implemented smoothly without needing to overhaul earlier work.


SYSTEM ANALYSIS
SYSTEM DESIGN
CODING
TESTING
IMPLEMENTATION
MAINTENANCE

Analysis Diagrams: Data Flow Diagram (DFD):

Introduction to Data Flow Diagrams (DFD)

A Data Flow Diagram (DFD) is a graphical tool used to visualize the flow of data through a system. It illustrates how data moves between processes, external entities, and data stores. For the Speed Typing Tester system, DFDs help developers and stakeholders understand how user inputs, typing tests, results, and data storage interact to form a complete system.

Components of a Data Flow Diagram (DFD)

1. **Processes**
   - **Definition**: Represent operations that transform or handle data.
   - **Symbol**: Circle or rounded rectangle.
   - **Example**: "Evaluate Typing Test" processes the typed text and calculates speed and accuracy.

2. **External Entities**
   - **Definition**: Entities outside the system that interact with it.
   - **Symbol**: Rectangle.
   - **Example**: The “User” interacting with the Speed Typing Tester.

3. **Data Flow**
   - **Definition**: Shows how data moves between components.
   - **Symbol**: Arrow.
   - **Example**: "Typed Text" flows from User to the Evaluation Process.

4. **Data Stores**
   - **Definition**: Repositories where data is stored.
   - **Symbol**: Two parallel lines.
   - **Example**: "User Test History" stores past test performance.

5. **Privilege Boundary**
   - **Definition**: Differentiates between access levels.
   - **Symbol**: Dashed line.
   - **Example**: Admin and user areas could be separated by a privilege boundary (optional in simple systems).

Types of Data Flow Diagrams:

1. **Context-Level DFD (Level 0 DFD)**

- **Purpose**: Displays the overall system as one process and its interaction with external entities.
- **Entities**:
  User (provides login info, takes typing test)
  Admin (views statistics or manages paragraphs)
- **Processes**:
- **Speed Typing Tester System**
- **Data Flows**:
  User → System: Registration/Login Details, Typing Input
  System → User: Test Interface, Results
  Admin → System: Paragraph Management
  System → Admin: User Statistics

2. **Level 1 DFD**
   ○ **Processes**:
   - **1.1 User Authentication** – Validates login/registration credentials
   - **1.2 Typing Test Engine** – Displays paragraphs, tracks typing input
   - **1.3 Result Evaluation** – Calculates WPM, accuracy, errors
   - **1.4 History Management** – Stores and retrieves user test history ○ **Data Stores**:
   - **D1: User Database** – Stores user credentials
   - **D2: Test History** – Stores test results
   - **D3: Paragraph Bank** – Stores paragraphs of varying difficulty ○ **Data Flows**:
   - User → 1.1 → D1 → 1.2 → D3
   - 1.2 → 1.3 (with typing data)
   - 1.3 → D2 (store result) → User (display result)

3. **Level 2 DFD (Example: Result Evaluation)**
   ○ **Processes**:
   - **2.1 Calculate Time** – Calculates duration of typing test
   - **2.2 Compare Text** – Compares user input with target paragraph
   - **2.3 Calculate WPM & Accuracy** – Computes performance metrics
   - **2.4 Save Result** – Saves result to database ○ **Data Flows**:
   - Typing Input → 2.1 → 2.2 → 2.3 → 2.4 → Test History

Advantages of Data Flow Diagrams for This Project:

1. **Context-Level DFD (Level 0 DFD)**

- **Simplified Overview:**
  Provides a **high-level view** of the entire system, making it easy for non-technical stakeholders to understand how users and admins interact with the system.
- **Clear Entity Relationships:**
  Highlights **external entities** (User and Admin) and their direct interactions with the Speed Typing Tester System.
- **Communication Aid:**
  Useful for initial project discussions and **requirement analysis** with clients, as it doesn't go into technical depth.

2. Level 1 DFD

- **Functional Breakdown:**
  Breaks down the system into **major functional components** like authentication, test engine, result evaluation, and history, giving a more detailed view of system operations.
- **Data Handling Clarity:**
  Shows how **data flows between processes and databases**, helping developers understand where data is coming from and going.
- **System Design Blueprint:**
  Acts as a **blueprint for system architecture**, aiding in structured coding and module development.

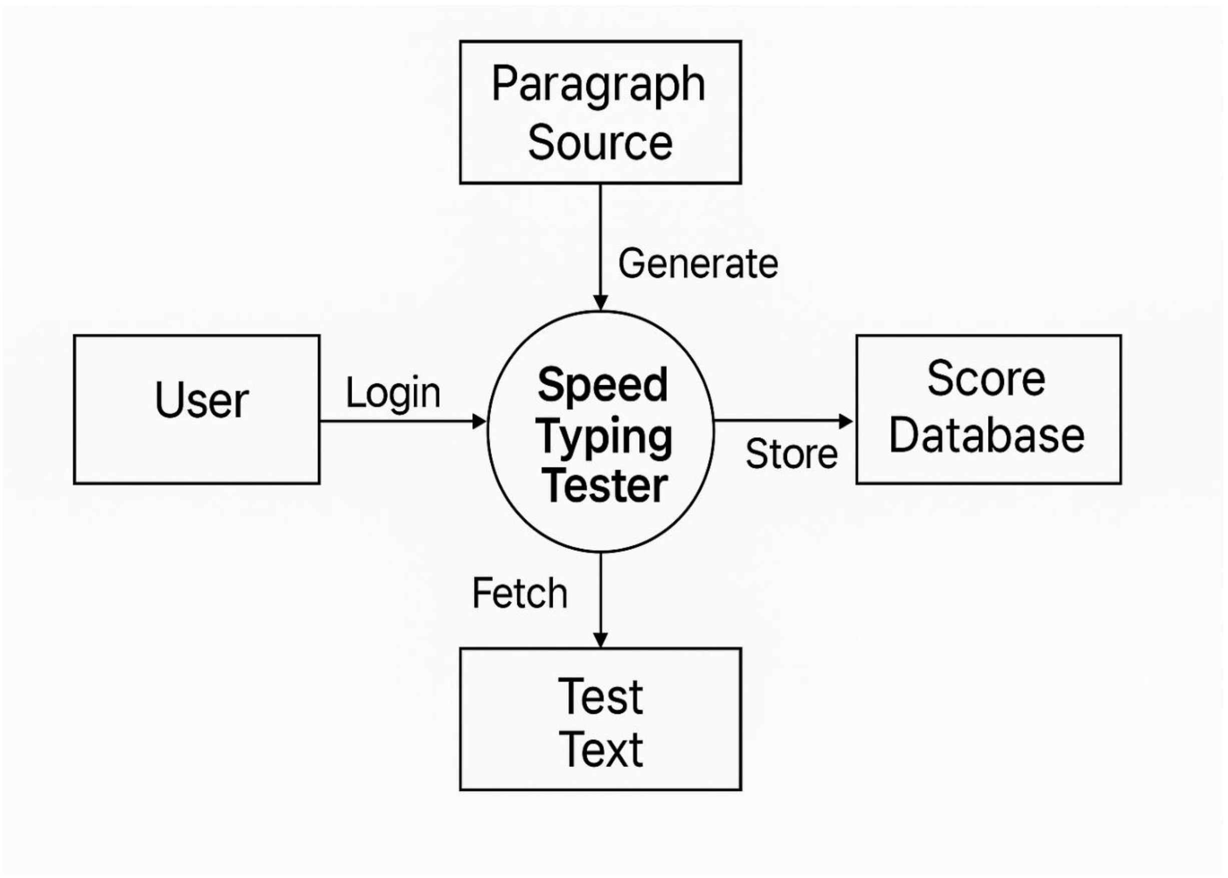
3. Level 2 DFD (Result Evaluation Module)

- **Detailed Process Analysis:**
  Focuses on **internal workings** of a specific process (like result evaluation), allowing in-depth understanding of how WPM and accuracy are calculated.
- **Debugging Support:**
  Makes it easier to **troubleshoot and refine** specific processes during development.
- **Improved Accuracy & Performance:**
  Helps in optimizing key performance areas like timing and result calculations by clearly visualizing **step-by-step logic**.

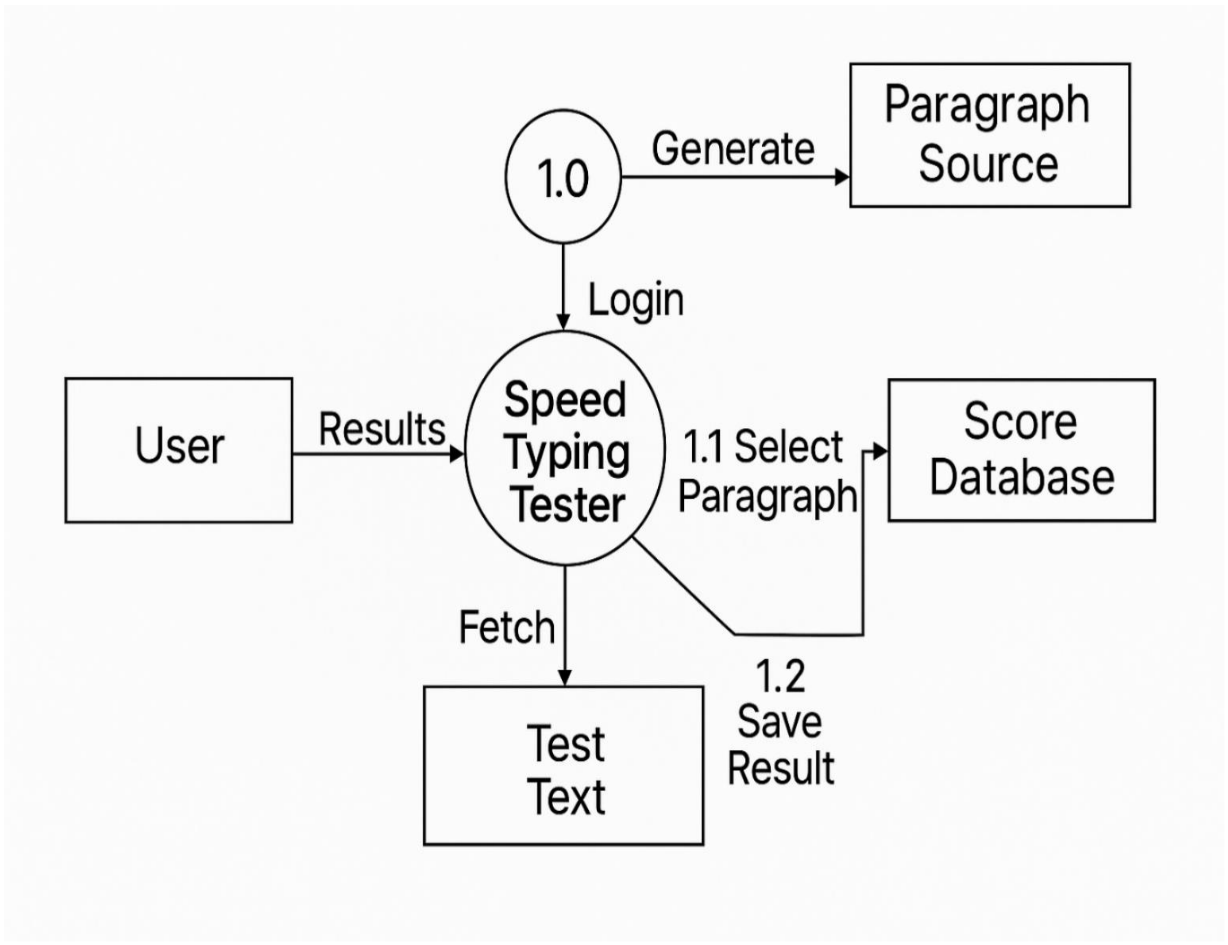
Conclusion

In the Speed Typing Tester system, DFDs provide a clear visualization of how data flows from user interaction to performance evaluation and history storage. They help structure the project into logical steps and improve both the design and implementation phases. By examining multiple levels of DFDs, developers can ensure a thorough understanding of system behavior and data movement, leading to better performance and usability.

