Digit Symbol Substitution Test (DSST) Web App – Programming Specification

Overview

The goal is to build a **clinic-grade** web application for the Digit Symbol Substitution Test (DSST), a widely used neuropsychological assessment of processing speed and executive function (Modernizing the Digit Symbol Substitution Test: a case for digital testing) (Digit Symbol Substitution Test: The Case for Sensitivity Over Specificity in Neuropsychological Testing - PMC). This specification outlines the requirements and design of a responsive web-based DSST suitable for clinical trials, emphasizing accuracy, timing precision, security, and compliance. The application will support multiple user roles (test-takers and administrators), secure data handling, and rigorous timing and scoring in line with standard DSST protocols. Key features include:

- **Accurate DSST Administration:** Timed test (typically 90 seconds) with precise measurement of responses and automatic scoring (Digit Symbol Substitution Test: The Case for Sensitivity Over Specificity in Neuropsychological Testing - PMC).
- **User-Friendly Interface:** Responsive UI showing symbol-digit pairs and input fields optimized for both desktop and mobile, replicating the paper test in digital form.
- **Multi-User Management:** Secure login for each user; each session yields a new DSST attempt with timestamped results stored for analysis.
- **Admin Dashboard:** Administrative interface to manage users and sessions, and export results (scores, timing metrics, etc.) in CSV format for research or clinical record-keeping.
- **Security & Compliance:** Data encryption, user privacy (HIPAA/GDPR compliance), audit trails (timestamping actions), and integration recommendations for authentication and data storage in regulated environments.

Background and Clinical Standards for DSST

The DSST has been used for over a century to measure associative learning and processing speed (Digit Symbol Substitution Test: The Case for Sensitivity Over Specificity in Neuropsychological Testing - PMC). In a typical DSST, the participant is given a reference key pairing each digit (e.g. 1–9) with a unique symbol. Within a fixed time limit (usually **90 to 120 seconds**), the individual must match as many symbols with their corresponding digits as possible (Digit Symbol Substitution Test: The Case for Sensitivity Over Specificity in Neuropsychological Testing - PMC). The score is the number of correct substitutions completed in the allotted time, and this simple metric has proven highly sensitive to cognitive dysfunction in various conditions (Digit Symbol Substitution Test: The Case for Sensitivity Over Specificity in Neuropsychological Testing - PMC) (Digit Symbol Substitution Test: The Case for Sensitivity Over Specificity in Neuropsychological Testing - PMC). Modern versions (e.g. WAIS-III) often use **120 seconds and enlarged stimuli** to improve accessibility (such as for left-handed participants) (Digit Symbol Substitution Test: The Case for Sensitivity Over Specificity in Neuropsychological Testing - PMC). Our application will default to the widely used **90-second** administration, with an option to adjust the time limit to match specific clinical trial protocols.

Digital Implementation in Clinical Trials: As remote and computer-based assessments become common, computerized DSSTs have been developed that maintain equivalence with the traditional paper test (JMIR Mental Health - An App-Based Digit Symbol Substitution Test for Assessment of Cognitive Deficits in Adults With Major Depressive Disorder: Evaluation Study) (Validation of a Smartphone-based Digit Symbol Substitution Task in Participants With Major Depression - Cambridge Cognition). Studies show a strong correlation between digital and paper DSST scores, supporting the validity of online administration (JMIR Mental Health - An App-Based Digit Symbol Substitution Test for Assessment of Cognitive Deficits in Adults With Major Depressive Disorder: Evaluation Study). For example, an app-based DSST correlated significantly with the paper version (r≈0.7–0.8) in clinical populations (JMIR Mental Health - An App-Based Digit Symbol Substitution Test for Assessment of Cognitive Deficits in Adults With Major Depressive Disorder: Evaluation Study). However, research also indicates that **interface design can affect performance** – participants tend to complete fewer items on a small-screen digital version compared to paper (on average ~20 fewer correct items on smartphones) (Validation of a Smartphone-based Digit Symbol Substitution Task in Participants With Major Depression - Cambridge Cognition). This underscores the need for an optimized UI to minimize any slowdowns due to the device or input method. Our design will incorporate best practices from recent digital DSST implementations to ensure results are as comparable as possible to the pen-and-paper standard.

Clinical Standards & Protocols: The DSST must be administered under standardized conditions. That means consistent instructions, a practice trial if needed, a strict timer, and uniform scoring rules. In clinical-grade use, timing must be **precise** and the environment controlled — any technical delay could bias the score on this rapid task.

Therefore, the app will use high-resolution timers and will be tested for timing accuracy (to the millisecond). Scoring will be automated to eliminate administrator scoring errors and variability (Digit Symbol Substitution Test (DSST): New Digital Version is Sensitive and Low-burden Option for Global Alzheimer's Disease Trials), providing immediate and objective results. Automatic scoring is a major advantage of digital administration, as it removes human error and the need for a trained neuropsychologist to manually count correct symbols (Digit Symbol Substitution Test (DSST): New Digital Version is Sensitive and Low-burden Option for Global Alzheimer's Disease Trials). The application will adhere to Good Clinical Practice (GCP) guidelines and, if used in regulated trials, facilitate compliance with FDA 21 CFR Part 11 for electronic records (e.g. audit trails, secure user authentication, timestamped data entries).

Timing Precision and Scoring Methodology

Timing Requirements: The DSST web app must enforce a strict countdown (typically 90 seconds, configurable up to 120 seconds). Once the user clicks “Start,” the timer begins and input is accepted only until time expiration. The countdown will be displayed to the user (e.g. a visible timer or progress bar) for awareness. **Timing precision** is critical – the system should use the browser’s high-resolution timing (e.g. performance.now() in JavaScript) to track elapsed time, ensuring the test length is exact to within a few milliseconds. This precision aligns with clinical standards, as the DSST’s sensitivity demands consistency in test duration (Digit Symbol Substitution Test: The Case for Sensitivity Over Specificity in Neuropsychological Testing - PMC). The app must handle timing locally (client-side) to avoid network latency issues, but also record server-side timestamps for audit. When time runs out, the interface should immediately prevent further input and automatically trigger the end-of-test routine (stopping at exactly 90.00s, for example).

Scoring Rules: The scoring is straightforward: **the number of correct symbol-digit matches completed within the time limit constitutes the score** (Digit Symbol Substitution Test: The Case for Sensitivity Over Specificity in Neuropsychological Testing - PMC). Each correct entry adds 1 point to the score. There is no penalty for wrong answers in the traditional DSST; typically, the examinee is simply expected to skip or continue after errors, and only correct responses count. Our system will implement this by either ignoring incorrect attempts or tracking them separately without subtracting from the score. At test end, the system calculates:

- **Total Items Attempted:** count of responses given (this can be used to compute errors = attempted minus correct).
- **Total Correct:** the primary DSST score (Digit Symbol Substitution Test: The Case for Sensitivity Over Specificity in Neuropsychological Testing - PMC).

