# **Web Interface Specification for Security Assessment Results**

## **1. Introduction**

This document provides a detailed specification for a web-based interface designed to display, explore, and manage security assessment results. The primary goal of this interface is to provide different user roles with the necessary tools and information to understand their organization's security posture, prioritize vulnerabilities, and manage remediation efforts effectively.

The target audience for this specification includes software developers, quality assurance engineers, UX/UI designers, product managers, and security stakeholders involved in the development, testing, and deployment of this web interface. This document outlines user journeys, layout designs, component specifications, interaction patterns, accessibility requirements, performance considerations, and frontend architecture guidelines necessary for implementation.

## **2. User Journey Mapping**

Understanding how different users will interact with the system is crucial for designing an effective interface. We define three key personas and map their typical journeys.1

### **2.1 Security Analyst Persona & Journey**

* **Persona:** Responsible for identifying, assessing, prioritizing, and tracking vulnerabilities across the organization's assets. Needs a comprehensive overview, deep-dive capabilities, and efficient workflows for managing the vulnerability lifecycle.2
* **Goals:** Quickly understand the current security posture, identify critical risks, investigate specific findings, assign remediation tasks, track progress, and generate reports for stakeholders.
* **Typical Journey:**
  1. **Login & Dashboard Review:** Logs in and lands on the main dashboard. Reviews KPIs (e.g., total open critical vulnerabilities, mean time to remediate (MTTR), scan coverage).4 Examines summary charts (findings by severity, status). Checks recent scan activity.
  2. **Identify High-Priority Issues:** Uses dashboard summaries or navigates to the Findings Exploration interface. Filters findings by 'Critical' or 'High' severity, 'Exploitable' status (potentially using EPSS scores > threshold 5), or specific CISA KEV status.6 Sorts by severity or risk score.
  3. **Investigate Specific Finding:** Clicks on a high-priority finding in the list/table. Navigates to the Finding Detail View. Reviews vulnerability description, affected asset details (potentially enriched with CMDB data 8), code location (file path, line number), evidence, and suggested remediation. Explores related findings or historical occurrences.
  4. **Assign & Track Remediation:** Assigns the finding to a specific developer or team (potentially suggested by CODEOWNERS integration 9). Sets a due date based on severity and policy.2 Adds comments or context. Monitors the finding's status (e.g., 'Open', 'In Progress', 'Resolved', 'Accepted Risk').
  5. **Reporting:** Uses the Report Customization interface to generate a summary report for management, filtering by specific criteria (e.g., application, severity, timeframe). Exports the report (e.g., PDF, CSV).

### **2.2 Developer Persona & Journey**

* **Persona:** Responsible for writing code and remediating vulnerabilities identified in their applications or components. Needs clear, actionable information about findings relevant to their work, including precise location and remediation guidance.1
* **Goals:** Understand vulnerabilities assigned to them, locate the exact code responsible, implement the correct fix, and mark the finding as resolved. Minimize time spent navigating the security tool.
* **Typical Journey:**
  1. **Notification/Assignment:** Receives a notification (e.g., email, Jira ticket integration) about a new finding assigned to them. Clicks a link to the Finding Detail View in the web interface.
  2. **Understand Vulnerability:** Reviews the Finding Detail View. Reads the description, severity, and specific remediation advice. Notes the exact file path and line number. Examines code snippets or evidence provided. Checks git blame information 11 or CODEOWNERS data 9 if available for context on code origin/ownership.
  3. **Locate & Fix Code:** Navigates to the specified location in their IDE or code editor. Implements the suggested fix based on the remediation guidance and secure coding practices.12
  4. **Verify Fix (Locally/CI):** Runs local tests or relies on CI pipeline scans (which might use the same security tools) to verify the fix.
  5. **Update Finding Status:** Returns to the Finding Detail View in the web interface. Changes the finding status to 'Resolved' or 'Fixed'. Adds comments about the fix applied. Commits and pushes the code changes.

### **2.3 Compliance Officer / Manager Persona & Journey**

* **Persona:** Responsible for ensuring the organization adheres to internal security policies and external regulations. Needs high-level overviews, trend analysis, and reporting capabilities to demonstrate compliance and track risk reduction.13
* **Goals:** Monitor overall security posture trends, assess compliance against standards (e.g., PCI DSS, HIPAA 2), generate audit reports, track remediation SLAs, and understand organizational risk exposure.
* **Typical Journey:**
  1. **Login & Dashboard Review:** Logs in and reviews the dashboard. Focuses on high-level KPIs (e.g., overall risk score, compliance status, number of policy violations, overdue findings).4 Examines trend charts for vulnerability discovery vs. remediation rates.
  2. **Compliance Assessment:** Navigates to a dedicated compliance view or uses the Report Customization interface. Filters findings based on compliance standards (e.g., OWASP Top 10 15, CWEs 24) or specific policies. Reviews the status of critical vulnerabilities against defined SLAs.13
  3. **Risk Analysis:** Uses the Findings Exploration interface or custom reports to analyze risk exposure for critical assets or applications (identified via CMDB integration 8). Filters by severity, exploitability (EPSS 5, KEV 6), or business impact.
  4. **Generate Audit Report:** Uses the Report Customization interface to create a report for auditors or management. Selects specific metrics, timeframes, and asset groups. Exports the report in a standard format (e.g., PDF, CSV).
  5. **Monitor Remediation Progress:** Reviews dashboard widgets or reports showing the status of findings, average time to remediate, and number of overdue findings to ensure remediation efforts align with policy.13

## **3. Dashboard Layout and Component Design**

The dashboard serves as the primary landing page, providing a high-level overview of the security posture and quick access to key areas.

