VIOE Setup Guide

Vulnerability Intelligence & Ownership Engine

Version: 1.0 **Date:** February 1, 2026 **Document Type:** Developer Setup & Configuration Guide

---

Table of Contents

A. Overview

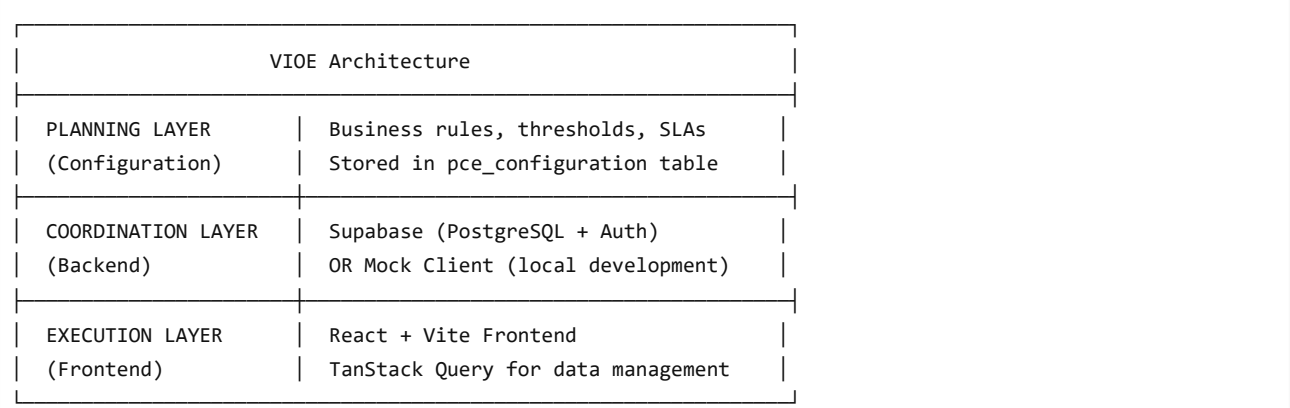
What is VIOE?

VIOE (Vulnerability Intelligence & Ownership Engine) is a comprehensive vulnerability management platform designed to help security teams:

- **Track and manage vulnerabilities** across applications and infrastructure
- **Assign ownership** to appropriate teams with AI-assisted confidence scoring
- **Prioritize remediation** based on risk scoring, CVSS, and EPSS metrics
- **Monitor SLA compliance** with configurable targets by severity
- **Generate compliance reports** for frameworks like SOC2, PCI-DSS, HIPAA, and GDPR
- **Conduct threat hunting** with proactive security investigations
- **Respond to incidents** with structured playbooks and workflows

Architecture Summary

VIOE follows the **PCE (Planning-Coordination-Execution) Framework**:



Technology Stack:

Layer	Technology

Frontend Framework	React 18.2
Build Tool	Vite 6.1
Styling	Tailwind CSS 3.4
UI Components	Radix UI + shadcn/ui
Data Fetching	TanStack Query 5.x
Charts	Recharts 2.x
Backend (Optional)	Supabase (PostgreSQL)
Authentication	Mock Auth / Supabase Auth

What the Local Setup Enables

Full Feature Access in Demo Mode:

- Complete UI with all 16 pages functional
- Mock data simulating realistic vulnerability scenarios
- All dashboard visualizations and charts
- Team performance analytics
- Vulnerability triage and prioritization workflows
- Compliance reporting features
- Threat hunting and incident response interfaces

Supabase Mode (Optional):

- Persistent data storage
- Multi-user authentication
- Row-level security
- Real-time updates

What the Local Setup Does NOT Enable

Limitations:

- **No real vulnerability scanners**: Integration with Snyk, SonarQube, etc. requires additional configuration
- **No Jira/Slack integrations**: External service webhooks are mocked
- **No email notifications**: Email functionality requires SMTP configuration
- **No AI inference**: AI features use simulated responses in demo mode
- **No production security**: Local setup bypasses enterprise security controls

---

B. System Requirements

Minimum Hardware Requirements

Component	Minimum	Recommended
CPU	2 cores	4+ cores
RAM	4 GB	8+ GB
Disk Space	2 GB	5+ GB
Display	1280x720	1920x1080

