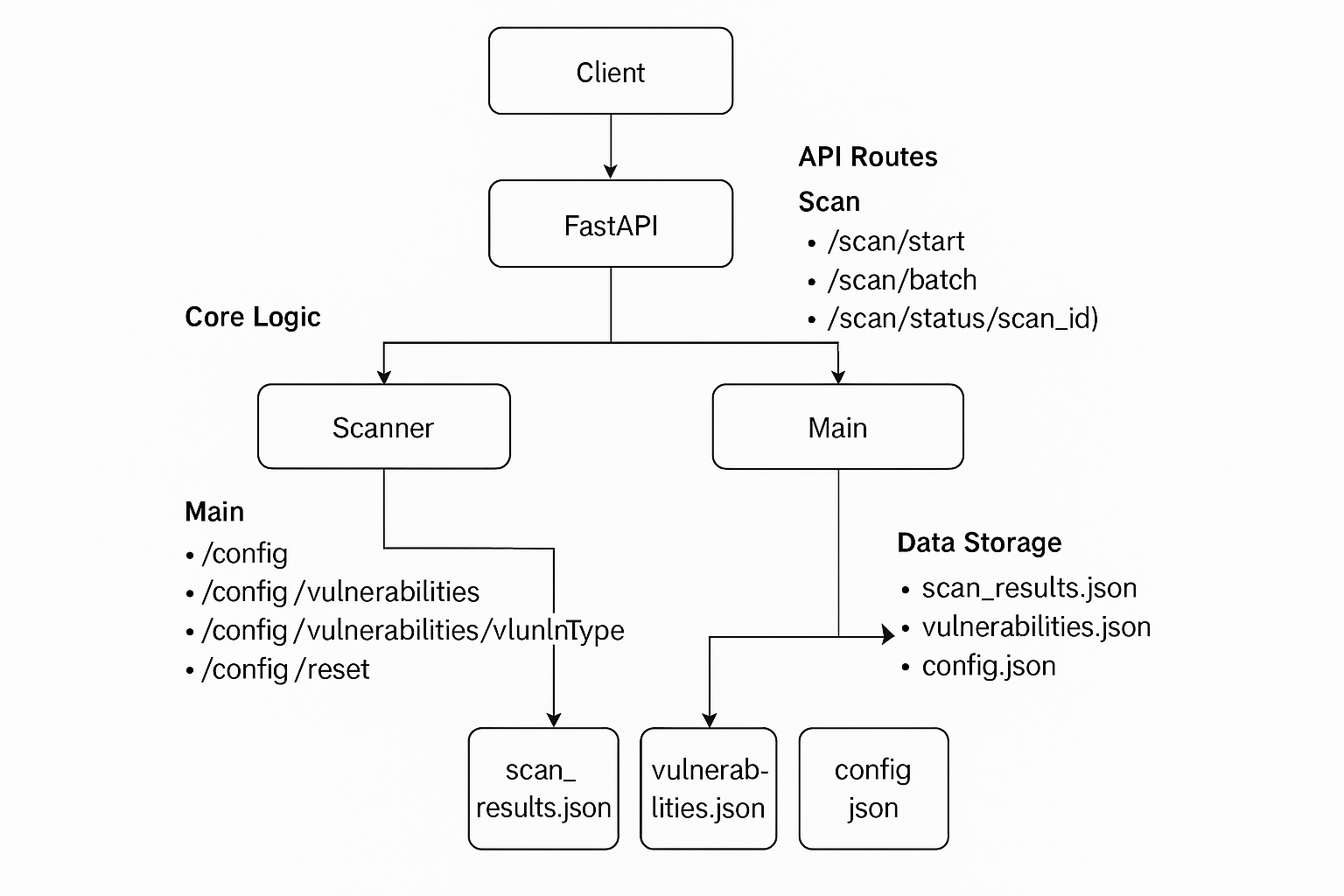
**CySmart.AI Backend**

This is a Fast API-based backend for scanning URLs, managing vulnerability reports, and handling configurations.

**Features**

* Start and monitor URL scans
* Generate reports with vulnerability details
* Manage scanning configuration and vulnerability rules
* JSON-based data storage for simplicity

**Architecture Diagram**

****

**Data Files**

All persistent data is stored in the data/ folder as JSON files.