### **3.1 Overall Layout**

* **Navigation:** A persistent sidebar or top navigation bar providing access to major sections: Dashboard, Findings, Reports, Settings, etc.
* **Main Content Area:** A flexible grid or layout system to accommodate various dashboard widgets/components. The layout must be responsive.31
* **Global Filters (Optional):** Consider persistent filters at the top (e.g., time range, primary application/asset group) that affect all dashboard components.

### **3.2 Key Performance Indicators (KPIs) Section**

* **Purpose:** Display critical, at-a-glance metrics.
* **Component:** A dedicated section, typically at the top, showing key numbers and potentially trend indicators (e.g., up/down arrows).
* **Example KPIs:**
  + Total Open Vulnerabilities (with breakdown by critical/high) 4
  + Mean Time to Remediate (MTTR) 4
  + Scan Coverage (%)
  + Number of Overdue Vulnerabilities
  + Overall Risk Score (if applicable) 4
* **Interaction:** Clicking a KPI could navigate to the Findings Exploration view, pre-filtered accordingly.

### **3.3 Findings Summary Components (By Severity, Status, Type)**

* **Purpose:** Provide visual summaries of the current findings landscape.
* **Components:** Utilize charts (specified in Section 5) such as:
  + Donut/Pie Chart: Open Findings by Severity (Critical, High, Medium, Low, Info).32
  + Bar Chart: Findings by Status (Open, In Progress, Resolved, Accepted Risk).
  + Bar Chart or Table: Top 5 Vulnerability Types (by CWE or Rule ID).
* **Interaction:** Chart segments should be clickable, filtering the Findings Exploration view based on the selected segment. Tooltips should show exact counts on hover.

### **3.4 Recent Scans/Activities Component**

* **Purpose:** Show recent security assessment activities.
* **Component:** A list or table displaying the latest completed scans.
* **Data Points:** Scan Target (Application/Asset), Scan Tool, Completion Time, Number of New Findings, Status (Success/Failure).
* **Interaction:** Clicking a scan entry could navigate to the Findings Exploration view filtered for that specific scan, or to a dedicated scan details page.

### **3.5 Trend Analysis Visualization Component**

* **Purpose:** Show how the security posture is changing over time.
* **Component:** A line or area chart (specified in Section 5) displaying trends.
* **Example Trends:**
  + Open Vulnerabilities Over Time (stacked by severity).
  + New vs. Resolved Vulnerabilities Over Time.
* **Interaction:** Allow adjusting the time range (e.g., last 30 days, 90 days, year). Tooltips should show data points for specific dates.

## **4. Finding Exploration Interface Design**

This interface allows users to view, filter, sort, and investigate individual security findings in detail.

### **4.1 Findings List/Table View**

* **Purpose:** Display a comprehensive, filterable, and sortable list of all findings.
* **Component:** A data table, optimized for potentially large datasets using virtualization/windowing.33
* **Columns (Examples):** Finding ID, Severity (color-coded + text), Status, Vulnerability Name/Title, CWE, CVE 35, Tool Name, Asset/Component Name, File Path 35, Line Number 35, Date Detected, Assignee, Due Date. Columns should be customizable/configurable by the user.
* **Interaction:**
  + Clickable column headers for sorting.
  + Clickable rows to navigate to the Finding Detail View.
  + Bulk actions (e.g., assign, change status) via checkboxes.36
  + Hover effects for quick info (optional).
* **Responsiveness:** Implement strategies like horizontal scrolling, column hiding/prioritization, or card view for smaller screens.31

### **4.2 Filtering and Sorting Controls**

* **Purpose:** Allow users to narrow down the list of findings based on various criteria.
* **Component:** A dedicated filter panel (e.g., sidebar, dropdown section) with controls for multiple fields.
* **Filterable Fields (Examples):** Severity, Status, Tool Name, CVE, CWE, Asset/Component Name, Assignee, Date Range (Detected/Due), Tags, Exploitable (EPSS Score Range 5, KEV Status 6), etc.
* **Control Types:** Dropdowns (single/multi-select), text search, date pickers, sliders (for scores).
* **Interaction:** Applying filters should update the Findings List/Table view dynamically (or via an "Apply" button for complex queries). Allow saving filter sets. Sorting options should align with table columns.