Supported Operating Systems

OS	Version	Status
macOS	12.0 (Monterey) or later	Fully Supported
macOS	11.0 (Big Sur)	Supported
Windows	Windows 10 (1903+)	Fully Supported
Windows	Windows 11	Fully Supported
Linux	Ubuntu 20.04+	Supported (instructions similar to macOS)

Required Software

Software	Version	Purpose
Node.js	18.x or 20.x LTS	JavaScript runtime
npm	9.x or 10.x	Package manager (included with Node.js)
Git	2.30+	Version control
Web Browser	Chrome 100+, Firefox 100+, Safari 16+, Edge 100+	Application access

Optional Software:

Software	Version	Purpose
Supabase CLI	Latest	For Supabase backend mode
VS Code	Latest	Recommended IDE
Docker	20.10+	For containerized deployment

C. Prerequisites Installation

macOS Installation

Step 1: Install Homebrew (if not installed)

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

After installation, follow the instructions to add Homebrew to your PATH:

```
echo 'eval "$(/opt/homebrew/bin/brew shellenv)"' >> ~/.zprofile
eval "$(/opt/homebrew/bin/brew shellenv)"
```

Step 2: Install Node.js

```
brew install node@20
```

Verify installation:

```
node --version    # Should output v20.x.x
npm --version     # Should output 10.x.x
```

Step 3: Install Git

```
brew install git
```

Verify installation:

```
git --version    # Should output git version 2.x.x
```

Step 4: (Optional) Install Supabase CLI

```
brew install supabase/tap/supabase
```

Verify installation:

```
supabase --version
```

Windows Installation

Step 1: Install Package Manager

Option A: Using Winget (Recommended - Built into Windows 10/11)

Winget comes pre-installed on Windows 10 (version 1809+) and Windows 11.

Verify winget is available:

```
winget --version
```

Option B: Using Chocolatey

Open PowerShell as Administrator and run:

```
Set-ExecutionPolicy Bypass -Scope Process -Force
[System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -
bor 3072
iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
```

Close and reopen PowerShell to use Chocolatey.

Step 2: Install Node.js

Using Winget:

```
winget install OpenJS.NodeJS.LTS
```

Using Chocolatey:

```
choco install nodejs-lts -y
```

Close and reopen your terminal, then verify:

```
node --version    # Should output v20.x.x
```

```
npm --version     # Should output 10.x.x
```

Step 3: Install Git

Using Winget:

```
winget install Git.Git
```

Using Chocolatey:

```
choco install git -y
```

Close and reopen your terminal, then verify:

```
git --version    # Should output git version 2.x.x
```

Step 4: (Optional) Install Supabase CLI

```
npm install -g supabase
```

Or using Chocolatey:

```
choco install supabase -y
```

D. Repository Setup

Step 1: Clone the Repository

macOS / Windows (Git Bash or PowerShell):

```
git clone https://github.com/your-org/VIOE.git
cd VIOE
```

Note: Replace `https://github.com/your-org/VIOE.git` with your actual repository URL.

If you received the codebase as a ZIP file:

```
# Extract the ZIP file to your desired location
# Navigate to the extracted folder
cd VIOE
```

Step 2: Verify Repository Structure

After cloning, verify the following structure exists:

```
VIOE/
├── extracted_app/              # Main application directory
│   ├── src/                    # Source code
│   │   ├── api/                # API layer (mock & Supabase adapters)
│   │   ├── components/         # React components
│   │   ├── contexts/           # React contexts (Auth)
│   │   ├── hooks/              # Custom React hooks
```

```
│   │   ├── lib/               # Utility functions
│   │   └── pages/             # Page components
│   ├── .env.example           # Environment template
│   ├── index.html             # HTML entry point
│   ├── package.json           # Dependencies
│   ├── tailwind.config.js     # Tailwind CSS config
│   └── vite.config.js         # Vite build config
├── supabase/                  # Supabase backend files
│   ├── schema.sql             # Database schema
│   ├── seed.sql               # Sample data
│   └── README.md              # Supabase setup guide
└── VIOE_Documentation/        # Additional documentation
```

Directory Structure Explanation

Directory	Purpose
`extracted_app/`	Main React application
`extracted_app/src/api/`	Data layer with mock and Supabase adapters
`extracted_app/src/components/`	Reusable UI components organized by feature
`extracted_app/src/pages/`	Page-level components (16 pages total)
`extracted_app/src/contexts/`	React Context providers (Authentication)
`supabase/`	Database schema and seed data for Supabase mode