- **Errors:** number of incorrect responses (for additional analysis, though not used in the primary score) (CognitionKit Digit Symbol Substitution (DSST) - Cambridge Cognition).
- **Completion Rate:** (optional) percentage of attempted items that were correct.

The raw score (total correct) is the key outcome and can be compared against normative data or baseline values. The app will present this score to authorized viewers (e.g. the administrator, or the user if desired) immediately after the test. Each session's result will be timestamped and stored. If needed, the scoring algorithm will accommodate different time limits (e.g. scaling to 120-second administration) simply by allowing more time for more items; the scoring mechanism remains the same.

Response Timing Metrics: In a clinical research setting, additional timing data can be valuable. The app will optionally record **response latencies** – the time taken to answer each item. This can help distinguish processing speed from motor speed and identify patterns across the 90-second interval (CognitionKit Digit Symbol Substitution (DSST) - Cambridge Cognition). For example, one could analyze whether the participant slowed down or maintained pace, which has been explored in digital analyses of DSST performance (Digital Technology Differentiates Graphomotor and Information ...). Our design will capture per-item timestamps (time when each symbol is presented and when the user responds) to allow calculation of inter-response times. These detailed metrics (if enabled) will be stored with the session data for later export, but they won't interfere with the user experience (the test taker just focuses on inputting answers). All timing events (start time, each response time, end time) will be recorded using a synchronized clock to ensure accuracy for research purposes.

User Interface and Usability Best Practices

The DSST web app's UI will be designed for **clarity, speed, and minimal error**, drawing on best practices from validated digital versions of the test. The interface consists of a **symbol-digit reference key**, the **test items (prompts)**, and an **input mechanism** for responses. It must remain intuitive even for users with little computer experience, and responsive to different device sizes. Key UI features and considerations include:

- **Constantly Visible Legend:** The digit-symbol key (mapping of 1–9 to their corresponding symbols) will be displayed at the top of the test screen at all times (Modernizing the Digit Symbol Substitution Test: a case for digital testing). This imitates the paper test where the key is printed on the page's header. The legend will use clear, high-contrast symbols and digits, large enough to be easily read on small screens. This reference allows users to quickly lookup the association as they proceed. Even in a reduced mobile view, the legend should be scrollable or condensed but accessible.

- **Sequential Item Presentation:** To optimize for digital interaction, the test will present **one symbol at a time** for the user to respond to. This design is supported by modern implementations (e.g., Cognition Kit and TestMyBrain) which show symbols one-by-one and have users input the matching digit (Comparing a Computerized Digit Symbol Test to a Pen-and-Paper Classic | Schizophrenia Bulletin Open | Oxford Academic). Sequential presentation focuses the user's attention and simplifies input, which is crucial on smaller devices. It also lets the software capture each response time individually. In our app, a large symbol (the prompt) will appear in the center of the screen, indicating "Which number corresponds to this symbol?". The user then enters the digit via the input controls. Immediately after an answer is submitted, the next symbol is shown, until the timer expires.
- **Efficient Input Controls:** The input method will be tailored to the device:
    - On **desktop/laptop**, the user can simply type the number key (1–9) on the keyboard corresponding to the symbol. We will also provide an on-screen number pad (1–9 buttons) that can be clicked, for users who prefer mouse input.
    - On **touch devices (tablet/phone)**, a large, touch-friendly keypad of digits 1–9 will be displayed below the symbol prompt. This avoids the need to bring up a mobile keyboard and ensures quick taps. Each button will be sized for easy tapping to minimize motor delays or input errors – digital versions have noted that tapping buttons (or dragging choices) can reduce variance caused by handwriting speed in older adults (Modernizing the Digit Symbol Substitution Test: a case for digital testing).
    - The interface will register the input as soon as a digit is entered or tapped. There's no "submit" button to press for each item; the act of entering the digit is the response, immediately triggering the next symbol display to maximize speed. (If a user presses the wrong key, they will simply continue to the next item; the error is recorded but the test doesn't pause.)
- **Layout and Responsiveness:** The design will use a responsive layout (CSS flexbox/grid) to adapt to various screen sizes:
    - **Desktop/Tablet:** likely a side-by-side or top-bottom arrangement with the legend at the top, the current symbol prompt large and centered, and the input area (number pad) below it. There may also be an

    indicator of progress (e.g., how many items completed, though this is optional and typically not shown on paper tests).
  - **Smartphone:** a full-screen view with perhaps the legend in a scrollable carousel at the top (or a toggle button to show/hide it if space is very tight), the symbol prompt occupying most of the screen, and the numeric keypad at the bottom. We will ensure even on a small phone the symbol and buttons are not too small. Scrolling during the test should not be necessary (to prevent losing time), so the layout will fit within the viewport for common device sizes.
  - We will test the interface in portrait and landscape orientations. Especially for tablet use, landscape might allow the legend and prompt side by side for a more similar feel to the paper grid. For phones, portrait might be more natural for one-handed operation.
- **Visibility & Accessibility:** All text and symbol elements will have high contrast. The symbols chosen will be simple geometric shapes or abstract figures that are easily distinguishable (as per standard DSST shapes). We avoid any culturally specific or complex images to keep the test neutral (Digit Symbol Substitution Test (DSST): New Digital Version is Sensitive and Low-burden Option for Global Alzheimer's Disease Trials). Font sizes and element spacing will be set to avoid mis-taps. Notably, earlier clinical use found that left-handed people had to reposition their hand to see the key on paper (Digit Symbol Substitution Test: The Case for Sensitivity Over Specificity in Neuropsychological Testing - PMC); our digital design avoids this by placing the key at the top and input below – no hand will cover the legend during input. The app will also provide auditory or visual alerts if needed (for example, a gentle "time's up" notification when the test ends). We will not use color alone to convey information (to be friendly to color-blind users), and all interactive controls will be labeled clearly (with ARIA labels for screen readers, if accessibility for impaired users is in scope).
- **Instruction and Practice Module:** To ensure users understand the task, especially in unsupervised remote settings, the app will include a brief instruction screen and an **optional practice trial** before the timed test. Users will see an example of the symbol-digit mapping and perhaps complete a few sample substitutions with guidance. This echoes the "learn" module approach used in some digital DSST versions to train users on the task rules (Digit Symbol Substitution Test (DSST): New Digital Version is Sensitive and Low-burden Option for Global Alzheimer's Disease Trials). The practice will not count toward the score and can be repeated if the user wants to be sure of what to do. Once the user is comfortable, they can proceed to the real test. This feature is important for clinical trials where we cannot assume a proctor

is present to explain the test; it standardizes the instruction process and has been shown to allow reliable unsupervised testing (Digit Symbol Substitution Test (DSST): New Digital Version is Sensitive and Low-burden Option for Global Alzheimer's Disease Trials).