### **4.3 Finding Detail View**

* **Purpose:** Provide comprehensive information about a single selected finding.
* **Layout:** A dedicated page or modal view with clearly separated sections.
* **Key Sections:**
  + **Summary:** Vulnerability Title, Finding ID, Severity (with CVSS score/vector if available 37), Status, Assignee, Dates (Detected, Due, Updated), Tool Name, Rule ID.
  + **Description:** Detailed explanation of the vulnerability (from the tool/rule definition).40 Include links to CWE 24, OWASP Top 10 15, etc.
  + **Location:** Precise location(s) where the vulnerability was found.42 Include File Path, Line Number(s), Code Snippet (syntax highlighted), potentially linked to SCM view. Consider resilient mapping techniques using ASTs or similar if possible.44
  + **Asset/Component Information:** Details of the affected asset (e.g., server name, IP, application name, library version). Integrate data from asset inventory/CMDB (e.g., Owner, Business Criticality).8
  + **Exploitability/Context:** Display EPSS score/percentile 5, CISA KEV status 6, internet exposure, sensitive data presence.48
  + **Remediation Guidance:** Recommended steps to fix the vulnerability (from tool/rule definition or enriched sources).40 Include links to relevant documentation or secure coding practices.50
  + **History/Audit Trail:** Log of status changes, assignments, comments.
  + **Enrichment Data:** Display linked CODEOWNERS 9, recent git blame information.9
  + **Actions:** Buttons/controls to change Status, Severity, Assignee, add Comments, link related issues, initiate remediation workflow (e.g., create Jira ticket).
* **Interaction:** Clear visual distinction between different sections. Code snippets should be easily copyable. Links should open in new tabs.

## **5. Visualization Component Specifications**

Visualizations must effectively communicate security data trends and distributions while adhering to accessibility standards.51

### **5.1 Severity Distribution Chart (e.g., Pie/Donut/Bar)**

* **Purpose:** Show the proportion of open findings across different severity levels.
* **Data:** Counts of open findings grouped by severity (Critical, High, Medium, Low, Info).
* **Type:** Donut chart is often preferred over Pie for aesthetic reasons and allows center space for a total count. Bar chart is also acceptable and potentially better for comparing exact values.
* **Interaction:** Tooltips on hover showing severity name and count/percentage. Segments should be clickable to filter the main findings list.
* **Accessibility:** Use distinct colors with sufficient contrast.53 Provide text labels or a legend clearly associating colors with severities. Ensure data is available in a tabular format as an alternative.51

### **5.2 Findings Trend Chart (e.g., Line/Area)**

* **Purpose:** Illustrate changes in finding counts over time.
* **Data:** Time series data (e.g., daily, weekly, monthly) of finding counts, potentially broken down by status (New, Resolved) or severity.
* **Type:** Line chart for showing trends of different categories (e.g., New vs. Resolved). Stacked area chart for showing total open findings broken down by severity over time.
* **Interaction:** Tooltips showing date and values for each series. Ability to zoom or select time ranges. Legend to toggle series visibility.
* **Accessibility:** Use distinct line styles/markers in addition to color.52 Ensure sufficient contrast. Provide data in a table format as an alternative.51 Clearly label axes.

### **5.3 Top Vulnerable Components/Assets Chart (e.g., Bar/Table)**

* **Purpose:** Highlight components or assets with the highest number of critical/high vulnerabilities.
* **Data:** Counts of critical/high findings grouped by component name/version or asset identifier.
* **Type:** Horizontal Bar chart (good for long labels) or a simple sorted Table. Limit to Top N (e.g., Top 10).
* **Interaction:** Bars/rows should be clickable, linking to the findings list filtered for that component/asset. Tooltips showing exact counts.
* **Accessibility:** Ensure clear labels for bars/rows and axes. Provide data in a table format.51

### **5.4 Data Table Visualization**

* **Purpose:** Display detailed findings or other list-based data.
* **Specification:** Refer to Section 4.1 (Findings List/Table View). Key aspects include virtualization for performance 33, clear alignment (text left, numbers right) 36, sortable columns, and responsive design strategies.31
* **Accessibility:** Use proper table semantics (<th> for headers with scope, <caption>). Ensure keyboard navigability and screen reader compatibility.36 Use sufficient contrast for text and borders. Consider zebra striping for readability.36

## **6. Report Customization Interface**

This interface allows users (primarily Managers and Compliance Officers) to generate tailored reports based on the security findings data.

### **6.1 Report Builder UI Elements**

* **Layout:** A dedicated page with sections for data selection, filtering, visualization options, and output format.
* **Controls:** Drag-and-drop interface or multi-select lists for choosing data fields/columns. Filter controls similar to the Findings Exploration view (Section 4.2). Chart selection options (Table, Bar, Line, Pie). Preview area.

### **6.2 Data Selection and Filtering**

* **Field Selection:** Allow users to choose which data fields (columns) to include in the report (e.g., Finding ID, Severity, CVE, Asset Name, Status, Remediation Details).
* **Filtering:** Provide comprehensive filtering capabilities identical to the Findings Exploration view (Severity, Status, Date Range, Asset Group, Tool, CWE, etc.). Allow saving and loading filter configurations.

### **6.3 Layout and Format Options**

* **Output Formats:** Support export to common formats like CSV, PDF, and potentially JSON or HTML.
* **Layout Customization (PDF/HTML):** Options to add a title, include summaries/charts, configure page orientation, add company logo/branding.
* **Templates:** Consider providing pre-defined report templates for common use cases (e.g., Executive Summary, Compliance Audit, Developer Remediation List).

### **6.4 Saving and Scheduling Reports**

* **Saving:** Allow users to save report configurations (selected data, filters, layout) for later re-use.
* **Scheduling:** Option to schedule report generation (e.g., weekly, monthly) and automatic delivery via email or other notification channels.

## **7. Responsive Design Considerations**

The interface must be usable across various screen sizes, from desktops to tablets and potentially mobile devices.31

### **7.1 Breakpoint Strategy**

* Define standard breakpoints (e.g., mobile, tablet, desktop) based on common device widths.
* Use CSS media queries to apply different styles and layouts at each breakpoint.