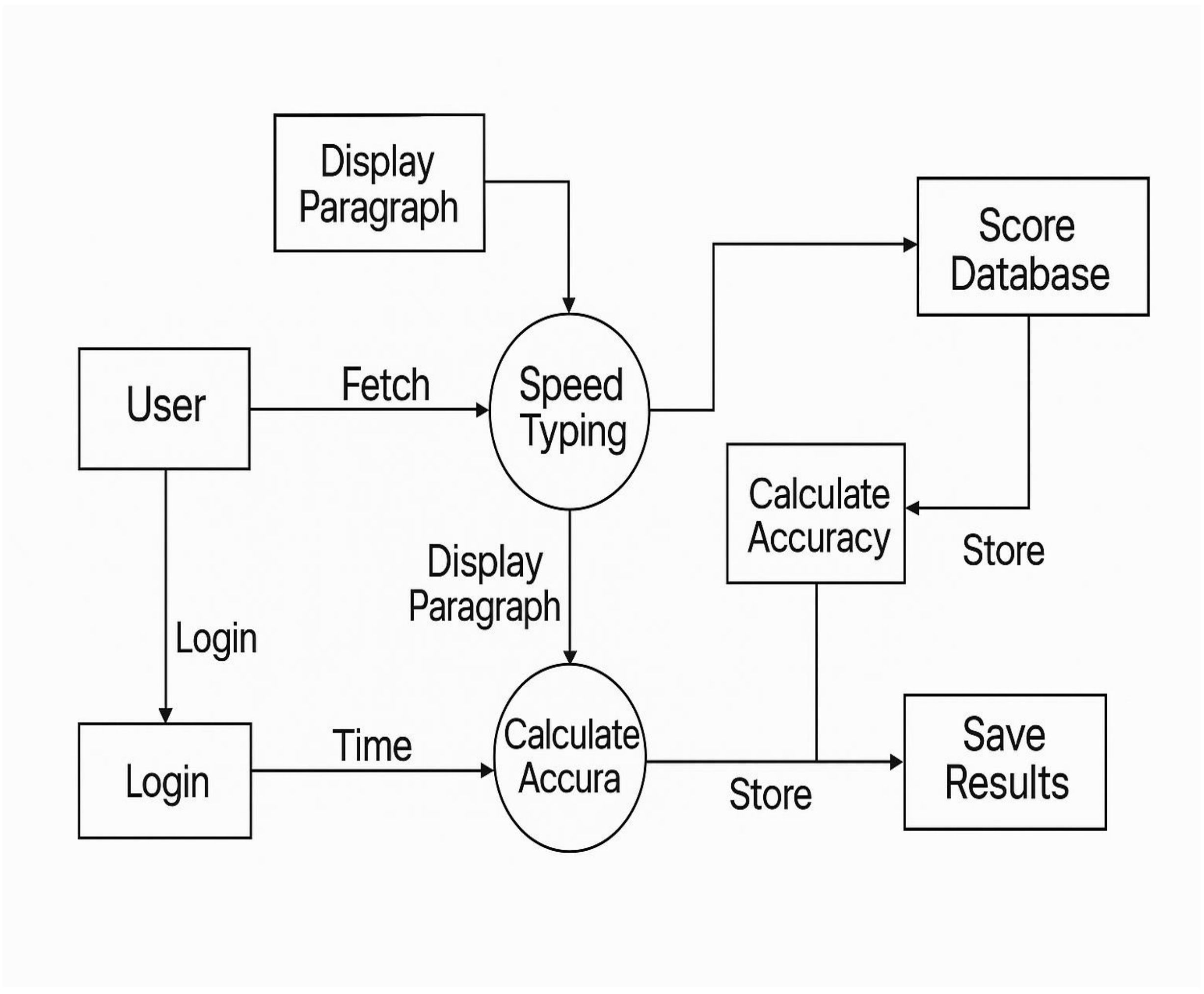
Level 0 DFD Diagram:



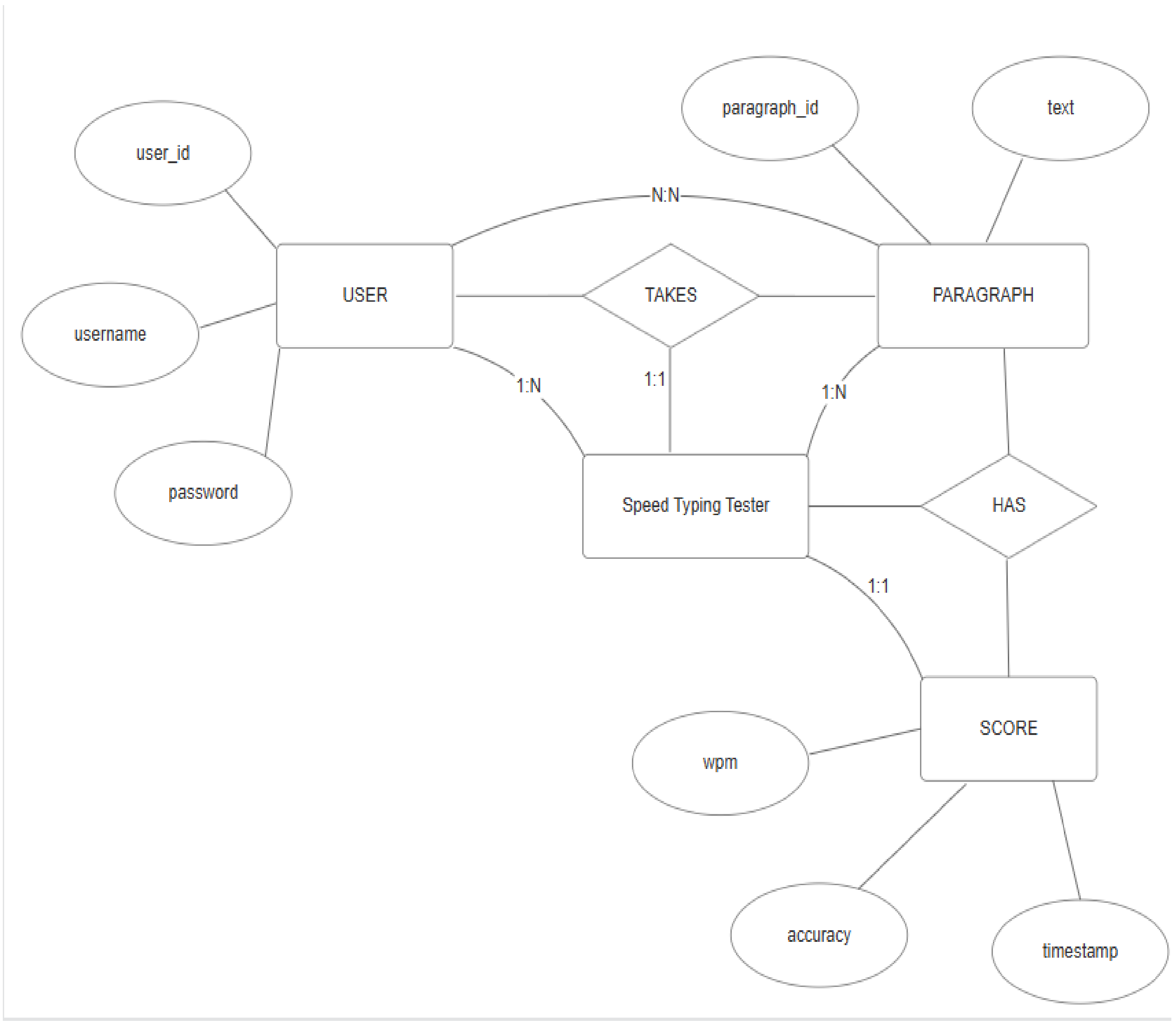
Level 1 DFD Diagram:



Level 2 DFD Diagram:



ER Diagram:



paragraph_id
text
user_id
N:N
USER
TAKES
PARAGRAPH
username
1:N
1:1
1:N
password
Speed Typing Tester
HAS
1:1
SCORE
wpm
accuracy
timestamp

SYSTEM DESIGN:

User Registration & Login
Users can create accounts using their name, email, and password. After registration, they can securely log in to access typing tests and view their results.

b) Paragraph/Text Management
This module manages the storage and retrieval of test paragraphs. Paragraphs can vary by difficulty and are randomly selected when a user starts a test.

c) Typing Test Execution
Once logged in, users can start a typing test. This module handles timer control, live input tracking, and real-time typing accuracy checking.

d) Result Calculation
At the end of the test, the system calculates WPM (words per minute), accuracy percentage, and number of mistakes, and displays the results to the user.

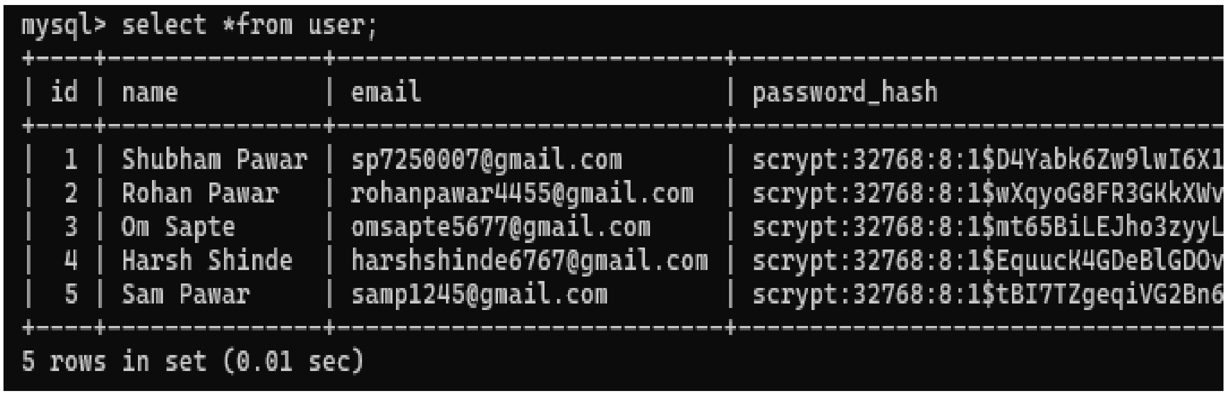
e) Score History & Leaderboard
Users can view their past typing test results. The system maintains a record of each user's performance and optionally displays a leaderboard showing top scorers.

f) Admin Interface (Optional)
Admins can manage user data, upload or edit sample paragraphs, and monitor typing statistics and test logs.

g) Security & Session Management
Passwords are securely hashed, and user sessions are managed to prevent unauthorized access. Pasting text in the typing box is disabled to ensure fair testing.