(JMIR Mental Health - An App-Based Digit Symbol Substitution Test for Assessment of Cognitive Deficits in Adults With Major Depressive Disorder: Evaluation Study) *Example of a mobile DSST interface, with the digit-symbol legend at the top and a single symbol prompt (triangle) shown for the user to match. A numeric response box or buttons allow the user to enter the corresponding digit. The design ensures quick access to the legend and a clear focal point on the current symbol.*

- **Feedback to User:** During the test, we typically will **not** give correctness feedback on each item (to mirror the standard administration, where examinees do not know if they made an error in real-time). The interface simply moves to the next item. After the test, a completion screen will inform the user that the session is complete. For a test-taker, we might display a simple message like “You completed X items. Thank you.” Optionally, it could show their score (X correct out of Y attempted) if appropriate in the context. In a blinded clinical trial, the user might not see their score; only the administrators would. This can be configurable.

Overall, the UI is designed to let the user focus on the core task: quickly matching symbols to digits. By reducing extraneous interactions (like needing to navigate, scroll, or confirm inputs) and keeping the reference info handy, we adhere to best practices and published implementations that found mobile DSSTs user-friendly and preferable to paper (JMIR Mental Health - An App-Based Digit Symbol Substitution Test for Assessment of Cognitive Deficits in Adults With Major Depressive Disorder: Evaluation Study). Usability testing will be conducted to fine-tune this interface (e.g., ensure the legend is readable and the buttons are the right size).

Data Collection and Storage Requirements

Every DSST test session should be rigorously logged and stored to meet clinical data standards. **Data integrity** and **completeness** are crucial — no result should be lost or ambiguously recorded. The system will use a secure database to store structured records of each session, including:

- **User Identification:** A reference to the user who took the test (user ID, not a plaintext name to protect identity). In a clinical trial, this might be a subject code rather than a real name to maintain pseudonymization of health data.

- **Session Metadata:** Each test session will have a unique Session ID, a timestamp for when the test started, and when it ended. The timestamps will be recorded in UTC for consistency (with conversion to local time only for display purposes). This provides an audit trail of when assessments occurred (important for studies that correlate test results with dosing or other events). We will also log the device or browser info (user agent) for each session, as this can be relevant if analyzing performance differences (e.g., if a session was done on a phone vs a PC, given known effects on speed (Validation of a Smartphone-based Digit Symbol Substitution Task in Participants With Major Depression - Cambridge Cognition)).
- **Results Data:** For each session, the primary stored outcomes will be **Score (total correct)**, **Number of Items Attempted**, and **Number of Errors**. These allow basic analysis of performance. Additionally, if enabled, the app will store an array or table of each **Item Response** (with the symbol identifier, the user's answer, whether it was correct, and the response time in milliseconds). This granular data might be toggled on for research mode but can be omitted in a simple clinical use to save storage. All data will be structured and clearly labeled to facilitate export and analysis. For example, a JSON or separate table for item-level data can be used.
- **Session Status:** Normally sessions end when time is up. We will mark if a session was aborted or ended early (e.g., if a user quit mid-test or a technical failure occurred) so that data quality filters can exclude such sessions. Partially completed tests might still save what was done, but flagged as incomplete.
- **Timestamping:** As part of compliance, every record will have creation and modification timestamps. The system will not allow editing of test results (they are read-only once captured, to preserve data integrity), but any administrative changes (like correcting a user's demographic info or merging accounts) will be logged with timestamps and user IDs performing the change.

The database should be **transactionally safe** – when a test session finishes, saving the results should be an atomic operation. If the network connection is lost at the end, the app should retry or cache the result locally to submit later, to avoid data loss (an important consideration for remote trials). We may implement a local fallback that, in case of temporary disconnection, stores the results in the browser and uploads when back online.

Data Retention and Export: All user session data will be retained in the database until an authorized deletion or export. The admin interface will allow exporting data in **CSV format**. Each CSV export will typically include one row per test session with columns such as: User ID, Session ID, Date/Time, Duration, Score, Attempts, Errors, etc. (Optionally, a separate CSV or a more complex format could be used for item-level responses if needed, but by default we focus on summary data in CSV). The export function will ensure proper formatting (escaping commas, etc.) and will include a header row with labels. The admin can download these CSV files to analyze in Excel or import into statistical software. Export functionality will be designed to comply with security – e.g., only admins can export, and an export action might be logged (who exported and when) to maintain an audit trail of data access.

Clinical Data Standards: In a clinical trial context, data might need to be **de-identified** or coded. The system can be configured such that user accounts use a study ID (with no directly identifying info stored alongside test scores). If integration with electronic data capture (EDC) systems or Electronic Health Records (EHR) is required, we will design the data schema to be compatible (for instance, we could map fields to standards like CDISC or send data via an API in HL7 FHIR format if needed in future). Initially, a straightforward relational schema will be used, but extensibility for compliance with **CDASH/SDTM** (clinical data standards) will be considered (e.g., capturing the test name, version, and result as would be needed for submission).

Data Security in Storage: See the Security section for details, but in brief: all stored data will reside in a secure database with encryption at rest. Access to the data is restricted to authorized accounts (admin users via the app, or database admins via secure credentials). We will implement regular backups of the database to prevent loss (with backups also stored securely). In compliance with regulations, data will be stored for the duration required by the study or policy (e.g., in clinical trials, often at least 5-15 years).

Security, Privacy, and Compliance

Given the sensitive nature of health-related cognitive data, the web app will be built with **security and privacy by design**. We will address both **HIPAA** requirements (for U.S. medical data protection) and **GDPR** (for EU personal data protection), ensuring the system can be used in clinical environments worldwide. Key security and compliance features include:

- **User Authentication & Access Control:** All users (participants and admins) must log in with a unique username and password. Passwords will be stored hashed (e.g. using bcrypt or Argon2) to prevent retrieval even if the database is compromised. We will enforce strong password rules and optional two-factor authentication for admin accounts. The system ensures users can only access their own data: a test-taker logging in will only trigger their own test session and cannot see other users' results, whereas an admin can access broader data (HIPAA Compliance for App & Web-based Digital Health Platforms. What is HIPAA? | by Nicholas Cole | Medium). Role-based access control is in place: admin pages and APIs require admin privileges. All API endpoints will validate the user's session token on each request to prevent unauthorized access. Sessions will expire after a period of inactivity (e.g., log out users after 10 minutes of no activity by default, and require login again) (HIPAA Compliance for App & Web-based Digital Health Platforms. What is HIPAA? | by Nicholas Cole | Medium). This prevents someone from accidentally leaving a logged-in session accessible on a shared computer.