### **7.2 Layout Adjustments (Navigation, Dashboard Widgets)**

* **Navigation:** Sidebar navigation might collapse into a hamburger menu on smaller screens. Top navigation might wrap or hide secondary items.
* **Dashboard Widgets:** Widgets arranged in a grid on desktop should reflow into a single column or fewer columns on smaller screens.31 Chart sizes should adapt, potentially simplifying complex visualizations on mobile.

### **7.3 Table Responsiveness Techniques**

* **Challenge:** Data tables are notoriously difficult on small screens.36
* **Strategies:**
  + **Horizontal Scrolling:** Simplest approach, allow horizontal scrolling within the table container. Ensure visual cues indicate scrollability.
  + **Column Hiding/Prioritization:** Hide less critical columns on smaller screens, potentially allowing users to toggle column visibility.36
  + **Card View:** Transform each table row into a stacked "card" layout on mobile, displaying data vertically.36
  + **Combination:** Use a mix, e.g., hide some columns and allow horizontal scrolling for the rest.
* **Selection:** The chosen strategy should prioritize readability and access to essential data on smaller viewports.

## **8. Accessibility Requirements**

The application must conform to Web Content Accessibility Guidelines (WCAG) 2.1 Level AA.53 This ensures usability for people with various disabilities.

### **8.1 Perceivable**

* **Text Alternatives:** Provide text alternatives (e.g., alt text) for all non-text content like images and icons (WCAG 1.1.1).
* **Color Contrast:** Ensure sufficient contrast between text and background (4.5:1 for normal text, 3:1 for large text) and for UI components and graphical elements (3:1) (WCAG 1.4.3, 1.4.11).53 Avoid using color as the sole means of conveying information (WCAG 1.4.1).53
* **Resizable Text:** Allow text resizing up to 200% without loss of content or functionality (WCAG 1.4.4).53
* **Adaptable Content:** Use semantic HTML to ensure content can be presented in different ways (e.g., by screen readers) without losing information or structure (WCAG 1.3.1).

### **8.2 Operable**

* **Keyboard Accessible:** All functionality must be operable through a keyboard interface without requiring specific timings (WCAG 2.1.1).53 Ensure no keyboard traps exist where focus cannot be moved away from a component using the keyboard (WCAG 2.1.2).53
* **Focus Management:** Provide a visible focus indicator for keyboard users (WCAG 2.4.7).53 Ensure logical focus order when navigating with the keyboard (WCAG 2.4.3).53
* **Timing:** Provide users enough time to read and use content. If timeouts exist (e.g., session timeout), warn users and allow them to extend the time (WCAG 2.2.1, 2.2.6).53 Avoid content that flashes more than three times per second (WCAG 2.3.1).53
* **Navigation:** Provide ways to help users navigate, find content, and determine where they are (e.g., breadcrumbs, clear page titles (WCAG 2.4.2), headings (WCAG 2.4.6)).53

### **8.3 Understandable**

* **Readable Text:** Ensure text content is readable and understandable. Specify the language of the page (WCAG 3.1.1) and parts (WCAG 3.1.2).53
* **Predictable Navigation:** Make web pages appear and operate in predictable ways. Ensure consistent navigation patterns (WCAG 3.2.3) and consistent identification of components (WCAG 3.2.4).53 Avoid changes of context initiated without user action (WCAG 3.2.1, 3.2.2).53
* **Input Assistance:** Help users avoid and correct mistakes. Provide clear error identification (WCAG 3.3.1), labels or instructions for input fields (WCAG 3.3.2), and error suggestions where appropriate (WCAG 3.3.3).53

### **8.4 Robust**

* **Compatibility:** Maximize compatibility with current and future user agents, including assistive technologies, by using valid HTML/CSS and ARIA (Accessible Rich Internet Applications) where necessary (WCAG 4.1.1, 4.1.2).

### **8.5 Data Visualization Accessibility**

* **Color:** Use colorblind-safe palettes and ensure sufficient contrast.51 Do not rely solely on color to convey information; use patterns, shapes, or direct labels as well.52
* **Labels:** Use direct labels on chart elements (bars, lines, slices) instead of relying solely on legends.51
* **Alternatives:** Provide data in alternative formats, such as accessible data tables, alongside visualizations.51 Provide text summaries describing the key insights from the visualization.51
* **Tool Choice:** Select charting libraries known to support accessibility features (e.g., keyboard navigation, ARIA attributes).

## **9. Performance Optimization Strategy**

Handling potentially large sets of security findings requires a deliberate performance optimization strategy, particularly for the frontend.33

### **9.1 Large Dataset Handling (Findings List/Tables)**

* **Virtualization/Windowing:** Mandate the use of virtualization libraries (e.g., react-window, react-virtualized, TanStack Virtual) for rendering long lists or tables of findings. Only DOM nodes for currently visible rows should be rendered to maintain UI responsiveness.33 This significantly reduces the performance cost associated with rendering thousands of elements.34
* **Pagination/Infinite Scrolling:** Implement server-side pagination for fetching findings data. The frontend should only request the data needed for the current view (page). Infinite scrolling can be an alternative UX, loading more data as the user scrolls, but pagination offers more predictable performance and easier navigation.34 Define a reasonable default page size (e.g., 50 or 100 findings).