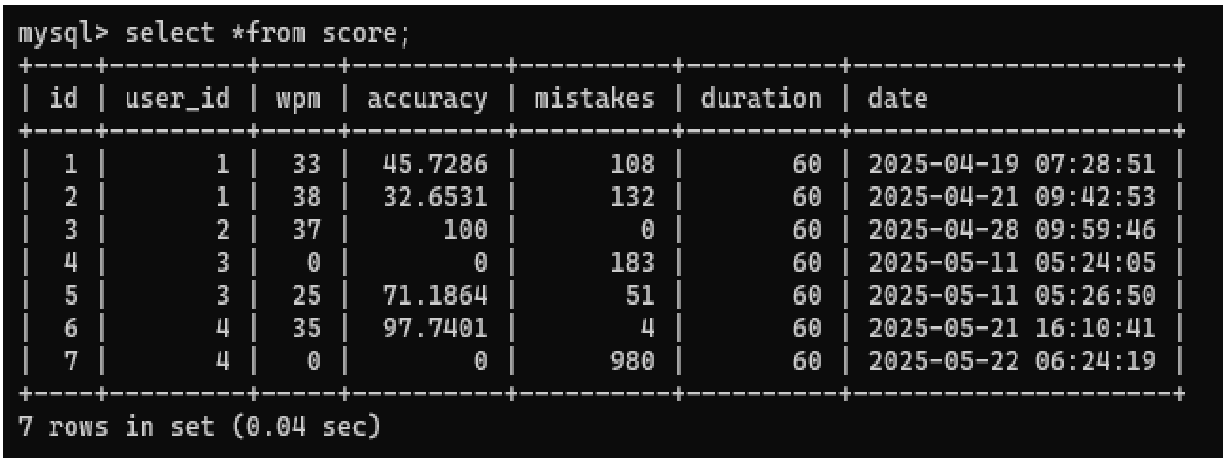
DATA STRUCTURE OF ALL MODULES:

User Table-



```
mysql> select *from user;
+----+---------------+---------------------------+-----------------------------------
| id | name          | email                     | password_hash
+----+---------------+---------------------------+-----------------------------------
|  1 | Shubham Pawar | sp7250007@gmail.com       | scrypt:32768:8:1$D4Yabk6Zw9lwI6X1
|  2 | Rohan Pawar   | rohanpawar4455@gmail.com  | scrypt:32768:8:1$wXqyoG8FR3GKkXWv
|  3 | Om Sapte      | omsapte5677@gmail.com     | scrypt:32768:8:1$mt65BiLEJho3zyyL
|  4 | Harsh Shinde  | harshshinde6767@gmail.com | scrypt:32768:8:1$EquucK4GDeBlGDOv
|  5 | Sam Pawar     | samp1245@gmail.com        | scrypt:32768:8:1$tBI7TZgeqiVG2Bn6
+----+---------------+---------------------------+-----------------------------------
5 rows in set (0.01 sec)
```

Score Table-



```
mysql> select *from score;
+----+---------+-----+----------+----------+----------+---------------------+
| id | user_id | wpm | accuracy | mistakes | duration | date                |
+----+---------+-----+----------+----------+----------+---------------------+
|  1 |       1 |  33 |  45.7286 |      108 |       60 | 2025-04-19 07:28:51 |
|  2 |       1 |  38 |  32.6531 |      132 |       60 | 2025-04-21 09:42:53 |
|  3 |       2 |  37 |      100 |        0 |       60 | 2025-04-28 09:59:46 |
|  4 |       3 |   0 |        0 |      183 |       60 | 2025-05-11 05:24:05 |
|  5 |       3 |  25 |  71.1864 |       51 |       60 | 2025-05-11 05:26:50 |
|  6 |       4 |  35 |  97.7401 |        4 |       60 | 2025-05-21 16:10:41 |
|  7 |       4 |   0 |        0 |      980 |       60 | 2025-05-22 06:24:19 |
+----+---------+-----+----------+----------+----------+---------------------+
7 rows in set (0.04 sec)
```

PROCEDURAL DESIGN:

PROCESS LOGIC (FLOWCHART DESCRIPTIONS) OF EACH MODULE

1. User Registration Module

- Start → Input: Name, Email, Password
- Check if Email already exists
    - Yes → Show error message
    - No → Hash password
- Store user data in database
- Show registration success message → End

2. Login Module

- Start → Input: Email, Password
- Fetch user by email
- Compare password with stored hash
    - Match → Redirect to typing test page
    - No Match → Show error → End
- Start session → End

3. Typing Test Module

- Start → Load sample paragraph
- Start timer
- User types text → Live display of WPM and accuracy
- Time completes or user submits
- Calculate WPM, accuracy, mistakes → Show result → End

4. Score Saving Module

- Start → Collect user test data (WPM, accuracy, etc.)
- Store in database
- Redirect to history or show confirmation message → End

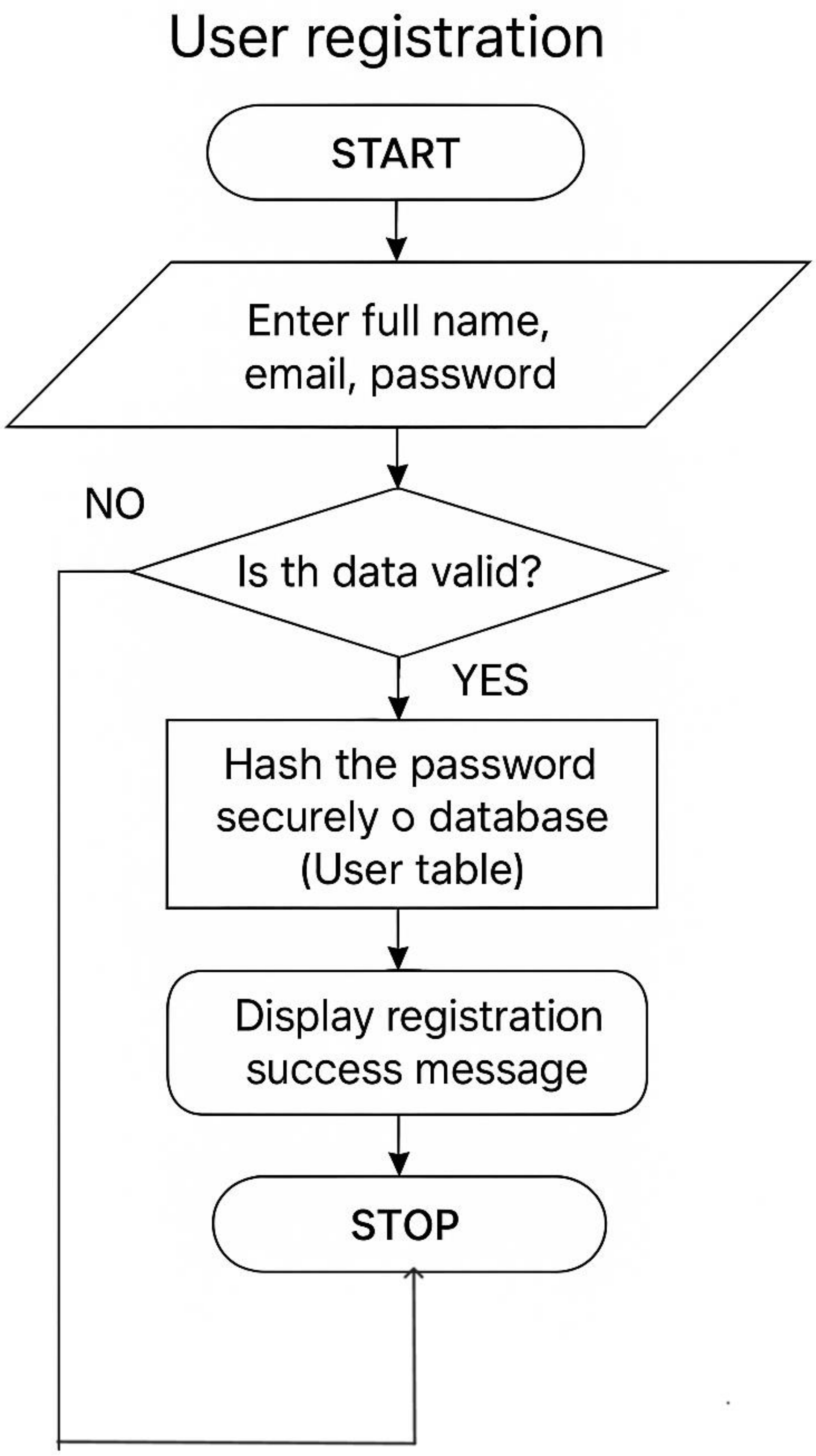
5. History Module

- Start → User requests history
- Fetch score records from database
- Display results in a table → End

6. Security Module

- Start typing → Check if user tries to paste
    - If paste detected → Block action, show alert
- Continue typing normally → End

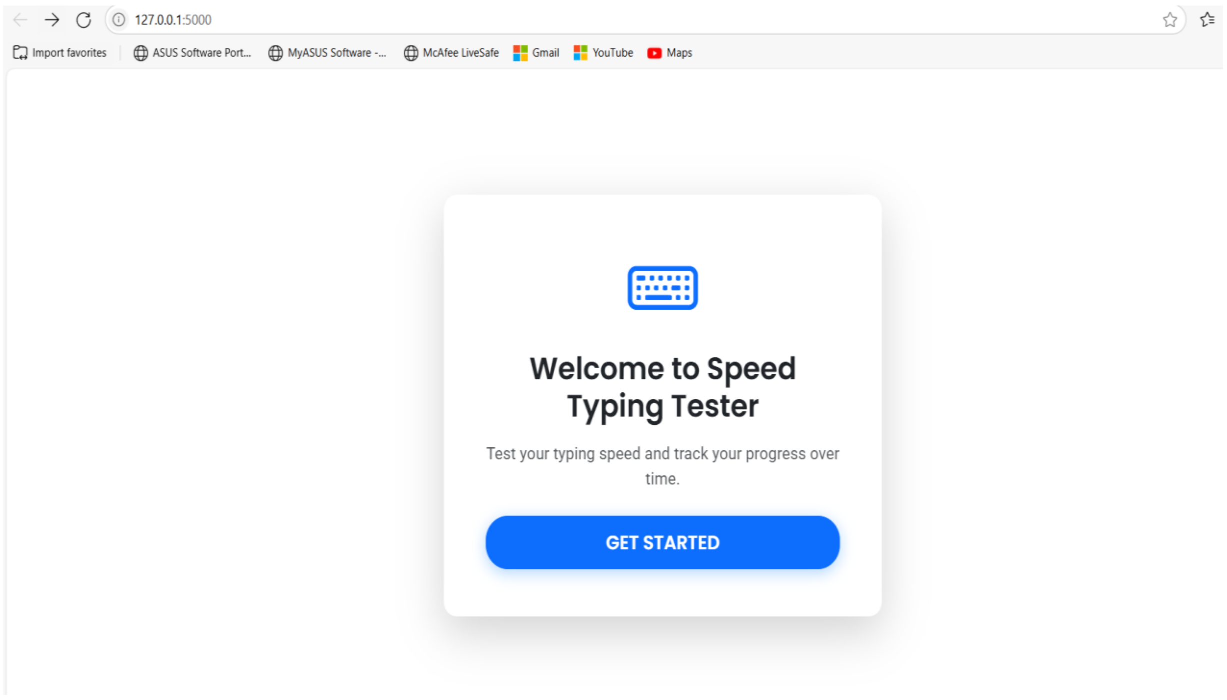
Flow chart for **User Registration Module:**