E. Environment Configuration

Step 1: Create Environment File

Navigate to the application directory and copy the example environment file:

macOS:

```
cd extracted_app
cp .env.example .env
```

Windows (PowerShell):

```
cd extracted_app
Copy-Item .env.example .env
```

Step 2: Configure Environment Variables

Open the `.env` file in your preferred text editor and configure the following:

Minimal Configuration (Demo Mode)

For quick local testing with mock data, use these settings:

```
# API Configuration
VITE_API_MODE=mock

# Authentication
```

```
VITE_AUTH_PROVIDER=mock

# Feature Flags
VITE_DEMO_MODE=true
VITE_AI_FEATURES_ENABLED=true

# Development
VITE_LOG_LEVEL=debug
VITE_ENABLE_DEVTOOLS=true
VITE_MOCK_DELAY=100
```

Full Configuration (Supabase Mode)

For persistent data with Supabase:

```
# API Configuration
VITE_API_MODE=supabase

# Supabase Configuration (REQUIRED for supabase mode)
VITE_SUPABASE_URL=https://your-project.supabase.co
VITE_SUPABASE_ANON_KEY=your-anon-key-here

# Authentication
VITE_AUTH_PROVIDER=mock

# Feature Flags
VITE_DEMO_MODE=false
VITE_AI_FEATURES_ENABLED=true
VITE_JIRA_INTEGRATION_ENABLED=false
VITE_SLACK_INTEGRATION_ENABLED=false

# External Services (Optional)
VITE_JIRA_BASE_URL=
VITE_JIRA_PROJECT_KEY=
VITE_SLACK_WEBHOOK_URL=

# Monitoring (Optional)
VITE_SENTRY_DSN=
VITE_LOG_LEVEL=info

# Development
VITE_ENABLE_DEVTOOLS=true
```

Environment Variables Reference

Variable	Required	Default	Description
`VITE_API_MODE`	No	`mock`	`mock` for demo mode, `supabase` for production
`VITE_SUPABASE_URL`	If supabase mode	-	Your Supabase project URL
`VITE_SUPABASE_ANON_KEY`	If supabase mode	-	Your Supabase anonymous key
`VITE_AUTH_PROVIDER`	No	`mock`	Authentication provider
`VITE_DEMO_MODE`	No	`true`	Enable demo features

`VITE_AI_FEATURES_ENABLED`	No	`true`	Enable AI-powered features
`VITE_LOG_LEVEL`	No	`info`	Logging verbosity
`VITE_MOCK_DELAY`	No	`100`	Simulated API latency (ms)
`VITE_ENABLE_DEVTOOLS`	No	`true`	Show React Query devtools

Security Notes

IMPORTANT: Never commit these to version control:

- `.env` file (already in `.gitignore`)
- Supabase service role keys
- OAuth client secrets
- API keys for external services

Safe to commit:

- `.env.example` (template without real values)
- Configuration files without secrets

F. Dependency Installation

Step 1: Navigate to Application Directory

```
cd extracted_app
```

Step 2: Install Dependencies

macOS & Windows:

```
npm install
```

This command will:

- Read `package.json` for required packages
- Download all dependencies to `node_modules/`
- Generate `package-lock.json` (if not present)

Expected Output:

```
added 720 packages, and audited 721 packages in 45s

231 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Step 3: Verify Installation

Check that key dependencies are installed:

```
npm list react vite @tanstack/react-query --depth=0
```

Expected Output:

```
base44-app@0.0.0
├── @tanstack/react-query@5.90.20
├── react@18.2.0
└── vite@6.1.0
```

Dependency Overview

Production Dependencies:

Package	Version	Purpose
`react`	^18.2.0	UI framework
`react-router-dom`	^7.2.0	Client-side routing
`@tanstack/react-query`	^5.90.20	Data fetching & caching
`@supabase/supabase-js`	^2.93.3	Supabase client
`recharts`	^2.15.1	Charts and visualizations
`lucide-react`	^0.475.0	Icon library
`tailwind-merge`	^3.0.2	Tailwind class merging
`zod`	^3.24.2	Schema validation

Development Dependencies:

Package	Version	Purpose
`vite`	^6.1.0	Build tool & dev server
`vitest`	^4.0.18	Unit testing
`tailwindcss`	^3.4.17	CSS framework
`eslint`	^9.19.0	Code linting

G. Database & Storage Setup

Option 1: Mock Mode (No Database Required)

If using `VITE_API_MODE=mock` (default), **no database setup is required**.

The application uses an in-memory mock client that provides:

- Pre-populated sample data
- Full CRUD operations (changes persist during session only)
- Simulated API responses

Skip to [Section H: Running the Application]

Option 2: Supabase Mode (Persistent Database)

For persistent data storage, follow these steps to set up Supabase:

Step 1: Create Supabase Project

1. Go to https://supabase.com

2. Sign up or log in to your account
3. Click **"New Project"**
4. Fill in project details:
   - **Name:** `vioe-local` (or your preferred name)
   - **Database Password:** Generate a strong password (save this!)
   - **Region:** Select closest to your location
5. Click **"Create new project"**
6. Wait for the project to provision (1-2 minutes)

Step 2: Get API Credentials

1. In your Supabase dashboard, go to **Settings** → **API**
2. Copy the following values:
   - **Project URL:** `https://xxxx.supabase.co`
   - **anon/public key:** `eyJhbGciOiJIUzI1NiIsInR5cCI6...`
3. Update your `.env` file:

```
VITE_API_MODE=supabase
VITE_SUPABASE_URL=https://your-project.supabase.co
VITE_SUPABASE_ANON_KEY=your-anon-key-here
```

Step 3: Run Database Schema

1. In your Supabase dashboard, go to **SQL Editor**
2. Click **"New Query"**
3. Open the file `supabase/schema.sql` from your local repository
4. Copy the entire contents and paste into the SQL Editor
5. Click **"Run"** (or press Ctrl+Enter / Cmd+Enter)

Expected Result: The query should complete without errors, creating:

- 14 tables (vulnerabilities, teams, assets, etc.)
- Database functions
- Triggers
- Row-level security policies
- Views

Step 4: Load Sample Data (Optional but Recommended)

1. In the SQL Editor, click **"New Query"**
2. Open the file `supabase/seed.sql` from your local repository
3. Copy and paste the contents
4. Click **"Run"**

Expected Result: Sample data is inserted including:

- 4 teams
- 8 assets
- 25+ vulnerabilities
- Remediation tasks
- Compliance reports

Step 5: Verify Database Setup

Run this query in the SQL Editor to verify:

```
SELECT
    (SELECT COUNT(*) FROM teams) as teams,
    (SELECT COUNT(*) FROM assets) as assets,
    (SELECT COUNT(*) FROM vulnerabilities) as vulnerabilities;
```

Expected Output:

teams	assets	vulnerabilities
4	8	25+

H. Running the Application

Step 1: Start the Development Server

From the `extracted_app` directory:

macOS:

```
npm run dev
```

Windows (PowerShell or Command Prompt):

```
npm run dev
```

Expected Output:

```
 VITE v6.1.0  ready in 239 ms

 ➜  Local:   http://localhost:5173/
 ➜  Network: use --host to expose
 ➜  press h + enter to show help
```

Note: If port 5173 is in use, Vite will automatically try 5174, 5175, etc.

Step 2: Access the Application

1. Open your web browser
2. Navigate to: **http://localhost:5173**
3. You should see the VIOE login page

Step 3: Log In

Demo Mode Credentials:

Email	Password	Role
`admin@vioe.demo`	`demo`	Admin (full access)
`analyst@vioe.demo`	`demo`	Analyst
`manager@vioe.demo`	`demo`	Manager
`viewer@vioe.demo`	`demo`	Viewer (read-only)

Or use any email with password `demo` to log in with default analyst role.

Step 4: Verify Application is Running

After login, you should see the Dashboard with:

- Vulnerability summary metrics
- Severity distribution charts
- Team workload overview
- Recent activity

Available NPM Scripts

Command	Description
`npm run dev`	Start development server with hot reload
`npm run build`	Build production bundle
`npm run preview`	Preview production build locally
`npm run lint`	Run ESLint code linting
`npm run test`	Run unit tests with Vitest
`npm run test:coverage`	Run tests with coverage report

Stopping the Server

Press `Ctrl+C` in the terminal to stop the development server.

I. Feature Validation & Testing

Test 1: Dashboard & Metrics

URL: http://localhost:5173/Dashboard

Steps:

1. Log in with any demo credentials
2. Verify the Dashboard loads

Expected Results:

- Total vulnerabilities count displayed
- Severity breakdown (Critical, High, Medium, Low)
- Charts render correctly
- "Needs Review" and "SLA Breached" counts visible

Test 2: Vulnerability Management

URL: http://localhost:5173/Vulnerabilities

Steps:

1. Navigate to Vulnerabilities page
2. Click on any vulnerability row
3. View vulnerability details

Expected Results:

- List of vulnerabilities displays
- Severity badges color-coded correctly

- Filtering by severity works
- Clicking a row shows detailed view

Sample Action - Triage a Vulnerability:

1. Click "Triage" button on any vulnerability
2. Select a team from the dropdown
3. Set confidence level
4. Click "Assign"

Expected: Vulnerability updates with new assignment

Test 3: Team Performance

URL: http://localhost:5173/Teams

Steps:

1. Navigate to Teams page
2. Click "View Performance Insights" on any team

Expected Results:

- Team cards display with member counts
- Performance metrics load
- AI insights panel shows recommendations

Test 4: Remediation Tasks

URL: http://localhost:5173/RemediationTasks

Steps:

1. Navigate to Remediation Tasks
2. Click "Create Task" button
3. Fill in task details
4. Submit the form

Expected Results:

- Task list displays existing tasks
- Create dialog opens
- New task appears in the list after creation
- Task status can be updated

Test 5: Threat Hunting

URL: http://localhost:5173/ThreatHunting

Steps:

1. Navigate to Threat Hunting page
2. View threat alerts
3. Click "New Hunt" to create a hunting session

Expected Results:

- Threat alerts display with severity indicators
- Alert cards show confidence scores
- Hunting sessions list with progress indicators
- "Investigate" buttons functional

Test 6: Compliance Reports

URL: http://localhost:5173/ComplianceReports

Steps:

1. Navigate to Compliance Reports
2. Select a framework (SOC2, PCI-DSS, etc.)
3. View compliance score

Expected Results:

- Framework selector works
- Compliance score displayed
- Control status breakdown visible
- Findings list populated

Test 7: Incident Response

URL: http://localhost:5173/IncidentResponse

Steps:

1. Navigate to Incident Response
2. View incident timeline
3. Check playbook section

Expected Results:

- Incident cards display
- Status indicators work
- Timeline shows events
- Playbook steps visible

Test 8: Asset Management

URL: http://localhost:5173/Assets

Steps:

1. Navigate to Assets page
2. Click "Add Asset" button
3. Fill in asset details

Expected Results:

- Asset inventory displays
- Criticality levels shown
- Environment tags visible
- New asset can be created

Running Automated Tests

```
# Run all tests
npm run test

# Run tests with coverage
npm run test:coverage
```

```
# Run tests in watch mode
npm run test -- --watch
```

J. Common Errors & Troubleshooting

Error: "Port 5173 is in use"

Cause: Another application is using the default Vite port.

Solution:

Vite automatically tries the next available port. Check the terminal output for the actual URL.

Or manually specify a port:

```
npm run dev -- --port 3000
```

Error: "Module not found" or "Cannot resolve"

Cause: Dependencies not installed or corrupted.

Solution:

```
# Remove existing modules and reinstall
rm -rf node_modules package-lock.json   # macOS
# OR
Remove-Item -Recurse -Force node_modules, package-lock.json   # Windows

# Reinstall
npm install
```

Error: "VITE_SUPABASE_URL is not defined"

Cause: Environment variables not loaded.

Solution:

1. Verify `.env` file exists in `extracted_app/` directory
2. Ensure variables are prefixed with `VITE_`
3. Restart the development server after changing `.env`

Error: "Failed to fetch" or "Network Error"

Cause: API mode mismatch or Supabase not configured.

Solution:

1. Check `VITE_API_MODE` in `.env`:
   - Use `mock` for local development without Supabase
   - Use `supabase` only if Supabase is configured
2. If using Supabase, verify:
   - `VITE_SUPABASE_URL` is correct
   - `VITE_SUPABASE_ANON_KEY` is correct

- Database schema has been applied

Error: "Cannot find module 'react'"

Cause: Node modules not installed.

Solution:

```
cd extracted_app
npm install
```

---

macOS: "Permission denied" when running npm

Solution:

```
# Fix npm permissions
sudo chown -R $(whoami) ~/.npm
sudo chown -R $(whoami) /usr/local/lib/node_modules
```

Or use a Node version manager like `nvm`.

Windows: "Execution Policy" Error

Cause: PowerShell script execution is restricted.

Solution:

Open PowerShell as Administrator:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

Windows: "CRLF vs LF" Git Warnings

Cause: Line ending differences between Windows and Unix.

Solution:

```
git config --global core.autocrlf true
```

Blank Page After Login

Cause: JavaScript error in console.

Solution:

1. Open browser Developer Tools (F12)
2. Check Console tab for errors
3. Common fixes:
   - Clear browser cache
   - Hard refresh (Ctrl+Shift+R)
   - Check `.env` configuration

Supabase: "Row Level Security" Errors

Cause: RLS policies blocking access.

Solution:

1. Ensure you're authenticated
2. Check RLS policies in Supabase dashboard
3. For development, you can temporarily disable RLS:

```
-- CAUTION: Only for development!
ALTER TABLE vulnerabilities DISABLE ROW LEVEL SECURITY;
```

K. Optional Advanced Setup

Dockerized Setup

Assumption: Docker support is not included in the current codebase but can be added.

Create a `Dockerfile` in the `extracted_app` directory:

```
# Build stage
FROM node:20-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

# Production stage
FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Create `nginx.conf` :

```
server {
    listen 80;
    root /usr/share/nginx/html;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }
}
```

Build and run:

```
docker build -t vioe:latest .
docker run -p 8080:80 vioe:latest
```

Access at: http://localhost:8080

Mock Scanner Integration

To simulate vulnerability scanner imports:

1. Navigate to http://localhost:5173/ImportVulnerabilities
2. Use the JSON import feature
3. Sample import format:

```
{
  "scanner": "manual",
  "findings": [
    {
      "cve_id": "CVE-2024-1234",
      "title": "Test Vulnerability",
      "severity": "high",
      "cvss_score": 7.5,
      "description": "Test description",
      "affected_component": "test-component"
    }
  ]
}
```

Test Data Generation

The mock client includes randomized data generators. To refresh mock data:

1. Open browser Developer Tools
2. Go to Application → Local Storage
3. Clear `vioe_user` and reload the page

Or programmatically:

```
// In browser console
localStorage.clear();
location.reload();
```

Running Background Workers Independently

Assumption: Background workers are not implemented in the current frontend-only architecture.

For future backend implementations, workers would be run separately:

```
# Example (not currently implemented)
npm run worker:triage
npm run worker:sla-checker
```

Enabling Debug Mode

For verbose logging:

1. Set in `.env` :

```
VITE_LOG_LEVEL=debug
VITE_ENABLE_DEVTOOLS=true
```

2. Open React Query DevTools (floating button in bottom-right)
3. View query cache and mutations

Production Build

To create a production-optimized build:

```
npm run build
```

Output will be in `extracted_app/dist/`. Preview locally:

```
npm run preview
```

Appendix: Quick Reference

Quick Start Commands

```
# Clone and setup
git clone <repository-url>
cd VIOE/extracted_app
cp .env.example .env
npm install
npm run dev
```

Demo Credentials

Email	Password	Role
admin@vioe.demo	demo	Admin
analyst@vioe.demo	demo	Analyst
(any email)	demo	Default Analyst

Key URLs

Page	URL
Dashboard	http://localhost:5173/Dashboard
Vulnerabilities	http://localhost:5173/Vulnerabilities
Teams	http://localhost:5173/Teams
Remediation	http://localhost:5173/RemediationTasks
Compliance	http://localhost:5173/ComplianceReports
Threat Hunting	http://localhost:5173/ThreatHunting
Settings	http://localhost:5173/Settings

Support

For issues and feature requests:

- Repository Issues: [GitHub Issues Link]
- Documentation: `/VIOE_Documentation/` folder

Document End

Generated: February 1, 2026