### **9.2 Data Fetching and Caching**

* **Server-Side Filtering/Sorting:** Ensure backend APIs support filtering and sorting operations server-side. The frontend should pass filter/sort parameters to the API rather than fetching all data and processing it client-side.
* **Frontend Caching:** Employ data-fetching libraries like React Query or SWR. These libraries provide caching, background data synchronization, and stale-while-revalidate strategies, reducing unnecessary API calls and improving perceived performance. Configure appropriate cache invalidation based on user actions or time intervals.

### **9.3 Component Rendering Optimization**

* **Memoization:** Utilize React.memo for functional components to prevent re-renders when props haven't changed.33 Use useMemo to memoize results of expensive calculations within components. Use useCallback to memoize callback functions passed down as props, preventing unnecessary re-renders of child components optimized with React.memo.33 Apply these techniques judiciously, focusing on components that render frequently or are computationally intensive.
* **Immutable State Updates:** Strictly enforce immutable state updates. Direct mutation can prevent React from detecting changes and skipping necessary re-renders, leading to UI inconsistencies.33 Use state setter functions correctly (e.g., functional updates setCount(c => c + 1)) when new state depends on the previous state.

### **9.4 Bundle Size Optimization**

* **Code Splitting:** Implement route-based code splitting using React.lazy and Suspense. Load code for different sections (Dashboard, Findings, Reports) only when the user navigates to them.33 Consider component-level code splitting for large, infrequently used components (e.g., complex report builder UI).
* **Dependency Analysis:** Regularly use tools like webpack-bundle-analyzer or source-map-explorer to inspect the production bundle size. Identify large dependencies and evaluate alternatives or strategies for reducing their impact (e.g., importing specific functions instead of entire libraries).33
* **Tree Shaking:** Ensure the build process (e.g., Webpack, Rollup) is configured for effective tree shaking to eliminate unused code from the final bundle.

## **10. Frontend State Management Approach**

Selecting an appropriate state management strategy is crucial for maintainability, scalability, and performance, especially in applications with shared state and complex interactions like filtering and reporting.55

### **10.1 Evaluation of Options**

* **React Context API:**
  + *Pros:* Built-in, simple for passing data down the tree without prop drilling.
  + *Cons:* Performance issues with frequent updates as all consumers re-render.55 Not ideal for complex global state logic.
* **Redux Toolkit (RTK):**
  + *Pros:* Predictable state container, excellent developer tools (time-travel debugging), robust ecosystem, good for complex state logic and middleware (e.g., async actions).55 Enforces clear patterns.
  + *Cons:* Can involve significant boilerplate code, potentially steeper learning curve.55 Might be overkill if application complexity remains moderate.
* **Zustand:**
  + *Pros:* Simple API, minimal boilerplate, good performance, unopinionated, hooks-based.55 Scales well for medium-to-large applications.
  + *Cons:* Less structured than Redux, fewer built-in dev tools compared to Redux DevTools.
* **Jotai:**
  + *Pros:* Atomic state model, excellent for granular updates and managing interdependent state slices, minimal boilerplate.55 Potentially very performant for specific use cases.
  + *Cons:* Atomic approach can be less familiar to developers accustomed to monolithic stores. Risk of "atom proliferation" if not managed carefully.55

### **10.2 Recommended Approach and Justification**

**Recommendation:** **Zustand**

**Justification:** The security assessment dashboard requires shared global state for elements like filters, user preferences, and potentially some UI state across different views (Dashboard, Findings List, Report Builder). The complexity involves managing filter states, coordinating data fetching based on filters, and handling UI interactions. While Redux Toolkit offers robust solutions, its boilerplate might add unnecessary overhead for the anticipated complexity. React Context alone is likely insufficient due to performance concerns with shared, frequently updated state like filters.55 Jotai's atomic model is powerful but might introduce unnecessary complexity and a steeper learning curve for this application type. Zustand provides a pragmatic balance: it offers a simple, hook-based API, minimal boilerplate, good performance characteristics, and sufficient structure for managing the global state required for cross-component communication (filters affecting the findings list) and UI persistence.55 Its unopinionated nature allows flexibility, and migrating to Redux later remains feasible if complexity dramatically increases.

### **10.3 High-Level State Structure**

A potential Zustand store structure could include slices (or separate stores) for:

* **filtersStore:** Manages the state of all filters applied in the Findings Exploration view (severity, status, date range, tool, text search, etc.). Includes actions to update filters.
* **findingsStore:** Could potentially cache fetched findings data, manage pagination state (current page, total pages), and loading/error states related to findings fetching. (Alternatively, use React Query/SWR for server state).
* **uiStore:** Manages global UI state, such as sidebar visibility, current theme, or global notifications.
* **reportStore:** Manages the state of the report customization interface (selected fields, chosen layout, report configuration).
* **userPreferencesStore:** Stores user-specific settings like default sort order, table column visibility preferences, etc.

Local component state (useState, useReducer) should still be used for state confined to a single component (e.g., form input values before submission, toggle states within a specific widget).

## **11. Example React Component Architecture**

A well-defined architecture promotes maintainability, testability, and scalability.