* config.json – Settings used during scans
* vulnerabilities.json – Known vulnerability patterns/rules
* scan\_results.json – Output of scans

**API Endpoints**

1. **Scan API**

|  |  |  |
| --- | --- | --- |
| **Method** | **Endpoint** | **Description** |
| POST | api/scan/start | Start a new scan |
| POST | api/scan/batch | Batch scan multiple URLs |
| GET | api/scan/status/{id} | Get scan status by ID |
| GET | api/scan/active | List active scans |

1. **Report API**

|  |  |  |
| --- | --- | --- |
| **Method** | **Endpoint** | **Description** |
| GET | /api/report | Get all Reports |
| GET | /api/report/{report\_id} | Get details of a specific report |
| GET | /api/report/summary/recent?days=7 | Get report summary of recent X days |
| GET | |  | | --- | |  |  |  | | --- | | /api/report/stats/vulnerability\_types | | List active scansGet stats by vulnerability types |
| DELETE | /api/report/{report\_id} | Delete a report |

1. **Config API**

|  |  |  |
| --- | --- | --- |
| **Method** | **Endpoint** | **Description** |
| GET | /api/config | Get current configuration |
| PATCH | /api/config | |  | | --- | |  |  |  | | --- | | Update configuration | |
| GET | /api/config/vulnerabilities | |  | | --- | |  |  |  | | --- | | Get vulnerability rules | |
| PATCH | |  | | --- | |  |  |  | | --- | | /api/config/vulnerabilities/{vulnType} | | Update a specific vulnerability rule |
| POST | /api/config/reset | Reset configuration |

**CySmart.AI Frontend**

1. ***Folder Structure:***

|  |  |
| --- | --- |
| **Folder/File** | **Purpose** |
| main.js | Vue app entry point, sets up plugins (Pinia, Element Plus, Router). |
| App.vue | Global Layout shell (navbar, footer, router-view). |
| router/index.js | Defines routes for Dashboard, Scan Center, Demo, and Reports |

***src/api/index.js:***

|  |  |
| --- | --- |
| **Folder/File** | **Purpose** |
| scanApi | Handles starting scans (single and batch), checking scan status, and listing active scans |
| reportApi | Handles fetching reports, report summaries, vulnerability stats, and deleting reports |
| configApi | Retrieves and updates configuration, vulnerability rules, and library. |

***src/router/index.js:***

Manages route definitions and navigation between views like Dashboard, Scan Center, Reports, Settings, etc.

***Src/store:***

|  |  |
| --- | --- |
| **Folder/File** | **Purpose** |
| scanStore.js | Store for managing web vulnerability scan tasks, configurations, and results. |
| configStore.js | Managing global configuration and vulnerability library data in the application.It provides actions to fetch, update, and reset configuration settings, as well as manage the vulnerability rule |

***src/views/:***

|  |  |
| --- | --- |
| **Folder/File** | **Purpose** |
| Dashboard.vue | Displays scan summaries, vulnerability statistics, security score, and trends. |
| Scan.vue | Allows users to run manual scans (URL input, config settings). |
| ScanCenter.vue | Central place to track running and completed scans |
| Reports.vue | Lists historical scan reports and links to details. |
| ReportDetail.vue | Displays a specific scan's results (vulnerabilities, metadata, possibly guidance). |
| Setting.vue | Configurations for scanner rules, thresholds, etc. |
| DemoImplementation.vue | Likely a guided implementation demo or sample site scan. |
| NotFount.vue | 404 fallback view for undefined routes. |

***src/components:***

|  |  |
| --- | --- |
| **Folder/File** | **Purpose** |
| AppNavbar.vue | |  | | --- | |  |  |  | | --- | | Top navigation bar shared across pages | |
| ReportPDFTemplate.vue | Probably used to generate downloadable PDF reports of scans. |
| ScanComponent.vue | Core scan interface or reusable scanner UI |
| ScanStatusBadge.vue | Displays scan status in a badge (running, success, failed) |
| SecurityScoreChart.vue | Circular chart (progress/dashboard type) for security score. |
| VulnerabilitiesBadge.vue | |  | | --- | |  |  |  | | --- | | UI badge for displaying a count or level of vulnerabilities | |
| VulnerabilityBadge.vue | Smaller reusable badge, possibly color-coded for severity. |
| VulnerabilityTable.vue | Table component to list vulnerabilities found during scan. |
| VulnerabilityTypeChart.vue | |  | | --- | |  |  |  | | --- | | Chart showing breakdown of vulnerabilities by type (e.g., XSS, SQLi) | |
| VulnerabilityVisualizer.vue | a graphical or network-based visualization of detected vulnerabilities. |
| guidance/\*.vue | Contains guidance/help articles on mitigating specific vulnerability types like SQL Injection, XSS, and others. |