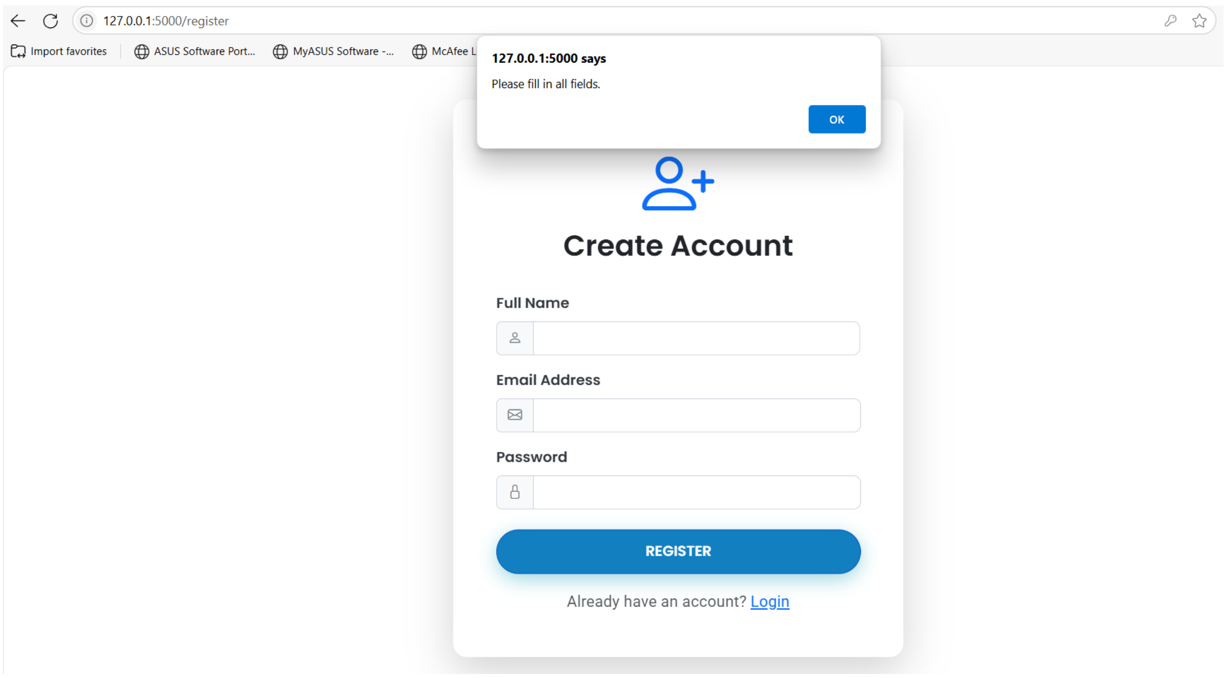
User registration
START
Enter full name, email, password
NO
Is th data valid?
YES
Hash the password securely o database (User table)
Display registration success message
STOP

SCREENSHOTS:

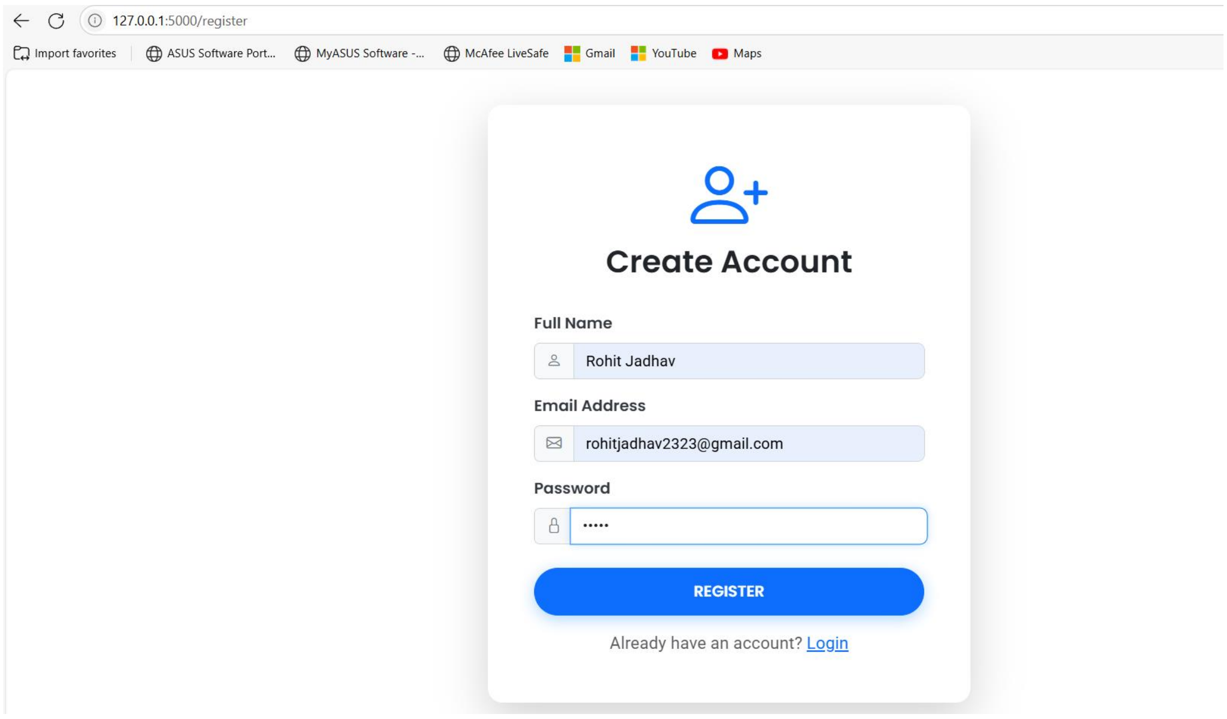
Home Page:



Register Page with Validation:



Register Page:


127.0.0.1:5000/register

Import favorites | ASUS Software Port... | MyASUS Software -... | McAfee LiveSafe | Gmail | YouTube | Maps

Create Account

Full Name

Rohit Jadhav

Email Address

rohitjadhav2323@gmail.com

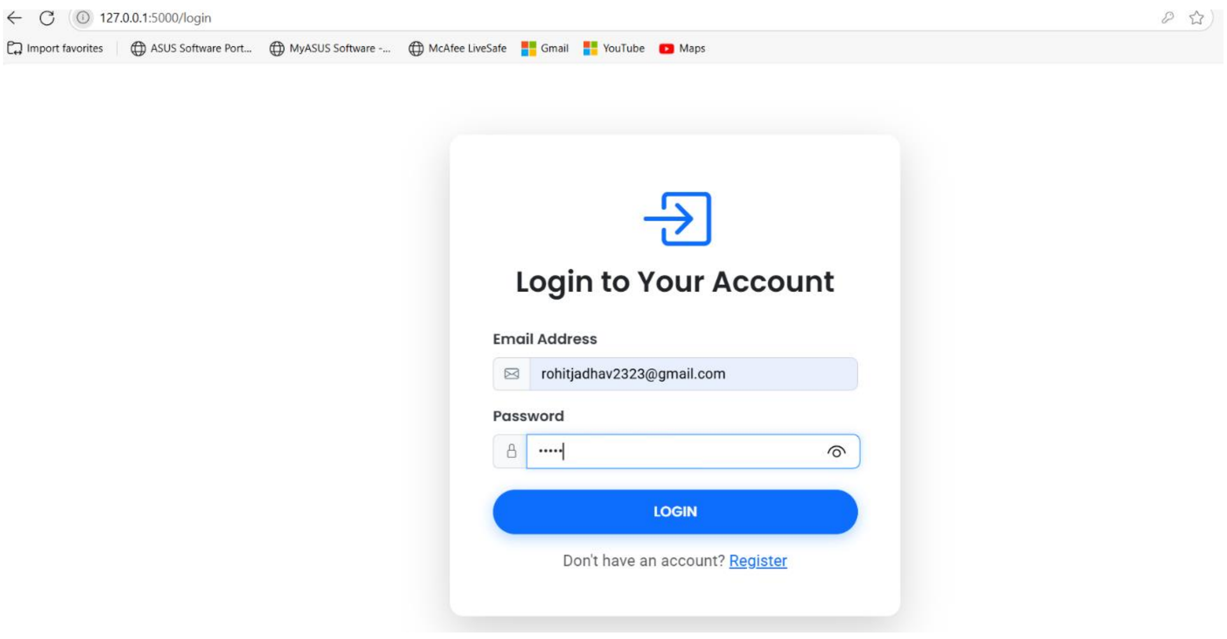
Password

•••••

REGISTER

Already have an account? Login

Login Page:


127.0.0.1:5000/login

Import favorites | ASUS Software Port... | MyASUS Software -... | McAfee LiveSafe | Gmail | YouTube | Maps

Login to Your Account

Email Address

rohitjadhav2323@gmail.com

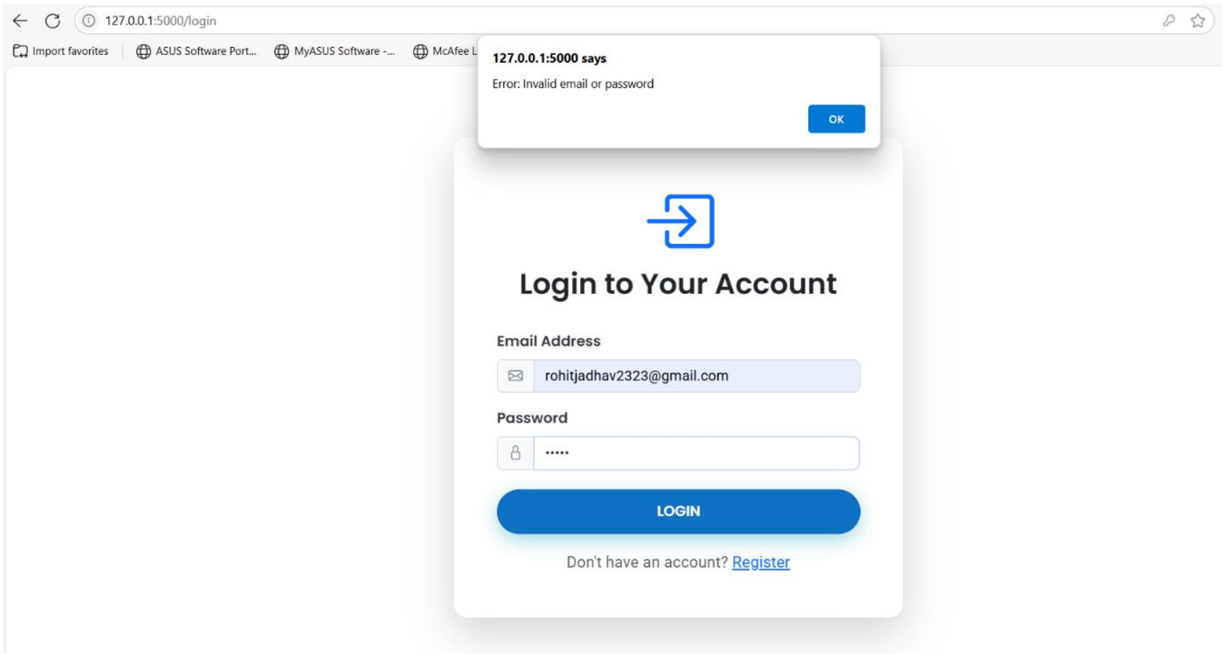
Password

•••••

LOGIN

Don't have an account? Register

Login Page with Validation:



127.0.0.1:5000/login

127.0.0.1:5000 says

Error: Invalid email or password

OK

Login to Your Account

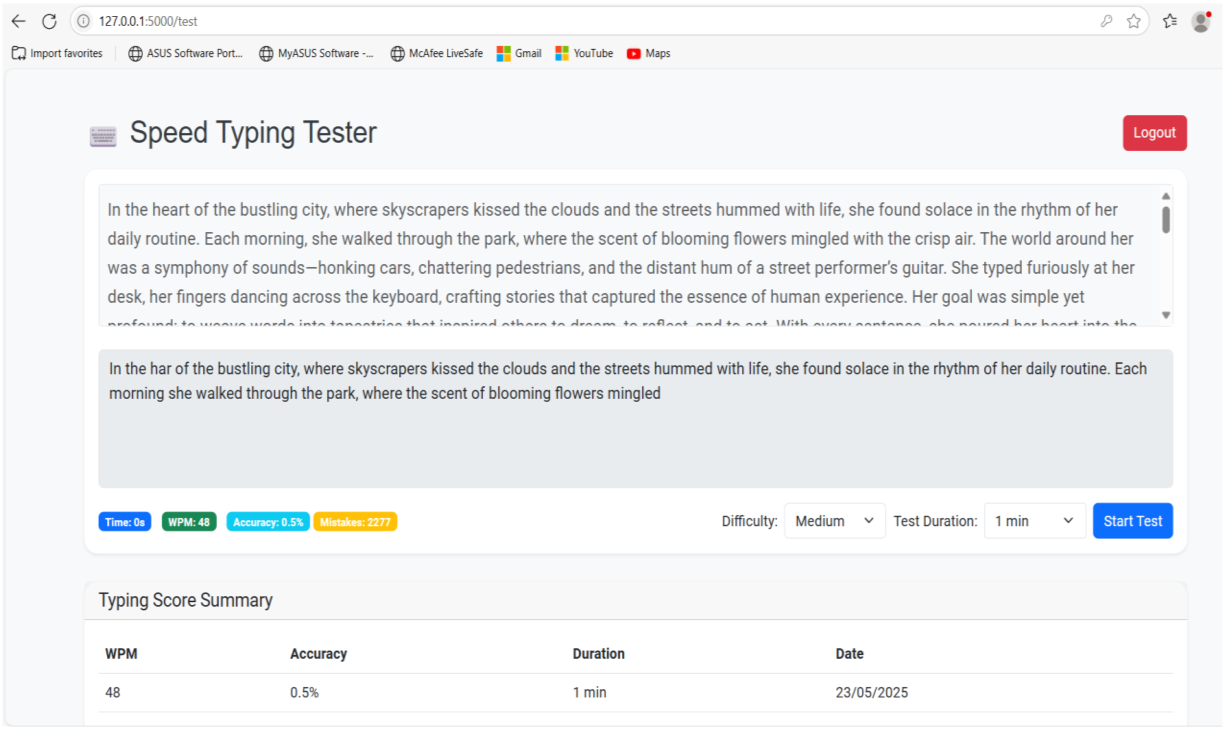
Email Address: rohitjadhav2323@gmail.com