### **11.1 Directory Structure**

A feature-based directory structure is recommended:

/src  
|-- /app # Global setup, routing, core layout  
|-- /assets # Static assets (images, fonts)  
|-- /components # Shared, reusable presentational components (Button, Table, ChartWrapper)  
|-- /features # Feature-specific components, hooks, state, services  
| |-- /dashboard # Components specific to the dashboard view  
| |-- /findings # Components for findings list, filtering, detail view  
| |-- /reports # Components for report building and viewing  
| |-- /settings # Components for user/system settings  
|-- /hooks # Shared custom hooks  
|-- /lib # External library configurations, API clients  
|-- /pages # Top-level page components mapping to routes  
|-- /store # Global state management (Zustand stores)  
|-- /styles # Global styles, theme configuration  
|-- /types # Shared TypeScript types/interfaces  
|-- /utils # Shared utility functions

### **11.2 Component Strategy**

Adopt a strategy combining **Feature-based organization** with **Container/Presentational principles** within features.

* **Feature Folders:** Group all code related to a major feature (e.g., findings) together. This includes page components, feature-specific UI components, hooks, and potentially feature-specific state logic.
* **Container Components (within features):** Components responsible for fetching data, managing feature-specific state (using local state or interacting with the global store), and passing data/callbacks down to presentational components. Often correspond to page-level components or complex widgets.
* **Presentational Components (within features or shared):** Components focused solely on rendering UI based on props received. They are reusable and do not contain business logic or data fetching. Shared presentational components reside in /src/components.

### **11.3 Example Component Breakdown (Pseudo-code/Descriptions)**

* **src/pages/FindingsPage.tsx:**
  + Renders the overall layout for the findings exploration view.
  + Likely renders FindingsFilterBar and FindingsListContainer.
* **src/features/findings/components/FindingsFilterBar.tsx:**
  + Renders filter UI elements (dropdowns, search input, date pickers).
  + Reads current filter values from filtersStore.
  + Calls actions from filtersStore on user interaction to update global filter state.
* **src/features/findings/components/FindingsListContainer.tsx:**
  + Container component.
  + Subscribes to filtersStore to get current filters.
  + Uses a custom hook (e.g., useFindingsData) to fetch findings data based on current filters and pagination state.
  + Handles loading and error states.
  + Manages local pagination state (current page).
  + Passes fetched findings data, pagination controls, and sort state/handlers to FindingsTable.
* **src/components/Table/FindingsTable.tsx:**
  + Presentational component.
  + Receives findings data, column definitions, sort state, and event handlers (onSortChange, onRowClick, onPageChange) as props.
  + Uses a virtualization library (react-window or similar) to render only visible rows.33
  + Renders table headers (clickable for sorting) and rows.
  + Invokes callback props on user interaction.
* **src/pages/FindingDetailPage.tsx:**
  + Retrieves finding ID from route parameters.
  + Uses a custom hook to fetch detailed data for the specific finding ID.
  + Handles loading/error states.
  + Renders FindingDetailView with the fetched data.
* **src/features/findings/components/FindingDetailView.tsx:**
  + Presentational component.
  + Receives detailed finding data as props.
  + Renders different sections (Summary, Location, Remediation, Context, etc.) using smaller, focused presentational components.
* **src/features/dashboard/components/SeverityChartWidget.tsx:**
  + Container component for the severity distribution chart.
  + Fetches aggregated severity data (or receives it via props if dashboard data is fetched centrally).
  + Passes formatted data to a shared DonutChart component.
* **src/components/Charts/DonutChart.tsx:**
  + Generic, reusable presentational chart component.
  + Receives data and configuration options as props.
  + Uses a charting library (e.g., Recharts, Nivo) to render the chart.
  + Implements necessary accessibility features.

### **11.4 Data Flow Example (Applying a Filter)**

1. **User Interaction:** User selects 'High' severity in FindingsFilterBar.
2. **State Update:** FindingsFilterBar calls an action (e.g., updateSeverityFilter('High')) from the Zustand filtersStore.
3. **Store Update:** The filtersStore updates its state, changing the severity filter to 'High'.
4. **Component Re-render:** FindingsListContainer, subscribed to filtersStore, re-renders because the filter state it depends on has changed.
5. **Data Re-fetch:** Inside FindingsListContainer, the useFindingsData hook detects the filter change and triggers a new API call to fetch findings, passing { severity: 'High', page: 1,... } as parameters. Pagination is reset to page 1.
6. **Loading State:** FindingsListContainer displays a loading indicator while data is fetched.
7. **Data Update:** The API returns the filtered findings. useFindingsData updates its state with the new data and loading/error status.
8. **Pass Data to Table:** FindingsListContainer re-renders again, passing the new, filtered findings data to the FindingsTable presentational component.
9. **UI Update:** FindingsTable renders the updated list of high-severity findings.

## **12. Conclusion**

This specification outlines the requirements and design considerations for a web interface dedicated to displaying and managing security assessment results. Key focus areas include providing tailored experiences for different user personas (Security Analyst, Developer, Compliance Officer/Manager), a clear and informative dashboard, robust finding exploration capabilities, and flexible reporting options.

Emphasis has been placed on critical non-functional requirements, including **responsive design** to ensure usability across devices, strict adherence to **WCAG 2.1 AA accessibility guidelines** to support all users, and **performance optimization strategies** (particularly virtualization and server-side processing) to handle potentially large volumes of findings data efficiently.

The recommended frontend architecture leverages **React** with **Zustand** for global state management, promoting a balance between simplicity, performance, and scalability. A feature-based directory structure combined with container/presentational component patterns is advised for maintainability.