- **Secure Transmission:** The app and its backend will require **TLS/SSL encryption** for all network communication. This means using HTTPS for the web interface and encrypting any API calls. Data entered by the user (including login credentials and test results) will never be sent in plaintext. This meets HIPAA's transmission security requirement (HIPAA Compliance for App & Web-based Digital Health Platforms. What is HIPAA? | by Nicholas Cole | Medium). We will use HSTS to enforce SSL and configure the server for strong cipher suites.
- **Encryption at Rest:** The database or storage where results are kept will be encrypted. If using a cloud service, we'll enable transparent disk encryption. For an added layer, especially for backup files or CSV exports, we can encrypt those files. Any sensitive fields that might identify a person (if we store names or emails for accounts) could be further encrypted or at least hashed. By default, we will minimize storing personally identifiable information – using codes for users. This aligns with GDPR's recommendation of pseudonymization (GDPR Compliance for Your Applications: A Comprehensive Guide - Security Compass). For example, rather than "John Doe", a user might be stored as "User123" with a separate secure mapping to their identity held by the study coordinator.
- **Privacy and GDPR Compliance:** If the app is used in the EU or for EU citizens, we will implement GDPR compliance measures. This includes a clear privacy notice explaining what data is collected and why, obtaining **user consent** when required (e.g., a consent checkbox during account registration for using their data in the study) (GDPR Compliance for Your Applications: A Comprehensive Guide - Security Compass). Users (or study participants) will have rights to their data: though in a clinical trial context some of these rights can be limited, our system design allows data export per user and data deletion. An admin could fulfill a GDPR "right to be forgotten" request by deleting a user's personal info and results if appropriate (with proper authorization, since in a regulated trial one might instead archive rather than

fully delete data). The system will also allow correction of data if needed (e.g., if a user's demographic info was recorded wrong, an admin can update it, and this action is logged). We practice **data minimization** – only data needed for the DSST assessment and research objectives is collected. No unnecessary tracking or personal info will be gathered (for instance, we won't ask for address or national ID unless absolutely required).

- **Audit Trails:** For clinical-grade compliance (GCP, 21 CFR Part 11), the system will maintain an audit log of key events. Every user login, logout, test start, test completion, data export, and admin data change can be recorded in an audit log with timestamp and user ID. These logs can be stored in an append-only format (to prevent tampering) or in a secure logging service. Part 11 compliance also involves ensuring records are not modifiable without trace; our approach of not allowing test result edits (only new records) inherently supports that. If an admin must adjust or annotate a record, the change should be saved as a new entry or field (with attribution and time). An **electronic signature** (in terms of Part 11) is essentially the combination of the user's credentials and a timestamp in our system; every action can be linked to a logged-in user. We will document the system's compliance features so that it can pass sponsor or regulatory audits (e.g., by demonstrating how we handle authentication, authorization, audit logs, data backup, and recovery).
- **Session Security:** We will use secure cookies or tokens for session management. The session token will be HTTP-only and encrypted/signed to prevent forgery. On the server, we'll implement rate-limiting and monitoring for suspicious logins (to mitigate brute force attacks). For additional security, especially under HIPAA, we might integrate a third-party identity provider that offers compliance (Auth0, Okta, or hospital SSO) to handle login securely. The app will also have a proper logout mechanism. If a user closes the browser, the session token can be invalidated after a short period.
- **Hosting and Compliance:** Deployment will be on a secure server or cloud that meets healthcare compliance standards. For example, using AWS or Azure with a Business Associate Agreement (BAA) for HIPAA. We will ensure data is stored in appropriate regions (EU data in EU data centers, etc.) (HIPAA Compliance for App & Web-based Digital Health Platforms. What is HIPAA? | by Nicholas Cole | Medium). Regular security updates and patches will be applied to the software (especially for any third-party libraries). We will also conduct vulnerability scanning and penetration testing before production. Any third-party integrations (analytics, error reporting) will be vetted for compliance – for instance, we will avoid sending any PHI to external

analytics. Usage analytics, if needed, can be done in-house or via a HIPAA-compliant service.

- **Backup and Disaster Recovery:** Data will be backed up on a schedule to encrypted storage. Only authorized personnel can restore or access backups. In case of a server failure, the system should be able to recover data without loss (backups, possibly redundant servers). This is part of compliance (ensuring availability and integrity of health data).

By embedding security from the ground up, the DSST web app will protect participant confidentiality and meet legal standards for handling cognitive assessment data. In summary, **only authorized users can access the data, all data transmissions are encrypted, sensitive data is safeguarded at rest, and full audit trails with timestamps are kept** (HIPAA Compliance for App & Web-based Digital Health Platforms. What is HIPAA? | by Nicholas Cole | Medium) (HIPAA Compliance for App & Web-based Digital Health Platforms. What is HIPAA? | by Nicholas Cole | Medium). These measures align with both HIPAA's technical safeguards and GDPR's privacy principles, ensuring the tool is acceptable for clinical use globally.

System Architecture and Technology Stack

To implement the above features, we propose a **web application architecture** with a separated front-end and back-end, using modern, well-supported technologies. The system will be designed for scalability (able to handle multiple users simultaneously) and maintainability (clean code structure to allow updates as standards evolve). Below is an outline of the architecture and recommended tech stack:

Front-End (Client-side):

- We will use a **single-page application (SPA)** approach for a smooth, responsive user experience. A framework like **React.js** is recommended for building the UI components and managing state (the timer countdown, the current symbol, user input, etc.). React is suitable due to its modular structure and large ecosystem. Alternatively, **Angular** or **Vue.js** could be used – any of these can meet our needs, but React with a UI library (like Material-UI or Ant Design) provides many ready-made responsive components for forms, buttons, and layouts.
- The front-end will handle the presentation of the DSST and capture user interactions. It will also enforce some logic like the countdown and sequence of items (to ensure responsiveness even if the network is slow – i.e., the test should not rely on constant server calls to get next item; the sequence can be generated or fetched once at start).

- **Responsive Design:** We'll utilize CSS media queries or a responsive framework (Bootstrap or Material Design) to quickly adapt the layout for different devices. The front-end will be tested on common browsers (Chrome, Firefox, Safari, Edge) and devices (Windows, Mac, iPad/Android tablet, iPhone/Android phone). We will use standard HTML5 and CSS3, ensuring compliance with Web Content Accessibility Guidelines (WCAG) as much as possible (for example, proper labels and focus management, so the app is usable to those with certain disabilities, if needed).
- **State Management:** For React, we might use a state management library (like Redux or the context API) to handle global state such as the current user session and test data being collected, especially for admin dashboards that list many records. For the test-taking view, local component state or hooks will suffice to track the item responses in real-time, then send to the server at end.
- **Timing Implementation:** Use window.requestAnimationFrame or setInterval for the countdown display updating (with performance.now() to correct any drift). The exact end time will be determined by comparing to the start time plus 90,000ms to ensure precision. The front-end can lock input when time is up, but it will also get a final confirmation from server when saving results, to ensure no extra input sneaked in.
- **Input Handling:** We will capture keypress events for number keys and onClick for on-screen buttons. Debounce or disable multiple rapid inputs (to prevent any chance of double-counting if someone presses a key right as time ends, etc.). The UI will give focus to the number input area automatically for keyboard users.