Password: •••••

LOGIN

Don't have an account? Register

Result Page:



127.0.0.1:5000/test

Speed Typing Tester

Logout

In the heart of the bustling city, where skyscrapers kissed the clouds and the streets hummed with life, she found solace in the rhythm of her daily routine. Each morning, she walked through the park, where the scent of blooming flowers mingled with the crisp air. The world around her was a symphony of sounds—honking cars, chattering pedestrians, and the distant hum of a street performer's guitar. She typed furiously at her desk, her fingers dancing across the keyboard, crafting stories that captured the essence of human experience. Her goal was simple yet

In the har of the bustling city, where skyscrapers kissed the clouds and the streets hummed with life, she found solace in the rhythm of her daily routine. Each morning she walked through the park, where the scent of blooming flowers mingled

Time: 0s | WPM: 48 | Accuracy: 0.5% | Mistakes: 2277

Difficulty: Medium | Test Duration: 1 min | Start Test

Typing Score Summary

WPM	Accuracy	Duration	Date
48	0.5%	1 min	23/05/2025

CODING

Script.js

```
let timer;
let timeLeft = 60;
let isTestActive = false;
let startTime;
let sampleText = "";

const startBtn = document.getElementById("start-btn");
const inputText = document.getElementById("input-text");
const sampleTextElement = document.getElementById("sample-text");
const timerElement = document.getElementById("timer");
const wpmElement = document.getElementById("wpm");
const accuracyElement = document.getElementById("accuracy");
const mistakesElement = document.getElementById("mistakes");
const timeSelect = document.getElementById("time-select");
const difficultySelect = document.getElementById("difficulty-select");

async function getNewText() {
  const difficulty = difficultySelect.value;
  const response = await fetch(/api/get-text?difficulty=${difficulty});
  const data = await response.json();
  return data.text;
}

async function loadUserScores() {
  const user = JSON.parse(localStorage.getItem("user"));
  if (!user || !user.id) {
```

```
  window.location.href = "{{ url_for('login_page') }}";
  return;
}

try {
  const response = await fetch(/api/scores/${user.id});
  if (!response.ok) throw new Error("Failed to load scores");

  const scores = await response.json();
  const scoresTable = document.getElementById("scores-table");

  scoresTable.innerHTML = scores
    .map(
      (score) => `
        <tr>
          <td>${score.wpm}</td>
          <td>${score.accuracy.toFixed(1)}%</td>
          <td>${score.duration / 60} min</td>
          <td>${new Date(score.date).toLocaleDateString("en-IN", {
            timeZone: "UTC",
            year: "numeric",
            month: "2-digit",
            day: "2-digit",
          })}</td>
        </tr>
      `
    )
    .join("");
} catch (error) {
  console.error("Error loading scores:", error);
  }
}

function calculateWPM(inputLength, timeElapsed) {
  const words = inputLength / 5; // Standard: 5 characters = 1 word
  const minutes = timeElapsed / 60;
  return Math.round(words / minutes);
}

function calculateAccuracy(input, sample) {
  let correct = 0;
  const inputLength = Math.min(input.length, sample.length);

  for (let i = 0; i < inputLength; i++) {
    if (input[i] === sample[i]) correct++;
  }

  return (correct / sample.length) * 100;
}

function calculateMistakes(input, original) {
  let mistakes = 0;
  const minLength = Math.min(input.length, original.length);

  for (let i = 0; i < minLength; i++) {
    if (input[i] !== original[i]) {
      mistakes++;
    }
  }
  mistakes += Math.abs(input.length - original.length);

  return mistakes;
}

async function startTest() {
  const selectedDuration = parseInt(timeSelect.value);
  console.log("Selected duration:", selectedDuration, "seconds");

  timeLeft = selectedDuration;
  timerElement.textContent = timeLeft;
  console.log("Initial timeLeft set to:", timeLeft);

  sampleText = await getNewText();
  sampleTextElement.textContent = sampleText;
  inputText.value = "";
  inputText.disabled = false;
  inputText.focus();

  isTestActive = true;
  startTime = new Date();
  startBtn.disabled = true;

  if (timer) {
    clearInterval(timer);
  }

  timer = setInterval(() => {
    timeLeft--;
    timerElement.textContent = timeLeft;
    console.log("Time left:", timeLeft);

    if (timeLeft <= 0) {
      endTest();
    }
  }, 1000);
}

async function endTest() {
  clearInterval(timer);
  isTestActive = false;
  inputText.disabled = true;
  startBtn.disabled = false;

  const input = inputText.value;
  const selectedDuration = parseInt(timeSelect.value);
  const duration = selectedDuration - timeLeft;

  const wpm = calculateWPM(input.length, duration);
  const accuracy = calculateAccuracy(input, sampleText);
  const mistakes = calculateMistakes(input, sampleText);

  wpmElement.textContent = wpm;
  accuracyElement.textContent = accuracy.toFixed(1);
  mistakesElement.textContent = mistakes;

  const user = JSON.parse(localStorage.getItem("user"));

  if (!user) {
    alert("User not found. Please log in again.");
```

```
      window.location.href = "{{ url_for('login_page') }}";
      return;
    }

    await fetch("/api/save-score", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        userId: user.id,
        wpm,
        accuracy,
        mistakes,
        duration: selectedDuration,
      }),
    });

    await loadUserScores();
  }

startBtn.addEventListener("click", startTest);
inputText.addEventListener("input", () => {
  if (!isTestActive) return;

  const duration = (new Date() - startTime) / 1000;
  const wpm = calculateWPM(inputText.value.length, duration);
  const accuracy = calculateAccuracy(inputText.value, sampleText);

  wpmElement.textContent = wpm;
  accuracyElement.textContent = accuracy.toFixed(1);
});

inputText.addEventListener("paste", (e) => {
  e.preventDefault();
  alert("Pasting is not allowed in the typing test!");
});

loadUserScores();

timeSelect.addEventListener("change", () => {
  timeLeft = parseInt(timeSelect.value);
  timerElement.textContent = timeLeft;
  console.log("Time select changed to:", timeLeft);
});
```

Style.css

```
/* General Styles */
body {
    min-height: 100vh;
    font-family: 'Roboto', sans-serif;
}

/* Background Gradient */
.bg-gradient {
    background: linear-gradient(135deg, #e3f2fd 0%, #ffffff 100%);
}
```

```
/* Card Styling */
.card {
    border: none;
    border-radius: 15px;
    background-color: #ffffff;
    transition: transform 0.3s ease;
}

.card:hover {
    transform: translateY(-5px);
}

/* Typography */
h1, h3 {
    font-family: 'Poppins', sans-serif;
    font-size: 2rem;
    color: #343a40;
}

p {
    font-size: 1.1rem;
}

/* Custom Button */
.custom-btn {
    padding: 12px 30px;
    font-family: 'Poppins', sans-serif;
    font-weight: 600;
    text-transform: uppercase;
    border-radius: 25px;
```

```
  box-shadow: 0 4px 15px rgba(0, 123, 255, 0.3);
  transition: all 0.3s ease;
}

.custom-btn:hover {
  background-color: #17a2b8;
  box-shadow: 0 6px 20px rgba(23, 162, 184, 0.4);
  transform: translateY(-2px);
}

/* Form Input Styling */
.form-label {
  font-family: 'Poppins', sans-serif;
  font-size: 1rem;
  color: #343a40;
}

.custom-input {
  border-radius: 0.5rem;
  transition: all 0.3s ease;
  border: 1px solid #ced4da;
}

.custom-input:focus {
  border-color: #007bff;
  box-shadow: 0 0 8px rgba(0, 123, 255, 0.2);
  transform: scale(1.02);
}

.input-group-text {
  background-color: #f8f9fa;
  border: 1px solid #ced4da;
  color: #6c757d;
}

/* Existing Styles (for test.html, index.html, register.html, etc.) */
#sample-text {
  font-size: 1.2rem;
  line-height: 1.6;
  padding: 10px;
  border: 1px solid #dee2e6;
  border-radius: 0.25rem;
  background-color: #f8f9fa;
}

#input-text {
  font-size: 1.1rem;
  resize: none;
}

.correct {
  color: #198754;
}

.incorrect {
  color: #dc3545;
  text-decoration: underline wavy #dc3545;
}

#time-select {
  font-size: 1rem;
  padding: 0.375rem 0.75rem;
  border-radius: 0.25rem;
}
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Welcome | Speed Typing Tester</title>
    <!-- Bootstrap CSS -->
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
      rel="stylesheet"
    />
    <!-- Google Fonts -->
    <link

href="https://fonts.googleapis.com/css2?family=Poppins:wght@600&family=Roboto:wght@4
00&display=swap"
      rel="stylesheet"
    />
    <!-- Animate.css -->
    <link
      href="https://cdn.jsdelivr.net/npm/animate.css@4.1.1/animate.min.css"
      rel="stylesheet"
    />
    <!-- Bootstrap Icons -->
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.5/font/bootstrap-icons.css"
      rel="stylesheet"
    />
    <!-- Custom CSS -->
    <link
      rel="stylesheet"
      href="{{ url_for('static', filename='style.css') }}"
    />
  </head>
  <body class="bg-gradient">
    <div class="container d-flex align-items-center justify-content-center min-vh-100">
      <div class="card shadow-lg p-5 text-center animate_animated animate_fadeInUp"
style="max-width: 500px;">
        <i class="bi bi-keyboard display-1 text-primary mb-3"></i>
        <h1 class="mb-3 fw-bold text-dark">Welcome to Speed Typing Tester</h1>
        <p class="text-muted mb-4">
          Test your typing speed and track your progress over time.
        </p>
        <a href="/register" class="btn btn-primary btn-lg custom-btn">Get Started</a>
      </div>
    </div>
    <script src="{{ url_for('static', filename='script.js') }}"></script>
  </body>
</html>
```