**Next Steps:**

* Develop detailed UI mockups and prototypes based on these specifications.
* Define the precise API endpoints and data schemas required to support the frontend interactions.
* Create user stories and populate the development backlog based on the defined features and user journeys.
* Begin implementation, prioritizing core functionality and iteratively incorporating visualizations, reporting, and advanced features.
* Continuously test for performance and accessibility throughout the development lifecycle.

Adherence to these specifications will guide the development team in building a robust, user-friendly, accessible, and performant platform for effective vulnerability management.

#### Works cited

1. What Is User Journey Mapping? | Salesforce US, accessed May 8, 2025, <https://www.salesforce.com/marketing/customer-journey/user-journey-mapping/>
2. Vulnerability Management: Complete Guide to Process and Tools - BlueVoyant, accessed May 8, 2025, <https://www.bluevoyant.com/knowledge-center/vulnerability-management-complete-guide-to-process-and-tools>
3. Vulnerability Management (As.) or Security Analyst : r/cybersecurity - Reddit, accessed May 8, 2025, <https://www.reddit.com/r/cybersecurity/comments/1jdomvv/vulnerability_management_as_or_security_analyst/>
4. 15 Critical Vulnerability Management KPIs | Indusface Blog, accessed May 8, 2025, <https://www.indusface.com/blog/measuring-the-performance-of-vulnerability-management-which-metrics-matter-which-dont/?amp>
5. Exploit Prediction Scoring System (EPSS), accessed May 8, 2025, <https://www.first.org/epss/api>
6. Reducing the Significant Risk of Known Exploited Vulnerabilities - CISA, accessed May 8, 2025, <https://www.cisa.gov/known-exploited-vulnerabilities>
7. Known Exploited Vulnerabilities Catalog | CISA, accessed May 8, 2025, <https://www.cisa.gov/known-exploited-vulnerabilities-catalog>
8. CyberSecurity Asset Management 3.0 (CSAM) | Qualys, accessed May 8, 2025, <https://www.qualys.com/apps/cybersecurity-asset-management/>
9. The EPSS Model, accessed May 8, 2025, <https://www.first.org/epss/model>
10. User Journey Map Examples: A Comprehensive Guide for SaaS Success | Userflow Blog, accessed May 8, 2025, <https://www.userflow.com/blog/user-journey-map-examples-a-comprehensive-guide-for-saas-success>
11. git-blame Documentation - Git, accessed May 8, 2025, <https://git-scm.com/docs/git-blame>
12. The 2025 In-Depth Guide to OWASP Top 10 Vulnerabilities & How to Prevent Them - Jit.io, accessed May 8, 2025, <https://www.jit.io/resources/security-standards/the-in-depth-guide-to-owasps-top-10-vulnerabilities>
13. What is Compliance Reporting and Types of Reports - Sprinto, accessed May 8, 2025, <https://sprinto.com/blog/compliance-reporting/>
14. What is compliance reporting? A detailed guide with steps to the reporting process - Vanta, accessed May 8, 2025, <https://www.vanta.com/collection/grc/compliance-reporting>
15. A03 Injection - OWASP Top 10:2021, accessed May 8, 2025, <https://owasp.org/Top10/A03_2021-Injection/>
16. A01 Broken Access Control - OWASP Top 10:2021, accessed May 8, 2025, <https://owasp.org/Top10/A01_2021-Broken_Access_Control/>
17. A02 Cryptographic Failures - OWASP Top 10:2021, accessed May 8, 2025, <https://owasp.org/Top10/A02_2021-Cryptographic_Failures/>
18. A04 Insecure Design - OWASP Top 10:2021, accessed May 8, 2025, <https://owasp.org/Top10/A04_2021-Insecure_Design/>
19. A05 Security Misconfiguration - OWASP Top 10:2021, accessed May 8, 2025, <https://owasp.org/Top10/A05_2021-Security_Misconfiguration/>
20. A08 Software and Data Integrity Failures - OWASP Top 10:2021, accessed May 8, 2025, <https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/>
21. A07 Identification and Authentication Failures - OWASP Top 10:2021, accessed May 8, 2025, <https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/>
22. A09 Security Logging and Monitoring Failures - OWASP Top 10:2021, accessed May 8, 2025, <https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/>
23. accessed December 31, 1969, <https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_(SSRF)/>
24. CWE-611: Improper Restriction of XML External Entity Reference, accessed May 8, 2025, <https://cwe.mitre.org/data/definitions/611.html>
25. CWE-89: Improper Neutralization of Special Elements used ... - CWE, accessed May 8, 2025, <https://cwe.mitre.org/data/definitions/89.html>
26. CWE-79: Improper Neutralization of Input During Web Page ... - CWE, accessed May 8, 2025, <https://cwe.mitre.org/data/definitions/79.html>
27. CWE-78: Improper Neutralization of Special Elements used ... - CWE, accessed May 8, 2025, <https://cwe.mitre.org/data/definitions/78.html>
28. CWE-502: Deserialization of Untrusted Data (4.17) - CWE, accessed May 8, 2025, <https://cwe.mitre.org/data/definitions/502.html>
