User Guide – Secura Sphere

Table of Contents

---

1. Introduction

This user guide explains how to use the SecuraSphere web application created as part of a semester-long software engineering project. The application uses a **React-based interface** for the client-side experience, a **Flask backend API** for server-side logic, and **JSON files** for storing supplemental data.

The purpose of this guide is to help users:

- Navigate the interface
- Understand the functionality of each feature
- Interact with both frontend tools and backend-powered features
- Resolve common issues

---

2. System Requirements

User Requirements

- A modern web browser: Chrome, Firefox, Safari, or Edge
- Internet connection (if hosted online)
- GitHub profile

Developer Requirements (for local use)

- **Node.js** (latest LTS version recommended)
- **Python 3.x**
- **Pip** for installing Python dependencies
- **Git** for cloning the repository

---

3. Getting Started

The application can be run locally. Eventually we hope to actually deploy it for online use.

Running Locally (Developer Mode)

1. Clone the GitHub Repository

```
git clone <https://github.com/rzop/seniorprojectsec>
cd <seniorprojectsec>
```

2. Install Frontend Dependencies (React)

```
cd frontend
npm install
```

3. Install Backend Dependencies (Flask)

```
cd ../backend
pip install -r requirements.txt
```

4. Run the Flask Backend

```
python start_backend.py
```

The backend will start on a local server (typically http://127.0.0.1:5000).

5. Run the React Frontend

```
cd ..
npm start
```

This launches the app in your browser at http://localhost:3000.

---

4. Accessing the Application

Once running, the homepage will load in the browser. Users are greeted with the main dashboard, which includes navigation buttons, links to tools, and link to our upgraded test UI which is fully functional.

---

5. Application Navigation Overview

The interface consists of:

1. Top Information Section

- Title of our website
- Brief description of our platform

2. Test UI button

- Allows the user to access our upgraded UI design with the same functionality of our current app.

3. Main Content Area

Displays the three module buttons which takes the user to whichever tool, page, or data visualization the user selected.

---

6. Frequently Asked Questions (FAQs)

This section addresses common questions users may have while using the application.

1. Why is the app not loading in my browser?

This usually occurs when the backend is not running. Ensure the Flask server is active and the React app is started with `npm start`.

2. Why am I getting a “Failed to Fetch” error?

This happens when the frontend cannot communicate with the backend. Check that Flask is running on the correct port and verify CORS settings if applicable.

3. Can I modify the JSON files?

Yes, but changes should follow valid JSON formatting. Incorrect formatting will cause backend parsing errors.

4. Why are some pages blank or not updating?

React components depend on state updates. Check the console for errors and ensure the backend response contains the expected data.

5. Do I need to refresh the app manually?

Most interactions update automatically thanks to Reacts state management. Refresh only if the backend was recently restarted.

6. Where do I report bugs or issues?

Use the project’s GitHub Issues page or contact the development team directly.

---

7. Troubleshooting & Common Issues

Problem: App won’t load (blank page)

Solutions: - Ensure Flask backend is running
- Ensure `npm start` is running in main directory
- Refresh the browser

Problem: “Failed to Fetch” error

Solutions: - Backend URL may be incorrect
- CORS settings may need adjustment
- Check that the backend is running on the expected port

Problem: JSON not loading

Solutions: - Validate JSON formatting
- Check file path references
- Ensure backend has read permissions

---

8. Workflows

This section describes the major workflows within the application, including user interaction flows, system processing flows, and development workflows used during the project.

8.1 User Workflows

User Workflow: Accessing and Using a Tool in the React Interface

1. User opens the homepage of the React application.
2. User selects a tool from the main content section.
3. React updates the displayed component based on user selection.
4. If the tool requires backend data, React sends a request to the Flask API.
5. User enters any required input into the UI.
6. User submits the input (button click).
7. The processed results return from the backend and display in the React interface.

User Workflow: Viewing JSON-Driven Content

1. User navigates to a page that loads predefined content (intermediate pages between homepage and tools).
2. React requests the corresponding JSON file or data endpoint.
3. JSON data is loaded and used to populate dropdowns, lists, or UI fields.
4. User interacts with the displayed content.

8.2 System Workflows

System Workflow: Request → Processing → Response

1. A React component triggers a GET or POST request.
2. Flask receives the request at a defined endpoint.
3. Flask validates the request and loads necessary JSON files.
4. Backend logic processes or transforms the data.
5. Flask returns a JSON response.
6. React updates component state using the returned data.
7. UI re-renders automatically.

System Workflow: JSON Data Handling

1. Flask opens the JSON file using a file path specified in the backend.
2. JSON is parsed into Python objects.
3. Data is filtered, searched, or modified according to logic.
4. Modified or selected data is sent back to React.

5. React renders UI components based on the new data.

8.3 Development & Team Workflows

Git Workflow (Version Control)

1. Repository is created on GitHub.
2. Team members clone the repo locally.
3. Developers create feature branches for isolated work.
4. Code changes are committed and pushed to GitHub.
5. Pull Requests are created for merging into the main branch.
6. Team reviews code for correctness and style.
7. PR is approved and merged.
8. Main branch updates trigger deployment or local testing.

9. Glossary

API – A connection between frontend and backend allowing data exchange.
Component – A reusable piece of the React interface.
Endpoint – A backend URL that handles requests.
JSON – A lightweight data format used for storing structured information.
State – React's storage for dynamic values.

---

10. Support Information

For questions, technical issues, or further documentation: **- Visit Our GitHub Repository linked above.**