Login.html

```
<!DOCTYPE html>
```

```
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Login | Speed Typing Tester</title>
    <!-- Bootstrap CSS -->
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
      rel="stylesheet"
    />
    <!-- Google Fonts -->
    <link

href="https://fonts.googleapis.com/css2?family=Poppins:wght@600&family=Roboto:wght@4
00&display=swap"
      rel="stylesheet"
    />
    <!-- Animate.css -->
    <link
      href="https://cdn.jsdelivr.net/npm/animate.css@4.1.1/animate.min.css"
      rel="stylesheet"
    />
    <!-- Bootstrap Icons -->
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.5/font/bootstrap-icons.css"
      rel="stylesheet"
    />
    <!-- Custom CSS -->
    <link
      rel="stylesheet"
      href="{{ url_for('static', filename='style.css') }}"
    />
  </head>
  <body class="bg-gradient">
    <div class="container d-flex align-items-center justify-content-center min-vh-100">
      <div class="card shadow-lg p-5 animate_animated animate_fadeInUp" style="max-width:
500px; width: 100%;">
        <div class="text-center mb-4">
          <i class="bi bi-box-arrow-in-right display-1 text-primary mb-3"></i>
          <h3 class="fw-bold text-dark">Login to Your Account</h3>
        </div>
        <form>
          <div class="mb-3">
            <label for="email" class="form-label">Email Address</label>
            <div class="input-group">
              <span class="input-group-text"><i class="bi bi-envelope"></i></span>
              <input
                type="email"
                class="form-control custom-input"
                id="email"
                name="email"
                required
              />
            </div>
          </div>
          <div class="mb-4">
            <label for="password" class="form-label">Password</label>
            <div class="input-group">
              <span class="input-group-text"><i class="bi bi-lock"></i></span>
              <input
                type="password"
                class="form-control custom-input"
                id="password"
                name="password"
                required
              />
            </div>
          </div>
          <button
            type="submit"
            class="btn btn-primary w-100 custom-btn"
            onclick="loginUser(event)"
          >
            Login
          </button>
        </form>
        <p class="mt-3 mb-0 text-center text-muted">
          Don't have an account? <a href="/register" class="text-primary">Register</a>
        </p>
      </div>
    </div>
    <script>
      async function loginUser(event) {
        event.preventDefault();

        const email = document.getElementById("email").value.trim();
        const password = document.getElementById("password").value;

        const payload = { email, password };

        const isEmptyInput = Object.values(payload).some(
          (value) => value === "" || value === null
```

```
);

if (isEmptyInput) {
  alert("Please fill in all fields.");
  return;
}

const isValidEmail = /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);

if (!isValidEmail) {
  alert("Please enter a valid email address.");
  return;
}

try {
  const response = await fetch("/api/login", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(payload),
  });

  const data = await response.json();

  if (response.ok) {
    localStorage.setItem("user", JSON.stringify(data));
    window.location.href = "/test";
  } else {
    alert("Error: " + data.error);
```

```
          }
        } catch (error) {
          console.error("Login error:", error);
          alert("Something went wrong. Please try again later.");
        }
      }
    </script>
  </body>
</html>
```

Register.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Register | Speed Typing Tester</title>
    <!-- Bootstrap CSS -->
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
      rel="stylesheet"
    />
    <!-- Google Fonts -->
    <link

href="https://fonts.googleapis.com/css2?family=Poppins:wght@600&family=Roboto:wght@4
00&display=swap"
      rel="stylesheet"
    />
    <!-- Animate.css -->
    <link
      href="https://cdn.jsdelivr.net/npm/animate.css@4.1.1/animate.min.css"
      rel="stylesheet"
    />
    <!-- Bootstrap Icons -->
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.5/font/bootstrap-icons.css"
      rel="stylesheet"
    />
    <!-- Custom CSS -->
    <link
      rel="stylesheet"
      href="{{ url_for('static', filename='style.css') }}"
    />
  </head>
  <body class="bg-gradient">
    <div class="container d-flex align-items-center justify-content-center min-vh-100">
      <div class="card shadow-lg p-5 animate_animated animate_fadeInUp" style="max-width:
500px; width: 100%;">
        <div class="text-center mb-4">
          <i class="bi bi-person-plus display-1 text-primary mb-3"></i>
          <h3 class="fw-bold text-dark">Create Account</h3>
        </div>
        <form>
          <div class="mb-3">
            <label for="name" class="form-label">Full Name</label>
            <div class="input-group">
              <span class="input-group-text"><i class="bi bi-person"></i></span>
              <input
                type="text"
                class="form-control custom-input"
                id="name"
                name="name"
                required
              />
            </div>
          </div>
          <div class="mb-3">
            <label for="email" class="form-label">Email Address</label>
            <div class="input-group">
              <span class="input-group-text"><i class="bi bi-envelope"></i></span>
              <input
                type="email"
                class="form-control custom-input"
                id="email"
                name="email"
                required
              />
            </div>
          </div>
          <div class="mb-4">
            <label for="password" class="form-label">Password</label>
            <div class="input-group">
              <span class="input-group-text"><i class="bi bi-lock"></i></span>
              <input
                type="password"
                class="form-control custom-input"
                id="password"
                name="password"
                required
```

```
          />
        </div>
      </div>
      <button
        type="submit"
        class="btn btn-primary w-100 custom-btn"
        onclick="registerUser(event)"
      >
        Register
      </button>
    </form>
    <p class="mt-3 mb-0 text-center text-muted">
      Already have an account? <a href="/login" class="text-primary">Login</a>
    </p>
  </div>
</div>
<script>
  async function registerUser(event) {
    event.preventDefault();

    const name = document.getElementById("name").value.trim();
    const email = document.getElementById("email").value.trim();
    const password = document.getElementById("password").value;

    const payload = { name, email, password };

    const isEmptyInput = Object.values(payload).some(
      (value) => value === "" || value === null
    );
```

```
if (isEmptyInput) {
  alert("Please fill in all fields.");
  return;
}

const isValidEmail = /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);

if (!isValidEmail) {
  alert("Please enter a valid email address.");
  return;
}

try {
  const response = await fetch("/api/register", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(payload),
  });

  const data = await response.json();

  if (response.ok) {
    alert("Registration successful! Welcome, " + data.name);
    window.location.href = "/login";
  } else {
    alert("Error: " + data.error);
  }
} catch (error) {
```

```
          console.error("Registration error:", error);
          alert("Something went wrong. Please try again later.");
        }
      }
    </script>
  </body>
</html>
```

Test.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Speed Typing Tester</title>
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
      rel="stylesheet"
    />
    <link
      rel="stylesheet"
      href="{{ url_for('static', filename='style.css') }}"
    />
  </head>
  <body class="bg-light">
    <div class="container py-5">
      <div class="d-flex justify-content-between align-items-center mb-4">
        <h1 class="mb-0">⌨ Speed Typing Tester</h1>
        <button class="btn btn-danger" onclick="logoutUser()">Logout</button>
```

```
   </div>

   <div class="card shadow-sm mb-4">
     <div class="card-body">
       <div id="sample-text" class="lead mb-4 text-muted" style="max-height: 150px;
overflow-y: auto;">
         Click Start to begin the test...
       </div>
       <textarea
         id="input-text"
         class="form-control mb-3"
         rows="5"
         disabled
       ></textarea>
       <div class="d-flex justify-content-between align-items-center">
         <div>
           <span class="badge bg-primary me-2"
             >Time: <span id="timer">60</span>s</span
           >
           <span class="badge bg-success me-2"
             >WPM: <span id="wpm">0</span></span
           >
           <span class="badge bg-info"
             >Accuracy: <span id="accuracy">0</span>%</span
           >
           <span class="badge bg-warning"
             >Mistakes: <span id="mistakes">0</span></span
           >
         </div>
         <div class="d-flex align-items-center">
           <label for="difficulty-select" class="me-2">Difficulty:</label>
```

```
        <select id="difficulty-select" class="form-select me-2" style="width: 120px;">
          <option value="easy">Easy</option>
          <option value="medium" selected>Medium</option>
          <option value="hard">Hard</option>
        </select>
        <label for="time-select" class="me-2">Test Duration:</label>
        <select id="time-select" class="form-select me-2" style="width: 120px;">
          <option value="60">1 min</option>
          <option value="300">5 min</option>
          <option value="600">10 min</option>
          <option value="900">15 min</option>
        </select>
        <button id="start-btn" class="btn btn-primary">Start Test</button>
      </div>
    </div>
  </div>
</div>

<div class="card shadow-sm">
  <div class="card-header">
    <h5 class="mb-0">Typing Score Summary</h5>
  </div>
  <div class="card-body">
    <div class="table-responsive">
      <table class="table table-hover">
        <thead>
          <tr>
            <th>WPM</th>
            <th>Accuracy</th>
            <th>Duration</th>
```

```
                <th>Date</th>
              </tr>
            </thead>
            <tbody id="scores-table"></tbody>
          </table>
        </div>
      </div>
    </div>
  </div>
  <script>
    function logoutUser() {
      localStorage.removeItem("user");
      window.location.href = "{{ url_for('login_page') }}";
    }
  </script>
  <script src="{{ url_for('static', filename='script.js') }}"></script>
</body>
</html>
```

Settings.py

```
import os
from pathlib import Path
from dotenv import load_dotenv

# Load environment variables
```

```
load_dotenv()

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(_file_).resolve().parent.parent

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = os.getenv("SECRET_KEY", "C3DF87348227358E39E3FEF7CE872")

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'corsheaders',
    'typing',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'typetest.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'typetest.wsgi.application'

# Database
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': os.getenv('MYSQL_DATABASE'),
        'USER': os.getenv('MYSQL_USER'),
        'PASSWORD': os.getenv('MYSQL_PASSWORD'),
        'HOST': os.getenv('MYSQL_HOST'),
        'PORT': '3306',
    }
}

# Password validation
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Custom user model
AUTH_USER_MODEL = 'typing.User'

# Internationalization
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
```

```
USE_I18N = True
USE_TZ = True

# Static files (CSS, JavaScript, Images)
STATIC_URL = 'static/'
STATICFILES_DIRS = [BASE_DIR / 'static']

# Default primary key field type
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

# CORS settings
CORS_ALLOW_ALL_ORIGINS = True  # Only for development
```

Urls.py

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('typing.urls')),
]
```

App.py

```
from flask import Flask, render_template, request, jsonify
from flask_sqlalchemy import SQLAlchemy
import random
```

```
from constants import SAMPLE_TEXTS

app = Flask(_name_)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

# Models
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(100), nullable=False)
    created_at = db.Column(db.DateTime, default=db.func.current_timestamp())
    scores = db.relationship('Score', backref='user', lazy=True)

class Score(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    wpm = db.Column(db.Integer, nullable=False)
    accuracy = db.Column(db.Float, nullable=False)
    mistakes = db.Column(db.Integer, nullable=False)
    duration = db.Column(db.Integer, nullable=False)  # Duration in seconds
    date = db.Column(db.DateTime, default=db.func.current_timestamp())

# Routes
@app.route('/')
def index():
    return render_template('index.html')
```

```
@app.route('/login')
def login_page():
    return render_template('login.html')

@app.route('/register')
def register_page():
    return render_template('register.html')

@app.route('/test')
def test_page():
    return render_template('test.html')

@app.route('/api/register', methods=['POST'])
def register_user():
    data = request.get_json()
    name = data.get('name')
    email = data.get('email')
    password = data.get('password')

    if not all([name, email, password]):
        return jsonify({'error': 'All fields are required'}), 400

    if User.query.filter_by(email=email).first():
        return jsonify({'error': 'Email already exists'}), 400

    user = User(name=name, email=email, password=password)
    db.session.add(user)
    db.session.commit()
    return jsonify({'id': user.id, 'name': user.name}), 201
```

```
@app.route('/api/login', methods=['POST'])
def login_user():
    data = request.get_json()
    email = data.get('email')
    password = data.get('password')

    user = User.query.filter_by(email=email, password=password).first()
    if user:
        return jsonify({'id': user.id, 'name': user.name}), 200
    return jsonify({'error': 'Invalid email or password'}), 400

@app.route('/api/get-text')
def get_text():
    difficulty = request.args.get('difficulty', 'medium').lower()  # Default to medium
    if difficulty not in SAMPLE_TEXTS:
        difficulty = 'medium'  # Fallback to medium if invalid
    text = random.choice(SAMPLE_TEXTS[difficulty])
    return jsonify({'text': text})

@app.route('/api/save-score', methods=['POST'])
def save_score():
    data = request.get_json()
    user_id = data.get('userId')
    wpm = data.get('wpm')
    accuracy = data.get('accuracy')
    mistakes = data.get('mistakes')
    duration = data.get('duration')

    user = User.query.get(user_id)
    if not user:
        return jsonify({'error': 'User not found'}), 400

    score = Score(user_id=user_id, wpm=wpm, accuracy=accuracy, mistakes=mistakes,
duration=duration)
    db.session.add(score)
    db.session.commit()
    return jsonify({'status': 'success'}), 201

@app.route('/api/scores/<int:user_id>')
def get_scores(user_id):
    user = User.query.get(user_id)
    if not user:
        return jsonify({'error': 'User not found'}), 400

    scores = Score.query.filter_by(user_id=user_id).order_by(Score.date.desc()).all()
    scores_data = [
        {
            'wpm': score.wpm,
            'accuracy': score.accuracy,
            'mistakes': score.mistakes,
            'duration': score.duration,
            'date': score.date.isoformat()
        }
        for score in scores
    ]
    return jsonify(scores_data)

if _name_ == '_main_':
    with app.app_context():
        db.create_all()
    app.run(debug=True)
```

manage.py

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys



def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'typetest.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed?"
        ) from exc
    execute_from_command_line(sys.argv)



if _name_ == '_main_':
    main()
```

requirements.txt

```
Flask==3.1.0
Flask_Cors==4.0.0
```

```
flask_sqlalchemy==3.1.1
python-dotenv==1.0.1
Werkzeug==3.1.3
mysqlclient==2.2.4
django==5.0.2
django-cors-headers==4.3.1
```

Error Handling:

1. **Database Connection Failure**
   - If the connection to MySQL fails, show: **"Database connection failed."**
   - Prevent further actions that depend on the database.