29. CWE-798: Use of Hard-coded Credentials (4.17) - CWE, accessed May 8, 2025, <https://cwe.mitre.org/data/definitions/798.html>
30. CWE-471: Modification of Assumed-Immutable Data (MAID) (4.17), accessed May 8, 2025, <https://cwe.mitre.org/data/definitions/471.html>
31. Effective Dashboard Design Principles for 2025 - UXPin, accessed May 8, 2025, <https://www.uxpin.com/studio/blog/dashboard-design-principles/>
32. Vulnerability Management Dashboard - Slide Geeks, accessed May 8, 2025, <https://www.slidegeeks.com/powerpoint/Vulnerability-Management-Dashboard>
33. React Optimization 101: Tips and Tricks - Cloudinary, accessed May 8, 2025, <https://cloudinary.com/guides/front-end-development/react-optimization>
34. How to Handle Large Datasets in Frontend Applications | Blog - GreatFrontEnd, accessed May 8, 2025, <https://www.greatfrontend.com/blog/how-to-handle-large-datasets-in-front-end-applications>
35. Documentation: 17: CREATE INDEX - PostgreSQL, accessed May 8, 2025, <https://www.postgresql.org/docs/current/sql-createindex.html>
36. Designing effective data table UI – best practices and tips - Justinmind, accessed May 8, 2025, <https://www.justinmind.com/ui-design/data-table>
37. CVSS v3.1 Specification Document - FIRST.org, accessed May 8, 2025, <https://www.first.org/cvss/v3.1/specification-document>
38. Assess severity of a vulnerability - DigiCert documentation, accessed May 8, 2025, <https://docs.digicert.com/en/software-trust-manager/threat-detection/best-practices-for-common-vulnerabilities-and-exposures/assess-the-risk-of-a-vulnerability.html>
39. Common Vulnerability Scoring System (CVSS) Value - Ivanti, accessed May 8, 2025, <https://www.ivanti.com/blog/common-vulnerability-scoring-system-cvss>
40. Evaluating Macie findings with AWS Security Hub - Amazon Macie, accessed May 8, 2025, <https://docs.aws.amazon.com/macie/latest/user/securityhub-integration.html>
41. Static Analysis Results Interchange Format (SARIF) Version 2.1.0, accessed May 8, 2025, <https://docs.oasis-open.org/sarif/sarif/v2.1.0/os/sarif-v2.1.0-os.html>
42. Source Code Analysis Tools | OWASP Foundation, accessed May 8, 2025, <https://owasp.org/www-community/Source_Code_Analysis_Tools>
43. What Is Code Scanning? Approaches and Best Practices, accessed May 8, 2025, <https://www.legitsecurity.com/aspm-knowledge-base/code-scanning-best-practices>
44. (PDF) Generalized Vulnerability Extrapolation using Abstract Syntax Trees - ResearchGate, accessed May 8, 2025, <https://www.researchgate.net/publication/233997025_Generalized_Vulnerability_Extrapolation_using_Abstract_Syntax_Trees>
45. Generalized Vulnerability Extrapolation using Abstract Syntax Trees - MLSEC, accessed May 8, 2025, <https://mlsec.tu-berlin.de/docs/2012-acsac.pdf>
46. Vulnerability Detection in Source Code via Subgraph Isomorphism Analysis, accessed May 8, 2025, <https://wpcdn.web.wsu.edu/wp-vcea/uploads/sites/3267/2024/05/Vulnerability-Detection-in-Source-Code-via-Subgraph-Isomorphism-Analysis.pdf>
47. General | Developer Tools | Abstract Syntax Tree - Codecademy, accessed May 8, 2025, <https://www.codecademy.com/resources/docs/general/developer-tools/abstract-syntax-tree>
48. Contextual Risk Estimation for Effective Prioritization | Microsoft ..., accessed May 8, 2025, <https://techcommunity.microsoft.com/t5/microsoft-defender-for-cloud/contextual-risk-estimation-for-effective-prioritization/ba-p/4035425>
49. Optional top-level ASFF attributes - AWS Security Hub, accessed May 8, 2025, <https://docs.aws.amazon.com/securityhub/latest/userguide/asff-top-level-attributes.html>
50. Security in Django | Django documentation | Django, accessed May 8, 2025, <https://docs.djangoproject.com/en/5.0/topics/security/#template-system-security>
51. Create Accessible Data Visualizations | UMass Office of the President, accessed May 8, 2025, <https://www.umassp.edu/inclusive-by-design/digital-accessibility-standards-and-resources/create-accessible-data>
52. Data Visualizations for Everybody: A Lesson on Accessibility, accessed May 8, 2025, <https://dataservices.library.jhu.edu/wp-content/uploads/sites/41/2024/03/24LDW_AccessibleDataViz_02-2024_NancyShin.pdf>
53. Web Content Accessibility Guidelines (WCAG) 2.1 - W3C, accessed May 8, 2025, <https://www.w3.org/TR/WCAG21/>
54. Accessibility Quick Tip: WCAG 2.1 in a Nutshell - Colorado Virtual Library, accessed May 8, 2025, <https://www.coloradovirtuallibrary.org/technology/accessibility/accessibility-quick-tip-wcag-2-1-in-a-nutshell/>
55. State Management in 2025: When to Use Context, Redux, Zustand, or Jotai, accessed May 8, 2025, <https://dev.to/hijazi313/state-management-in-2025-when-to-use-context-redux-zustand-or-jotai-2d2k>
56. Top 15 React state management libraries to use in 2025. - Nimblechapps, accessed May 8, 2025, <https://www.nimblechapps.com/blog/15-react-state-management-libraries-to-use-in-2025>