Infosys Accessibility Platform System Architecture

Kenny Nguyen
Infosys iCETS Team
kennyhoang.trn@infosys.com

Abstract

This report presents the architecture of an accessible, AI-augmented coding platform built to support children with cognitive, visual, auditory, and motor impairments. By combining block-based programming, natural language input, and multimodal interaction, the platform reduces barriers to learning code. The system is modular and lightweight, with a React-based frontend, Flask backend, and swappable LLM layer. Designed for inclusivity and future extensibility, it serves as both an educational tool and a testbed for research into accessible AI systems, supporting enhancements such as voice input, eye tracking, and intelligent code assistance.

1 System Overview

The Infosys Accessibility Platform is a browser-based coding environment designed to support children with diverse impairments through block-based programming, natural language input, and adaptable interfaces.

An embedded AI assistant, powered by large language models, facilitates real-time code generation, personalized feedback, and dynamic learning paths. Users can describe intentions in plain language, receive context-aware hints, and engage with scaffolded challenges tailored to their needs.

The system supports modality adaptation—adjusting interface pacing, complexity, or input method—and is designed to scale toward future enhancements like eye tracking, voice control, and curriculum-linked personalization.

By targeting low-resource devices and emphasizing accessibility-first design, the platform functions as a flexible, research-driven testbed for inclusive educational technology.

2 Core Architecture

The system uses a modular client-server design. The frontend, built in React, embeds Blockly for visual code manipulation and Monaco Editor for optional text entry. Accessibility-first UI design includes keyboard navigation and progressive enhancements toward screen reader support.

The Flask backend serves RESTful endpoints for user management, prompt processing, and LLM communication. SQLite handles lightweight persistence, ideal for offline or low-latency deployment.

A swappable inference layer supports both local (Hugging Face Transformers) and remote (LangChain, OpenAI/Anthropic) LLM providers. HTTP-based communication ensures clean frontend-backend separation.

The system is divided into three primary layers:

- **Interaction Layer:** Manages UI input, feedback, and user actions.
- **Orchestration Layer:** Routes prompts, dispatches tasks, and handles model execution.
- **Persistence Layer:** Tracks user data, progress, and state.

This layered structure allows incremental evolution—new models, interfaces, or accessibility tools can be added with minimal architectural disruption.

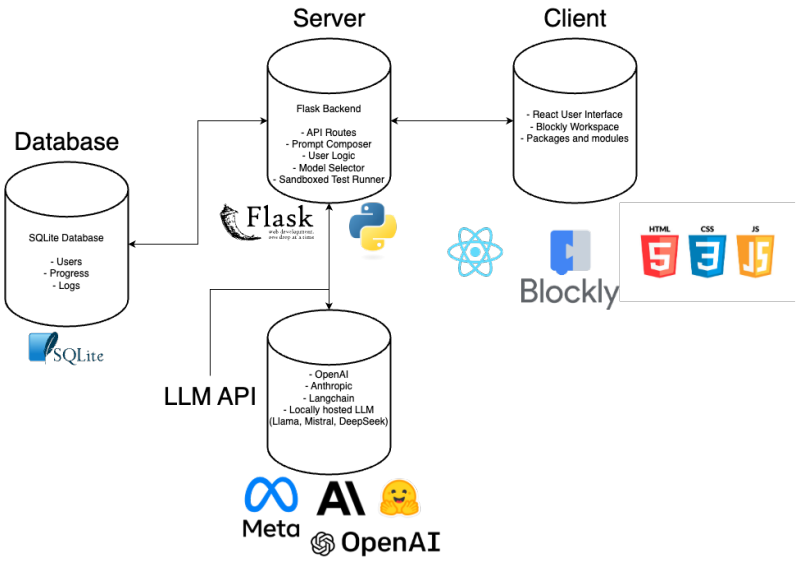

Figure 1: High-level system architecture.

3 LLM Integration Strategy

The backend exposes a natural language endpoint that routes user prompts to large language models for code explanation and contextual assistance. The integration layer supports both local (via Hugging Face Transformers) and cloud-based (via LangChain) inference. This modular setup allows for model swappability without frontend dependency.

4 Key Components

Table 1: Core System Components

Component	Description
React UI	Accessible, child-friendly interface
Blockly Workspace	Visual code editor for block-based programming
Flask Backend	Handles API calls and user logic
SQLite	Local database for user progress
LLM API Module	AI assistant using Transformers and LangChain

5 Design Priorities

- **Accessibility-Aware** — Structured for compatibility with future screen readers and keyboard navigation.
- **Lightweight and Deployable** — Minimal stack
- **Modular and Extensible** — Clean separation of UI, logic, and backend; supports plugin-like features.
- **Child-Friendly UX** — Interfaces are simple, visual, and intuitive.
- **AI-Ready** — Backend designed to interface with LLMs without impacting core UX.

6 Accessibility Design Features

Our platform is built with a forward-looking approach to accessibility, aiming to support users with cognitive, visual, and motor impairments. While the current implementation emphasizes visual clarity and intuitive block-based programming, the system architecture supports progressive integration of multimodal accessibility technologies.

6.1 Planned Enhancements

- **Eye Tracking Support** — Future versions will integrate with browser-based eye tracking libraries to allow gaze-based input for users with motor impairments.
- **Voice Input and Commands** — Planned integration of the `Web Speech API` or react speech recognition will enable users to control the interface and generate code via spoken instructions.
- **Text-to-Speech Feedback** — Features like reading blocks aloud or reading UI prompts will be supported using browser-native TTS engines.
- **Cognitive Load Reduction** — UI components will introduce progressive disclosure patterns (e.g., via custom step-by-step flows) to reduce overwhelm and guide users gradually.
- **Real-Time Captioning** — Future support for automatic speech recognition (ASR) tools (e.g., OpenAI Whisper, Google Cloud Speech-to-Text) to transcribe instructions for hearing-impaired users.

This architecture ensures that as new interaction modes become available or as user needs evolve, they can be incrementally added without disrupting the core interface or logic.