Back-End (Server-side):

- The backend will expose a set of **RESTful API endpoints** (or GraphQL if preferred, but REST is straightforward here) to handle: user authentication (login/logout), retrieving the DSST test configuration (symbol set, etc.), saving test results, and admin operations (user management, fetching results, triggering CSV export).
- A high-level choice is between a **Node.js/Express** stack or a **Python/Django** stack (or others like Java Spring, Ruby on Rails, etc.). Any of these can be made secure and compliant, but each has strengths:
    - **Node.js + Express:** Suitable for real-time interaction, lightweight JSON handling, and can easily integrate with front-end tooling. There are libraries like Passport.js for authentication, bcrypt for hashing passwords, and CSV generation libraries (e.g., json2csv or csv-writer) for export. Node can also use frameworks like NestJS to structure the app in a more enterprise way (with TypeScript).

    - **Python + Django:** Django offers an all-in-one solution with an ORM, an admin interface, and good security practices by default. It might accelerate development especially for the admin portal – Django admin could list users and sessions out-of-the-box. CSV export could be done via a management command or admin action. Python also has scientific libraries if any advanced analysis is needed server-side.
    - **Java + Spring Boot:** A heavier option that could be considered if integrating into a larger enterprise environment. It offers robust security frameworks (Spring Security) and would be highly scalable. But development might be slower compared to Node or Django for this scope.

For this project, **Node.js** with Express is recommended for its agility and the fact that the team can use the same language (JavaScript/TypeScript) on front and back-end. We will ensure to structure the code with separation of concerns (routes, controllers, services, models) to keep it maintainable.

- **Database:** Use a **relational database** such as **PostgreSQL** or **MySQL** to store user accounts and test results. A relational model fits well because we have clear entities (users, sessions, possibly response records). PostgreSQL is a strong choice given its reliability and JSON support (we could store detailed response arrays in a JSON column if not normalizing into a separate table). It also has features for encryption and robust backup tools. We'll design the schema (detailed in Data Models section) and use an ORM for the chosen backend (e.g., Sequelize or TypeORM for Node, or Django ORM for Python) to interact with the database safely. The ORM can help with input validation and preventing SQL injection by design.
- **Server Security:** The server will include middleware for authentication (e.g., JWT verification if we use token auth, or session middleware if using server sessions). It will also sanitize and validate all inputs (for example, ensuring someone can't submit a malformed result or attempt to access others' data via an API parameter). Frameworks like Express have security middlewares (helmet for setting secure HTTP headers, csurf for CSRF protection if needed, etc.). We will employ those best practices.
- **Session Management:** If using traditional sessions, use secure, http-only cookies with a session ID that maps to a server-side session store (like Redis or database). Alternatively, use JWT tokens that are sent in the Authorization header for API calls. In either case, implement refresh token or re-login after expiry to balance security and usability.
- **Symbol Set Configuration:** The set of symbol-digit pairs used in DSST is standard (typically digits 1-9 each assigned a distinct symbol). We will define

this in the database or config. The symbols could be stored as image files or Unicode characters. We might include them as SVGs for clarity. The back-end can provide the mapping to the front-end on request (or the front-end can have it pre-defined if static). If the study ever wants a modified DSST (like the "DSST-meds" version with pills and days (Digit Symbol Substitution Test (DSST): New Digital Version is Sensitive and Low-burden Option for Global Alzheimer's Disease Trials)), the app could support multiple mappings configured in the database, but for now one standard mapping suffices.

- **Generating Test Items:** The DSST essentially has a large set of symbol positions to fill (on paper, a sheet might have 100+ blanks, though few finish all). In our sequential design, we can simulate an infinite sequence that cycles through random symbols. Implementation: we can pre-generate a random sequence of symbol prompts for the duration of the test (e.g., 100 symbols long, which is definitely enough for 90 seconds). This can be done on the client or server. To ensure consistency, we might generate on the server (so it's recorded exactly which symbols were shown in which order, for audit). Alternatively, a simple algorithm on the client (with a fixed random seed for reproducibility if needed) can generate symbol IDs on the fly. Either approach is fine as long as the result is unpredictable and evenly distributed among the 9 symbols. We'll avoid any bias (each of the 9 symbols should appear roughly equally often in a long test). The server can send a "test packet" which includes the mapping and a long random sequence of symbols to present, when the user begins. This way, everything is determined upfront and the client just walks through it – this avoids client needing to call the server for each next symbol (which would risk network delays affecting the test flow).
- **Scalability:** The system as described is not extremely heavy – even if many users take the test simultaneously, the front-end does most of the heavy lifting (timing and UI). The back-end mainly handles login and saving results, which are relatively low-frequency operations per user. A single server with a decent database can handle dozens of concurrent users easily. If needed, we can scale horizontally by having multiple app servers behind a load balancer (stateless JWT auth would work well for that) and a single database. We should also ensure the database is indexed on keys like user ID for efficient querying (especially for the admin listing results).
- **Integrations & Third-Party Services:** The design allows integration of certain services for added functionality:
    - **Authentication Service:** Instead of custom-building auth, we could integrate with an OAuth provider (e.g., Auth0) to manage logins securely. Auth0, for example, can store users, handle password resets, and even provide logs of logins (helpful for audit) – and it can be

configured to be HIPAA compliant. This might save development time and improve security, but it introduces an external dependency. The decision will depend on the project scope and budget for external services.

  - **Analytics:** If we want to track usage metrics (like how many users completed the test, average scores, etc.), we can include an analytics library. Google Analytics is common, but since it may collect user agent/IP which could be PHI in context, a self-hosted or HIPAA-compliant analytics solution (or simply our own admin dashboard stats) might be preferred. This is optional.
  - **Error Monitoring:** Use a tool like Sentry (which can be self-hosted) to capture any runtime errors in the client or server. This helps maintain reliability. Ensure it's configured to scrub any sensitive data from error reports.
  - **Data Storage/EDC Integration:** Optionally, the app could push data to a central clinical database or electronic data capture (EDC) system. For example, if a sponsor uses REDCap or Medidata Rave, our back-end could have an integration module that sends the test results there via API. However, direct CSV export might suffice for many cases, where a data manager will import the CSV into the analysis pipeline.

Features and Functional Modules

1. User Registration & Login

User Model & Registration: User accounts will be created with attributes like username (or email), password, and role. In a clinical trial scenario, an admin might pre-create accounts for each participant (to avoid handling registration emails etc.). We will provide an **admin-controlled user creation** function: the admin can add a new user by providing a username (which could be a unique code or email) and a temporary password. The user can then log in and possibly be forced to reset password on first login (for security). Self-registration (public sign-up) is not typical for a controlled trial setting, but the system could allow it if needed (with email verification, etc.). That is optional and would be disabled by default to keep the system closed.

Login Process: A login page will collect username and password. On submission, the credentials are sent securely to the server (over HTTPS). The server verifies the password (hash comparison) and if valid, creates a session (server session or issues a JWT). The client is then redirected to the appropriate interface:

- If the user is role “participant” (test-taker), navigate to the DSST test start page.
- If role “administrator”, navigate to the Admin Dashboard.