1. ***Scan Workflow Steps & UI Mapping***

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Description** | **File/view** | **Key Function** |
| Prepare | Initialize scan and check accessibility | ScanComponent.vue | simulateScanProcess(scanId) (see logs like 'Initializing the scanning environment...' and 'Checking accessibility of target websites...') |
| Crawl | Discover pages and links | ScanComponent.vue | simulateScanProcess(scanId) (see log 'The website is accessible and start crawling the page...' and page discovery) |
| Detection | Test for vulnerabilities | ScanComponent.vue | simulateScanProcess(scanId) (see log 'Start detecting potential vulnerabilities...' and simulated vulnerability logs) |
| Analyze | Analyze and summarize findings | ScanComponent.vue | simulateScanProcess(scanId) (see log 'Analyzing collected data...') |
| Finish | Complete scan and update results | ScanComponent.vue | completeScan(scanId) (called at the end of simulateScanProcess) |

**Where is the crawler and Scanner code?**

**Frontend:** The scan workflow steps are simulated in ***ScanComponent.vue***, especially in ***simulateScanProcess(scanId).***

**Backend:** The Crawling and Detection logic in the backend, likely in ***backend/app/core/scanner.py*** and called by ***run\_scan*** in ***scan.py***.

* See run\_scan and scan\_url\_for\_vulnerabilities for actual crawling and detection logic.
* Frontend simulates these steps for UI feedback.

**UI:** The scan progress and phases are shown in the Scan Center view (***ScanCenter.vue***), with phase labels like "***Preparing***", "***Crawling***", "***Detecting***", "***Analyzing***", "***Completed***"

1. ***Which OWASP Top 10 Vulnerabilities This Tool Detects?***

|  |  |  |
| --- | --- | --- |
| **Vulnerability Type** | **OWASP Top 10 2021 Mapping** | **Description** |
| SQL Injection (sql\_injection) | **A03:2021 - Injection** | Tool scans for patterns like SQL syntax errors, MySQL/Oracle/Postgres-specific messages |
| Cross-Site Scripting (xss) | **A03:2021 - Injection** | Looks for alert payloads, document.cookie, and JS injection patterns |
| CSRF (csrf) | **A01:2021 – Broken Access Control** | Matches missing token messages, token validation failures |
| Insecure File Upload (file\_upload) | **A05:2021 – Security Misconfiguration** | Searches for dangerous file extensions like .php, .jsp, .exe, .sh |

1. ***Where in the Code Are These Problems Found?***

**Configuration of Detected Vulnerabilities:**

scanStore.js

* See the scanConfig object, especially vulnerability\_definitions:
* ***'sql\_injection'***: SQL Injection
* ***'xss'***: Cross-site Scripting (XSS)
* ***'csrf'***: Cross-site Request Forgery (CSRF)
* ***'file***\_***upload'***: File Upload Vulnerability

**Detection Logic:**

ScanComponent.vue

* (where scan simulation occurs)

scanStore.js

* Functions like ***generateMockVulnerabilities*** simulate detection of these vulnerabilities.

**UI Display:**

Reports.vue

* Displays detected vulnerabilities in reports.

Dashboard.vue

* Shows statistics and summary of vulnerabilities.

**Backend (Detection):**

scanner.py

* It would contain the actual crawling and detection logic for SQLi, XSS, CSRF, and file upload vulnerabilities.

**How to Run the Project**

Before you begin, ensure the following tools are installed on your system:

1. **Prerequisites**

* Python (version 3.8 to 3.11)
* Node.js (version 16 or above)
* Visual Studio with C++ build tools (for native module compilation)
* Rust (install via rustup-init)

1. **Running the Backend**

Option 1: Using the preconfigured script

* Navigate to the project root directory (CySmart-ai).
* Double-click the ***Start.bat*** file.

Option 2: Manual steps via terminal

* Open a terminal and navigate to the backend directory.
* Install the required Python packages:  
  ***pip install -r requirements.txt***
* Start the backend server:  
  ***python run.py***

1. **Running the Frontend**

* Navigate to the frontend directory:  
  ***cd frontend***
* Install dependencies:  
  ***npm install***
* Start the development server:  
  ***npm run dev***

Once both servers are running, visit <http://localhost:3000> (or your specified port) in your browser to access the application.