2. **User Input Validation**
   - If the user tries to start the test without logging in, show: **"Please login to start the test."**

3. **Empty Typing Input**
   - If the input field is submitted empty, alert: **"Typing area cannot be empty."**

4. **Result Calculation Errors**
   - If any error occurs during WPM or accuracy calculation, show: **"Error calculating results. Please retake the test."**

5. **Database Query Failures**
   - Wrap queries in try-except blocks.
   - If error occurs (e.g., insert or fetch fail), show: **"Something went wrong. Try again later."**

6. **Missing Score Data**
   - If the history is empty, show: **"No test history available yet."**
   - Avoid showing empty tables.

7. **Login Errors**
   - If wrong email/password, show: **"Invalid credentials. Please try again."**
   - If login service fails, show: **"Login service temporarily unavailable."**

8. **Signup Errors**
   - If email already exists, show: **"Email already registered."**
   - If insertion fails, show: **"Signup failed. Please try again."**

9. **Session Expiry / Unauthorized Access**
    - If a logged-out user accesses /test, redirect to login with: **"Please login first."**

10. **Typing Test Disruption**

- If user refreshes or exits during the test, warn: **"Test progress will be lost."**

11. **Result Download (Optional Feature)**

- If implemented, disable export button if no data is available.
- Show: **"No data available to download."**

12. **Unexpected Errors**

- Use generic handler to catch unknown errors: **"Something went wrong. Please try again later."**

Code Improvements:

Security and Best Practices

- **Password Security**: Ensure passwords are stored using secure hashing (already implemented using werkzeug.security).
- **SQL Injection Protection**: Continue using parameterized queries to prevent injection.
- **Session Security**: Use session cookie flags like SESSION_COOKIE_SECURE, SESSION_COOKIE_HTTPONLY for secure sessions.

2. Code Structure and Efficiency

- **Separate Files (Modularization)**:
  - Move database logic (save scores, fetch scores) to a separate models.py.
  - Move route definitions to a dedicated routes.py.
- **Error Handling**: Implement graceful error handling for common errors (404, 500) instead of console logs or alerts.

3. Front-End Improvements

- **Loading Indicator**: Add a spinner when the typing test is submitting or results are being calculated.
- **Responsive Layout**: Adjust margins/padding for better mobile experience (use Bootstrap grid more efficiently).
- **Clear Input Field**: Automatically clear typing area after test is completed.
- **Prevent Unwanted Actions**: Add confirmation before exiting during an active test.

4. Performance Optimization

- **Connection Pooling**: Use MySQL connection pooling (mysql.connector.pooling) to avoid reconnecting on each request.
- **Optimized Queries**: Reduce redundant queries; fetch all required data (user info + scores) in single optimized queries.

5. Feature Upgrades (Optional Enhancements)

- **Leaderboard**: Add global leaderboard based on WPM and accuracy.
- **Typing History Chart**: Show progress graph (using Chart.js or Matplotlib).
- **Dark/Light Mode**: Add a toggle switch for dark and light themes.
- **Admin Dashboard**: Manage sample texts and user results from an admin panel.

- **Custom Timer Option**: Let users choose typing test duration.
- **Sound Feedback**: Add typing sound effects for better user interaction.
- **Export Results**: Allow users to export their typing history as CSV or PDF.

Parameter Passing:

User → Frontend (test.html)

- The user types text into the <textarea> (id="input-text").
- When the user clicks the **"Start Test"** button, the test begins, and JavaScript handles live input tracking.
- Once the timer finishes, JavaScript sends the test result data to the backend using fetch() with **POST** method.

Data Sent to Backend:

```
json
CopyEdit
{
  "userId": 1,
  "wpm": 52,
  "accuracy": 95.3,
  "mistakes": 4,
  "duration": 60
}
```

2. Frontend (JS) → Backend (Flask @app.route('/api/save-score', methods=['POST']))

- JavaScript fetches the logged-in user's ID from localStorage.
- It calculates the WPM, accuracy, and mistakes in real time.
- After test ends, it sends the data using fetch() to the Flask route /api/save-score.

Backend receives JSON data with:

```
python
CopyEdit
data = request.get_json()
user_id = data['userId']
wpm = data['wpm']
accuracy = data['accuracy']
mistakes = data['mistakes']
duration = data['duration']
```

3. Backend → Database (MySQL)

- Flask receives the result data and stores it in the **Score Table** using SQLAlchemy.
- Score model includes fields like: user_id, wpm, accuracy, mistakes, duration, and date.

Example Entry in DB:

text
CopyEdit
```
User ID: 1
WPM: 52
Accuracy: 95.3%
Mistakes: 4
Duration: 60 seconds
Date: 2025-05-25
```

4. Backend → Frontend (Data Retrieval)

- When the user visits the test page again, the JS calls /api/scores/<user_id> to fetch the last 10 scores.
- Flask queries the database and sends the result back as JSON.

Example JSON Response:

json
CopyEdit
```
[
  {"wpm": 52, "accuracy": 95.3, "date": "2025-05-25"},
  {"wpm": 47, "accuracy": 90.1, "date": "2025-05-24"},
  ...
]
```

5. Frontend (JS) → Display Results in Table

- The JavaScript receives the data and updates the **history table** in the UI.

Example:

html
CopyEdit
```
<td>52</td>       <!-- WPM -->
<td>95.3%</td>    <!-- Accuracy -->
<td>25/05/2025</td> <!-- Date -->
```

✅ **Summary Chart**

Step	Who	Action
1	User	Types text and starts the typing test
2	JS Frontend	Calculates WPM, accuracy, and mistakes
3	JS → Flask	Sends data via POST to /api/save-score
4	Flask → DB	Stores test result in Score Table

Step	Who	Action
5	JS → Flask	Fetches test history from /api/scores/<id>
6	JS → Browser	Displays test results and history in a table

Validation Checks:

Typing Test Form Validation (Frontend & Backend)

- Ensure the typing input field is **not empty** before starting or submitting the test.
- **Prevent pasting** inside the typing area to avoid cheating (already handled).
- Limit typing input length to a **reasonable maximum** (e.g., 1000 characters).
- Use trim() in JavaScript to remove leading/trailing spaces.
- Avoid client-side script manipulation using basic DOM validation.
- Backend handles result saving only if valid data (wpm, accuracy, user id) is present.

2. Signup Form Validation

- Ensure **name, email, and password** fields are not empty.
- Validate email format using regular expressions (e.g., example@domain.com).
- Ensure password has at least **6 characters**.
- Check for **duplicate email** before inserting into the database (already done).
- Sanitize input fields to prevent special character issues.

3. Login Form Validation

- Check if **email and password** are filled before submission.
- Validate email format on the backend before querying.
- Show clear error messages if login fails (e.g., "**Invalid credentials**") – already implemented.
- Redirect to /login if session is not found during protected access.

4. Database Validation (MySQL)

- Set fields in the **user table** as NOT NULL.
- Enforce **UNIQUE** constraint on the email column.
- Validate test score inputs (WPM, accuracy, etc.) before insertion.
- Use parameterized queries with SQLAlchemy to prevent SQL Injection (already in place).

5. Session Validation

- Protect /test route to be accessible **only by logged-in users** (already done).
- If session or localStorage user is missing, redirect to /login.
- Clear session properly on logout to prevent unauthorized access.

6. Additional Suggestions

- Show friendly **flash messages** on login/signup pages for success/failure.
- If database connection fails, show message: **"Server error. Please try again later."**
- Disable test result download (if implemented) when no data is present.
- Log any unexpected errors and inform user with a general message: **"Something went wrong."**

Testing:

1. Unit Testing

- **Test Database Connection**
  - Check if MySQL connects successfully.
  - Handle connection failures gracefully with error messages.
- **Test Score Saving Function**
  - Verify score is saved with correct user ID, WPM, accuracy, mistakes, and duration.
  - Handle invalid or missing data.
- **Test Typing Logic (Frontend)**
  - Ensure WPM is calculated correctly (characters / 5 ÷ time).
  - Validate accuracy and mistake detection functions.
  - Check timer functionality and reset.
- **Test User Authentication**
  - Signup with valid data.
  - Signup with existing email – show error.
  - Login with correct/incorrect credentials – redirect or show error.
  - Logout – clear session/localStorage and redirect.

2. Integration Testing

- **Test Routes**
  - /register, /login, /test, /api/save-score, /api/scores/<id>
  - Ensure complete flow works: Register → Login → Take Test → Save Score → Logout.
- **Form Validation**
  - Test form submission with empty inputs – expect validation alerts.
  - Test submission with valid data – expect success response.
- **Typing Score Retrieval**
  - Load test history after login.
  - Validate that correct scores display per user.
- **Session Control**
  - Try accessing /test or saving scores without login – redirect to login.

3. UI Testing

- **Responsive Design**
  - Ensure UI works well on desktop, tablet, and mobile (Bootstrap grid).
  - Check font size, spacing, and alignment across screen sizes.
- **Navigation Bar & Footer**
  - Navbar shows logout button and correct redirection.
  - Footer displays author/project credits correctly on all pages.
- **Error & Success Messages**
  - Clear messages for login/signup errors (e.g., “Invalid credentials”, “Email already exists”).
  - Show messages for successful registration, logout, and test completion.

4. Security Testing

- **Session Handling**
  - Verify session/localStorage is cleared on logout.
  - Protect /test route from unauthorized access.
- **Input Validation**
  - Prevent special characters in text input.
  - Block pasting in the test area (already implemented).
- **Password Protection**
  - Ensure passwords are securely hashed using werkzeug.security.

5. Performance Testing

- **Load Testing**
  - Simulate multiple users submitting scores simultaneously.
  - Check if app remains responsive.
- **Database Query Time**
  - Measure fetch time from /api/scores/<user_id> for performance.
  - Optimize data retrieval and minimize server load.

6. Manual Testing Checklist

Test Case	**Expected Result**
Register with a new user	Account created and redirected to login
Register with existing email	Show “Email already registered” error
Login with correct credentials	Redirect to typing test page
Login with wrong credentials	Show “Invalid credentials”

Test Case	**Expected Result**
Start and finish typing test	WPM, accuracy, and mistakes are calculated
Submit empty test input	Show “Typing area cannot be empty”
View test history	Table displays user’s past scores
Try accessing /test without login	Redirected to login with warning
Logout from the app	Session/localStorage cleared, go to login

System Security:

- **Password Security**

  - Passwords are securely hashed using werkzeug.security before saving to the database.
  - During login, check_password_hash() is used to verify the password securely.