We will implement protections like account lockout after too many failed attempts (e.g., lock for 5 minutes after 5 failed logins) to prevent brute-force guessing. Also, ensure any password reset mechanism is secure (if using email, or alternatively allow admin to manually reset passwords).

2. DSST Test Session Flow (Participant UI)

This is the core module where users take the test.

Start Session: When a user (non-admin) logs in, the application will immediately present them with the option to begin a new DSST. If each login is meant to correspond to one test session (as hinted by “each login initiates a fresh DSST session”), we could design it so that upon successful login, the user is taken directly into the instructions/practice and then the test without additional clicks. Alternatively, we show a welcome screen with a “Begin Test” button – this might be better if we want to allow the user to read instructions first. In either case, once the user proceeds, the system creates a Session record in the database (with start time and linking to the user).

Instructions/Practice: Before the timer starts, the user sees instructions. They can optionally try a practice question or view an example mapping. This screen will have a “Start Test” button that the user clicks when ready, which will trigger the timed test to start. The practice mode can be implemented by letting the user perform a few symbol-digit matches with no timer and giving feedback (“Correct”/“Incorrect”) to ensure they understand. The app should ensure the user cannot accidentally skip the instructions without at least scrolling through (maybe require them to check “I understand” or something to ensure compliance).

During Test: Once the test starts:

- The countdown timer becomes visible and starts decrementing.
- The first symbol prompt is shown (center of screen, large).
- The user inputs the digit via keyboard or tapping. As soon as input is received, if the time is still >0, the next symbol replaces the previous one. This continues in a loop. We will handle input and rendering such that it feels instantaneous (no page reloads – just dynamic content change). Possibly preloading the next symbol can ensure no flicker.
- If the user does not answer one item, they can’t explicitly skip it (since it’s one at a time). However, if they choose not to answer, the clock will simply run –

this would hurt their score due to lost time, so users are incentivized to respond (even guess). We might not implement a “pass” button, as it’s not standard in DSST (on paper they might just leave it blank and move on, but here moving on requires an input. Guessing is typically fine). We could allow a "skip" input that moves to next without giving an answer, but that functionally is the same as giving a wrong answer (since no point gained, time used). For simplicity, we will not add a separate skip control.

- The app will continuously record each response. For each item, we capture the symbol ID, the user’s response, and the timestamp (relative or absolute). This can be stored in memory on the client (an array of responses) and later sent to server, or sent one by one via AJAX. To reduce server load, we will send the bulk at the end, unless there’s concern about losing data if the user closes the window mid-test. Given the short duration, it’s reasonable to send all at once when finished. If mid-test data safety is a concern, we could send incremental updates (like every 10 responses or every 15 seconds via background AJAX) to at least partially save progress.
- The UI will prevent any outside distractions: we can use fullscreen API to fill the screen (optional), and disable certain keys (maybe ignore F1, etc., to avoid accidental triggers). If the user tries to leave or refresh (on web, if they press back or refresh), we will prompt: “Are you sure you want to exit the test? Your progress will be lost.” – to avoid accidental aborts.

End of Test: The moment the timer hits zero, the test ends:

- The UI will stop presenting new symbols and freeze the input controls. If the user was in the middle of typing a digit (unlikely since input is instantaneous), that last input will be accepted only if completed before time elapsed – otherwise it’s discarded.
- A “Time’s up” message might flash, and then we proceed to a summary or thank-you screen.
- The client will send the session data to the server (if not already sent incrementally). This includes the end timestamp, and all responses or at least the summary counts. The server will compute the final score (though the client could too, we trust server for official scoring). The server updates the Session record with the end time, computes score = count(correct responses) and errors = count(incorrect), etc., and saves those. It might also store the raw response list if we choose to in the database.
- The user is then shown a **completion screen**. This page will confirm the test is completed and may display results like “You got 30 symbols correct!”. This can be configured; in some trials, they may not want the patient to see their score to prevent any emotional reaction or gaming. We can allow an option to

hide the score from the participant and just say “Thank you, your session is complete.” In any case, at this point the user can log out or just close the app. We could provide a “Log out” button here for convenience (especially if someone else might use the same device next).

Multiple Sessions: If the design allows a user to do multiple sessions (e.g., repeat the test daily), we would after completion either log them out automatically or bring them back to a dashboard where they could potentially start another session or see a history of their sessions. The prompt “each login initiates a fresh DSST session” suggests maybe each user will do it one time per login. We can interpret it that after finishing, if they log in again later, they can do it again (creating a new Session record). So we will allow multiple sessions per user account. The app should handle if a user somehow hasn’t finished a session and tries to start another (though normally one would finish within 90 sec or not at all). For safety, we might prevent starting a new session if one is active.

3. Administrative Interface

The admin interface is accessible to users with the “admin” role after logging in. It will be a web dashboard with multiple sections: User Management, Session Results, and Data Export. The design will prioritize simplicity and security (only available to admins, clearly separated from the test UI to avoid confusion).

Admin Dashboard Homepage: This could show some high-level stats (e.g., total number of users, total sessions completed, date of most recent test) as a quick overview. It will also have navigation links or tabs to the main functions.

User Management:

- A page listing all user accounts in the system (with pagination if many). For each user: display username, role, and perhaps last login time or number of sessions completed.
- Admin can create a new user (form to enter username, temp password, role). On creation, the system emails the user a link or the admin notes the credentials to give to the participant. If email invites are desired, integrate an SMTP or email service (with caution around PHI if emailing).
- Admin can edit a user – typically just to reset password or change role. We won’t allow admin to see a user’s password (only set a new one). Also possibly allow deactivate/delete user. Deletion might be disallowed if their data must be kept (instead could mark inactive). We will require confirmation for any deletion.

- This section will ensure the admin cannot inadvertently view test results here unless needed; results are in their own section. However, linking user and their results can be done (e.g., clicking a user could filter the sessions list to only that user's sessions).

Session Results Browsing:

- A page showing a table of all test sessions. Columns could include: Session ID, User ID (or username), Score, Errors, Date/Time, Duration. The admin can sort or filter this table (filter by user, date range, or score range). This helps quickly find a particular session or see overall performance.
- Clicking on a session could open a detailed view: showing all the stored details, including maybe the sequence of responses (for in-depth analysis or troubleshooting, e.g., "the user started strong then slowed down…"). This detail view is optional; minimally, the admin might just rely on exporting data for analysis rather than viewing each in the web interface.
- We will incorporate search functionality (by user or date). For privacy, if usernames were actual names, the admin interface must be secured to only authorized staff, which we have covered with auth. We could also display only user codes.

