

Workflow: Accessibility Simulator for Web Apps

1. System Architecture Overview

- **Frontend (Simulation & UI Layer)**
 - Technologies: HTML5, CSS3, JavaScript (React optional)
 - APIs: Canvas API, Web Speech API, CSS Filters
 - Responsibilities:
 - Render input website inside sandboxed iframe.
 - Apply impairment simulation layers (visual, motor, cognitive).
 - Display live preview of impaired experience.
 - Provide developer control panel for selecting impairment type and severity.
 - **Backend (Optional, for reports & storage)**
 - Technologies: Node.js (Express) or Python (Flask/Django)
 - Responsibilities:
 - Generate and export reports (PDF/CSV/JSON).
 - Store test sessions and results.
 - Optional API integration with WCAG validation tools (e.g., axe-core).
 - **Data/Reports Layer**
 - Export formats: PDF, CSV, JSON
 - Libraries: jsPDF / Puppeteer (Node.js) / ReportLab (Python)
 - Includes: Test summary, impairment settings used, detected WCAG compliance issues, developer recommendations.
-

2. Workflow Steps

Step 1: Input Layer

- Developer provides:
 - **Target URL** (website to test).
 - **Impairment Type** (visual, motor, cognitive).
 - **Severity Level** (mild, moderate, severe).
 - Technical Handling:
 - Target URL is embedded inside **sandboxed iframe**.
 - Control panel UI lets user select impairments (checkboxes, sliders).
-

Step 2: Simulation Layer

2.1 Visual Impairments

- **Color Blindness:**
 - Use SVG/CSS filters to transform color space.
 - Example: protanopia (red-green), tritanopia (blue-yellow).
 - **Low Vision / Blurriness:**
 - Canvas API filters (`blur(px)`, `contrast()`, `brightness()`).
 - Zoom simulation via CSS `transform: scale()`.
 - **Grayscale / Monochrome:**
 - `filter: grayscale(100%)`.
 - **Screen Reader Navigation (Optional):**
 - Web Speech API to simulate reading page structure aloud.
 - Keyboard-only navigation enforced (disable mouse events).
-

2.2 Motor Impairments

- **Delayed Input:**
 - Add artificial lag (`setTimeout`) before event handlers fire.
 - **Shaky Cursor Simulation:**
 - Cursor position randomized within small pixel radius.
 - **Restricted Keyboard Input:**
 - Disable certain keys (e.g., Tab, Enter).
 - Require multiple presses to register an action.
-

2.3 Cognitive Impairments

- **Delayed Content Reveal:**
 - Load DOM elements with timeouts.
 - Simulate “slowed processing.”
 - **Random Distractions:**
 - Pop-ups or background noise (via Web Audio API).
 - **Memory Overload Simulation:**
 - Flashing/highlighted elements to mimic attention difficulties.
-

Step 3: Feedback & Analytics Layer

- **Live Feedback Panel:**
 - Display detected WCAG issues (via axe-core integration).
 - Show accessibility hints (e.g., “Low contrast detected”).
- **Analytics Visualization:**

- Libraries: Chart.js / D3.js
 - Display summary charts: % compliance, issue categories, user effort.
-

Step 4: Reporting Layer

- **Exportable Reports:**
 - Export impairment settings + test results + WCAG warnings.
 - Formats: PDF (via jsPDF/Puppeteer), CSV, JSON.
 - Include:
 - Target URL
 - Impairment Type & Severity
 - Key accessibility issues
 - Suggested developer fixes
-

3. Dependencies & Requirements

- **Frontend**
 - HTML5, CSS3, JavaScript
 - React (optional, for modular UI)
 - Canvas API, Web Speech API, Web Audio API
 - CSS filters
 - **Backend (Optional)**
 - Node.js (Express) OR Python (Flask/Django)
 - Puppeteer/Playwright for automated report generation
 - axe-core API for WCAG compliance checks
 - **Libraries & Tools**
 - jsPDF / Puppeteer → PDF report generation
 - Chart.js / D3.js → Analytics visualization
 - axe-core → Automated WCAG testing
-

4. Deployment & Hosting

- **Development Environment:**
 - Local testing with Node.js server or Flask
 - GitHub for version control
- **Deployment:**
 - Netlify / Vercel for frontend hosting
 - Heroku / Railway for backend (if needed)
- **Browser Compatibility:**

- Must support modern APIs (Chrome, Firefox, Edge).