 Session Management

- Sessions expire after 30 minutes using PERMANENT_SESSION_LIFETIME.
- Secure session cookies are enabled (SESSION_COOKIE_SECURE, SESSION_COOKIE_HTTPONLY) to protect session data.

 SQL Injection Protection

- All database queries use **parameterized queries** (e.g., cursor.execute(query, (data,))).
- This prevents malicious SQL input from being executed.

 Cross-Site Scripting (XSS) Prevention

- Flask auto-escapes variables in templates using Jinja2.
- Inputs are sanitized to avoid harmful HTML or script injections.

 Cross-Site Request Forgery (CSRF) Protection

- Flask-WTF is used to insert CSRF tokens into all forms.
- CSRF tokens ensure that only valid and authenticated form submissions are accepted.

 Secure HTTP Headers

- Strict-Transport-Security header is added to enforce HTTPS usage.
- This prevents man-in-the-middle attacks and ensures secure data transfer.

 File Upload Security *(for future feature)*

- Only specific file types (like PDF, PNG) will be allowed.
- Uploaded files will be stored securely and outside the public directory.

 Database Security

- Database users have **restricted roles/permissions** (read/write only as needed).
- Regular backups are recommended to prevent data loss.

 Logging and Monitoring

- Important actions (e.g., login attempts) are logged using Python's logging module.
- Integration with tools like Sentry or CloudWatch is suggested for real-time monitoring.

 Rate Limiting

- Flask-Limiter can be used to restrict login attempts (e.g., 5 per minute).
- Helps prevent brute-force and spam attacks.

 Dependency Management

- Regular checks for outdated or vulnerable packages using pip-audit or safety.
- Ensures the latest security patches are applied.

Cost Estimation:

Hosting and Server

To host the Speed Typing Tester (developed using Flask), you'll need a web server or cloud provider like **Render, Heroku, AWS**, or **DigitalOcean**.

- **Estimated Cost:** $5 to $50/month (depending on traffic and compute usage)

2. Database (MySQL)

Since the application uses **MySQL** (possibly via phpMyAdmin/XAMPP or hosted on cloud), hosting will be needed if deployed online.

- **Estimated Cost:** $10 to $50/month (based on data size and access load)

3. API & Utility Services

- **Speech Recognition**: Your app uses the **Web Speech API**, which is free in most browsers for moderate use. For higher-level voice services, paid APIs may apply.
    - **Estimated Cost:** Free (for most use cases) or $10–$50/month for scalable APIs
- **PDF Export (Optional)**: If implemented using client-side libraries like html2pdf.js, it’s free.
    - **Estimated Cost:** $0

4. Frontend Design and Development

- **Bootstrap** and frontend frameworks are free.
- If you outsource design/custom development:
    - **Estimated Cost:** $500 to $2,000 (one-time cost depending on UI/UX complexity)

5. Security (HTTPS & SSL Certificates)

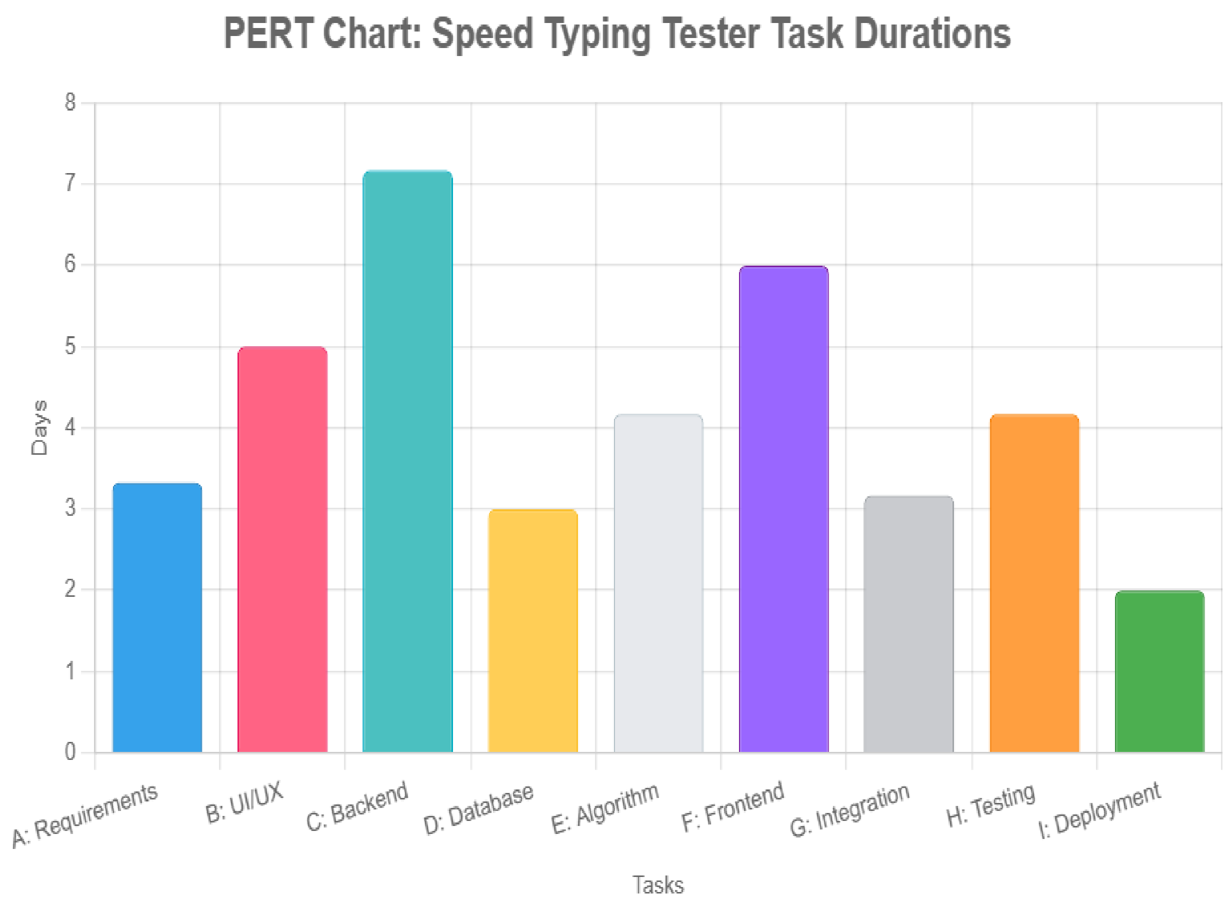
To run the website securely on HTTPS, you’ll need SSL.

- **Estimated Cost:**
    - Free (with Let’s Encrypt)
    - $50–$200/year (for premium SSL certificates)

6. Miscellaneous Costs

- **Domain Name**: For a custom site like speedtypingtest.com, a domain is required.
  - **Estimated Cost:** $10–$20/year
- **Maintenance and Bug Fixes**: Includes future updates, security patches, and server monitoring.
  - **Estimated Cost:** $50 to $200/month (depending on usage and dev support)

Pert Chart:


PERT Chart: Speed Typing Tester Task Durations
Days
8
7
6
5
4
3
2
1
0
A: Requirements
B: UI/UX
C: Backend
D: Database
E: Algorithm
F: Frontend
G: Integration
H: Testing
I: Deployment
Tasks

Future Scope:

Add More Typing Tests

Right now, the system offers basic typing tests with limited text samples. In the future, we can add:

- Different difficulty levels (easy, medium, hard)
- Tests for specific skills (e.g., coding, numbers, or punctuation)
- Fun challenges (e.g., quotes, stories, or programming code snippets)
- Custom tests based on user interests (e.g., medical terms, legal terms)

This will make the typing tester useful for a wider range of users, like students, coders, or professionals.

Build a Mobile App

Creating a mobile app for Android and iOS will make the typing tester easier to use on phones and tablets. We can also add an offline mode where users can practice basic tests without internet, using pre-loaded texts stored in the app.

Add Leaderboards and Social Features

To make the typing tester more fun, we can add:

- A leaderboard to show top typists (stored in MySQL database).
- Options to share results on social media (using JavaScript APIs).
- Multiplayer mode where users can compete with friends in real-time typing races (using Flask for live updates).

Improve Privacy and Security

To keep user data safe and follow rules like GDPR, we can:

- Use encryption to protect data stored in MySQL (via XAMPP/phpMyAdmin).
- Ask for user permission before saving any data, like test results.
- Add clear privacy policies to explain how data is used.

Work with Typing Experts

We can team up with typing coaches or educators to:

- Review and improve the text samples used in tests.
- Add expert tips for better typing (e.g., hand position or shortcuts).
- Ensure the system is accurate and helpful for learning.

Conclusion:

The **Speed Typing Tester** is a Flask-based web application developed to help users enhance their typing speed and accuracy through a practical and interactive testing environment. By analyzing real-time user input, the system calculates key performance metrics such as **Words Per Minute (WPM)**, **accuracy**, and **mistakes**, providing users with immediate feedback to track and improve their typing skills.

The platform features a secure **user authentication system** that enables personalized access and stores individual test histories in a structured MySQL database. Users can register, log in, take multiple typing tests, and review their past performance through an intuitive and clean interface.

With its lightweight architecture, responsive design, and real-time functionality, the application serves as an ideal tool for students, professionals, and anyone aiming to practice and enhance their typing ability. The system also integrates important validation features such as paste restrictions to maintain fairness and data integrity.

Overall, the **Speed Typing Tester** project showcases how **web development and backend technologies** can be combined to build an efficient, user-centric tool for skill assessment. It contributes to digital literacy by offering a reliable, secure, and accessible platform for continuous self-improvement in typing performance.

Bilbography:

1. **Flanagan, D. (2020). JavaScript: The Definitive Guide (7th Edition). O'Reilly Media.**
   This book explains JavaScript in detail, covering how to create interactive features like real-time typing tests and dynamic score displays for the frontend of the Speed Typing Tester.

2. **Duckett, J. (2014). HTML and CSS: Design and Build Websites. Wiley.**
   A beginner-friendly guide to designing attractive and user-friendly interfaces using HTML and CSS, perfect for making the typing tester's frontend look clean and engaging.

3. **Werkhoven, P. (2019). Flask Web Development with Python: Developing Web Applications with Python (2nd Edition). O'Reilly Media.**
   This book covers how to use Python Flask for backend development, including setting up routes and handling user data, which is useful for the Speed Typing Tester's backend.

4. **Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). Database System Concepts (7th Edition). McGraw-Hill Education.**
   A key resource for understanding MySQL and database management, helping with storing user scores, test results, and leaderboards in the Speed Typing Tester.

5. **Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.**
   This book provides insights into machine learning techniques that can be used to analyze typing patterns and suggest personalized tests, enhancing the system's smart features.

References:

1. **Flask Web Framework Documentation**
   Official Resource: flask.palletsprojects.com
   - Covers: Routing, templates, SQLAlchemy integration, and REST API development
   - Key Features: Session management, error handling, and deployment configurations
2. **Typing Speed Metrics & Algorithms**
   TypingClub Research: TypingClub Methodology
   - Formulas for WPM (Words Per Minute) and accuracy calculation
   - Techniques for real-time error detection and highlighting
3. **Database Management with SQLAlchemy**
   Official Docs: SQLAlchemy ORM Guide
   - MySQL database modeling for user profiles and test results
   - Query optimization and relationship management (1-to-many for User-Scores)
4. **Frontend Development for Typing Tests**
   MDN Web Docs: JavaScript Keyboard Events
   - Handling keypress events for typing input
   - UI best practices for timer displays and result visualization
5. **Deployment & Scaling**
   Heroku Flask Guide: Deploy Python Apps
   - Configuring Gunicorn webserver
   - Database setup on cloud platforms
   - Domain and SSL certification