CSV Export:

- An interface (or just a button) for exporting data. Possibly under the Sessions page, an "Export" button that takes current filter into account. For example, the admin could filter to last month and then export, or just export all data.
- When export is triggered, the server will gather the relevant data and generate a CSV file. This might be done on the fly, or, if data is large, the server could perform asynchronously and then provide a download link. For most cases, the data volume (scores of a cognitive test) is small, so on-the-fly is fine.
- The CSV file will be offered for download to the admin's browser. The content will include a header row with column names like "UserID, SessionID, StartTime, EndTime, Score, Attempts, Errors, …". If item-level detail is requested, we might output a separate CSV or a combined format (but that can get complex with multiple rows per session). Typically, for clinical data, each session's summary is one row; item-level data might be analyzed via separate scripts if needed.
- We will ensure the CSV respects any locale or formatting needs (likely use ISO date times, use '.' as decimal separator, etc., to be standard). Also, no sensitive personal identifiers in the CSV unless explicitly needed, to avoid

accidental disclosure if the file is shared. Ideally, user IDs in the CSV are study codes.

- Security: Only admins can trigger exports. We may log the event “Admin X exported data on date Y” in an audit log. The file generation happens server-side to ensure data isn’t pulled from client-side (which could be manipulated). Also, consider limiting export frequency or size to prevent performance issues (though unlikely an issue unless thousands of records).

Admin Authentication and Safety: The admin pages will double-check the user’s role on each request. We might even implement IP restrictions or 2FA for admin logins if this is a highly sensitive environment (optional). Also, we ensure that if an admin leaves their station, the session times out and data isn’t left open.

4. Integration & Compliance Features

This isn’t a separate UI module, but rather background features already discussed that will be implemented:

- **Logging & Audit:** All key events (logins, test start/stop, data changes) will be logged to a secure log. This could be simply a database table “audit_log” with columns (timestamp, user, action, details). For example: “2025-03-01 10:00:00, admin1, CREATED_USER, user=john_doe”. Or “2025-03-01 10:05:00, user123, TEST_COMPLETED, score=30, errors=2”. These logs can be reviewed if needed. If the system is part of a clinical trial submission, these logs help show compliance (though they might not be routinely exported).
- **Consent tracking:** If needed, the app can include a consent form where the user must agree before proceeding (common in digital health apps). We can store a record of consent given (timestamp, version of consent form text). This ensures GDPR compliance (proof of user consent for data use) (GDPR Compliance for Your Applications: A Comprehensive Guide - Security Compass). In a trial, usually consent is handled offline, but if this is a standalone tool, including it is prudent.
- **Audit mode for data changes:** If any test data needed correction (which is rare since it’s automatically scored, but imagine a scenario where a software bug caused an error in scoring that needs retroactive fix), the system should allow adding an *addendum* rather than silent change. An admin could flag a session with a note rather than editing the original. This way the original stays intact (21 CFR Part 11 principle of not altering original data without trace).

Data Models and Structures

We define the core data entities and their attributes. These can be translated into database tables (for SQL) or collections (for NoSQL) as needed. Below are the key models:

- **User:** Represents a person with login credentials (could be a participant or an admin).
    - user_id (primary key, e.g. UUID or integer)
    - username (string, unique) – could be an email or an assigned code.
    - password_hash (string) – hashed password.
    - role (string or enum) – e.g., “participant” or “admin”. (We can also use a boolean like is_admin, but role is more extensible.)
    - created_at (datetime) – account creation timestamp.
    - last_login (datetime) – last login time (for monitoring inactivity).
    - (Optional) email (string) – if using emails. Or other fields like first_name, last_name if needed for admin reference, but not required for functionality.
    - (Optional) status (string) – e.g., “active”, “inactive” for disabling accounts.
- **DSSTSession:** Represents one test attempt by a user.
    - session_id (primary key, UUID or int).
    - user_id (foreign key to User) – identifies who took it.
    - start_time (datetime).
    - end_time (datetime).
    - duration (integer or float) – duration in seconds (should equal time limit if completed). Could be derived from times, but stored for convenience.
    - score (integer) – number of correct responses.
    - attempted (integer) – number of items attempted (or total responses given).
    - errors (integer) – number of incorrect responses.
    - details (JSON or text, optional) – detailed log of the session. If storing per-item data, this can hold an array of objects like [{symbol: "◆", correct_digit: 4, answer_given: 4, latency_ms: 1500}, {...}]. Alternatively, we have a separate table Response (see below).
    - device (string, optional) – device or platform info (e.g., “Windows/Chrome” or “iPhone iOS15”), if we choose to record it for analysis.
    - completed (boolean) – true if completed normally, false if not (in case of abort or timeout issues). Normally true for all since even timeout is a normal completion. False might indicate a technical error.
- **Response (optional):** If we choose to store each response in a separate table (normalized form instead of JSON). This table could be very large if

many sessions, but each session max ~100 responses, which is fine. Storing separately allows SQL queries on response times etc. Fields:

  - response_id (pk)
  - session_id (fk to DSSTSession)
  - item_index (int) – the order of the item in that session (0-based or 1-based).
  - symbol_shown (string or int) – an identifier for the symbol (could be a code like 1-9 corresponding to which symbol from the key).
  - correct_digit (int) – the correct answer for that symbol (though this can be inferred if we know symbol->digit mapping, but storing it avoids needing to join with symbol reference).
  - answer_given (int) – the user's input.
  - is_correct (boolean) – whether the answer_given equals correct_digit.
  - response_time (int) – time in milliseconds from appearance to answer. (We might also store the absolute timestamp if needed, but relative within test is enough and more analysis-friendly.)

However, since the primary outcomes are summary scores, we might not need this table unless detailed analysis through SQL is required. For portability, storing details as JSON inside DSSTSession might be simpler (one less table).

- **Symbol (optional static reference):** A reference table for symbol-digit mapping. For example: 9 entries, each with digit 1-9 and maybe symbol_code or file name for the symbol image. This might not be necessary if the mapping is hardcoded in the app. But having it in the database allows flexibility (like swapping out symbol sets for different studies). It also can be linked in responses if needed. For now, it's optional.
- **AuditLog (optional):** If we implement an audit trail in the DB:
  - log_id (pk)
  - timestamp (datetime)
  - user_id (who performed the action, nullable if system)
  - action (string) – e.g., "LOGIN", "LOGOUT", "CREATE_USER", "TAKE_TEST", "EXPORT_DATA" etc.
  - details (text) – free text or JSON describing the action (could include IP address, target user for admin actions, etc.).

The relationships are: User 1–* DSSTSession, DSSTSession – Response (if implemented). Cascade deletions appropriately (if a user is deleted, maybe keep their sessions but anonymize? Usually, you wouldn't delete a user if their data is needed, you'd deactivate instead).

We will enforce data validation at the model level: e.g., score cannot exceed attempted, attempted cannot exceed some reasonable maximum (like 150 for 2 minutes, etc.), errors = attempted - score. The app will ensure consistency, but as a safeguard, the back-end can double-check these invariants when saving a session.

Tools, Frameworks, and Libraries Recommendations

To achieve a robust, compliant implementation, we recommend the following tools and libraries in the development stack, chosen for their stability and support in building secure health applications:

- **Frontend:**
  - **React** with **create-react-app** or Next.js for easy setup. Use **TypeScript** for type safety (helps prevent bugs).
  - Component library: **Material-UI (MUI)** for readily styled components (forms, buttons, layout grid) which are accessible and responsive. Material-UI also has theming which we can adapt to a clean medical-style theme (company branding or neutral).
  - State management: React's Context or Redux for global state like user session.
  - Timing and utility: Use the **browser DOM API** (e.g., performance.now()) for timers; no special library needed, but ensure polyfill for older browsers if needed (most modern browsers support these).
  - Testing: Use Jest and React Testing Library to write unit tests for components (e.g., ensure the timer stops at 0, ensure input logic works). This is important for verification of correctness.
- **Backend:**
  - **Node.js** (LTS version) with **Express**. Use **Express middleware** such as Helmet (sets security headers), morgan (logging), body-parser (JSON parsing).
  - **Passport.js** or **JSON Web Tokens (JWT)** for authentication. For example, we can implement login to return a JWT that the front-end stores (in memory or secure cookie) for subsequent requests. Passport has strategies for local auth (username/password) and can integrate with OAuth if needed later.
  - **bcrypt** library (bcryptjs or bcrypt) for hashing passwords.
  - **Joi** or **Yup** for request data validation (ensuring, for instance, the format of data sent from client is correct).

    - **sequelize** or **TypeORM** for database ORM in Node, which will handle our models and migrations. These support Postgres/MySQL and can enforce schemas.
    - **csv-writer** or **fast-csv** for generating CSV files easily from JSON data when exporting. This saves time formatting and ensures proper escaping.
    - If using Python Django instead: Django's built-in auth system (handles hashing and user sessions), Django Rest Framework for APIs, and standard Python CSV library for export. Django also has built-in admin UI we could use to list users and sessions (with customization), potentially speeding up admin feature development.
- **Database:**
    - **PostgreSQL** as primary data store. Use SSL connections to the DB. We can enable the **pgcrypto** extension if we want to encrypt certain fields at the DB level.
    - Alternatively, **MySQL** or **MariaDB** would also work. For a simpler start or smaller scale, even **SQLite** could be used in development (not for production with multiple users though).
    - Use **Knex** if we want query building without a full ORM, but ORM is easier for model definitions.
- **Deployment:**
    - Host on a **HIPAA-compliant cloud**. For example, **Heroku** has a Private Spaces offering with HIPAA compliance, **AWS** or **Azure** have HIPAA-eligible services. We will ensure to sign a BAA if needed.
    - Use **Docker** to containerize the app for consistency across environments and easier cloud deployment. This can also help in scaling (orchestrating multiple containers).
    - Set up **automated backups** of the database (e.g., daily dumps stored securely). Use infrastructure-as-code (Terraform scripts or Docker Compose) to document the deployment setup (makes compliance reviews easier when you can show exactly how the environment is configured).
- **Compliance & Testing Tools:**
    - Utilize static code analysis (linters, SonarQube) to catch any security issues in code.
    - Use an automated testing framework for security (like ZAP or custom scripts) to test the web app for common vulnerabilities (XSS, SQL injection, CSRF). Since we're dealing with sensitive data, a security audit is recommended before production.

- ○ Logging: Consider using a centralized log management (ELK stack or a service like Papertrail) to store logs from server (and optionally client events) securely. Ensure logs do not contain PHI (like don't log actual symbols or answers with user identifiers in plain text).

By leveraging these tools and libraries, the development team can ensure that the DSST web app is not only functional and user-friendly but also meets the stringent requirements of clinical trials and data protection regulations. We choose popular frameworks with large community support, which helps in long-term maintenance and finding solutions to any issues (for instance, React and Express are widely used, and Django if chosen has decades of use in secure web apps). Additionally, these frameworks have been used in medical applications before, indicating they can be configured for compliance (e.g., numerous healthcare startups use React/Node or Django, complying with HIPAA by following best practices).

Conclusion

This specification provides a comprehensive blueprint for implementing a **responsive, secure, and clinically compliant DSST web application**. By adhering to established clinical protocols (timed administration, standard scoring (Digit Symbol Substitution Test: The Case for Sensitivity Over Specificity in Neuropsychological Testing - PMC)) and integrating modern digital best practices (intuitive UI, automatic scoring (Digit Symbol Substitution Test (DSST): New Digital Version is Sensitive and Low-burden Option for Global Alzheimer's Disease Trials), and detailed timing analysis (CognitionKit Digit Symbol Substitution (DSST) - Cambridge Cognition)), the application will be capable of delivering DSST assessments in settings ranging from clinical trials to remote patient monitoring. The design accounts for multi-user management and data needs of researchers, with robust data capture and export capabilities. Furthermore, the emphasis on security (encryption, access control, audit trails) and compliance (HIPAA, GDPR, 21 CFR Part 11) ensures that the tool can be deployed in real-world healthcare and research environments responsibly (HIPAA Compliance for App & Web-based Digital Health Platforms. What is HIPAA? | by Nicholas Cole | Medium) (GDPR Compliance for Your Applications: A Comprehensive Guide - Security Compass).

By following this specification in a development environment like Cursor.ai, a development team can generate and refine code modules for each part of the system – from front-end components for the test interface to back-end endpoints for session handling – ultimately creating a reliable DSST web application that meets both technical and regulatory standards. The result will be a valuable digital neuropsychological test platform combining the proven utility of the DSST with the convenience and precision of modern web technology.

Sources: The above design is informed by current literature on digital DSST implementations and cognitive testing standards. For instance, digital variants of DSST have been validated to correlate well with traditional methods (JMIR Mental Health - An App-Based Digit Symbol Substitution Test for Assessment of Cognitive Deficits in Adults With Major Depressive Disorder: Evaluation Study), and design considerations such as limiting motor demands and ensuring cultural neutrality of symbols have been highlighted by researchers (Modernizing the Digit Symbol Substitution Test: a case for digital testing) (Digit Symbol Substitution Test (DSST): New Digital Version is Sensitive and Low-burden Option for Global Alzheimer's Disease Trials). The importance of timing accuracy and automatic scoring in computerized cognitive tests is consistently noted (Digit Symbol Substitution Test: The Case for Sensitivity Over Specificity in Neuropsychological Testing - PMC) (Digit Symbol Substitution Test (DSST): New Digital Version is Sensitive and Low-burden Option for Global Alzheimer's Disease Trials). Security and privacy recommendations draw from established healthcare IT guidelines (encryption, authorized access, audit logs) (HIPAA Compliance for App & Web-based Digital Health Platforms. What is HIPAA? | by Nicholas Cole | Medium) (HIPAA Compliance for App & Web-based Digital Health Platforms. What is HIPAA? | by Nicholas Cole | Medium) and data protection principles (GDPR Compliance for Your Applications: A Comprehensive Guide - Security Compass). This ensures the proposed application aligns with both cognitive assessment best practices and the high-level requirements of clinical data management.

—--------