CyberGuard AI Analyst: Study Documentation

1. Project Overview

CyberGuard AI Analyst is a web-based tool that leverages Google Gemini AI to analyze suspicious text (emails, logs, URLs) for potential cyber threats. It extracts indicators of compromise (IoCs), assesses severity, and provides actionable recommendations—all in a simple, user-friendly interface.

---

2. Key Features

- **Paste suspicious text** (email, log, URL) for instant AI-powered threat analysis.
- **AI-driven threat report**: Gemini LLM analyzes the input and simulated tool outputs.
- **Severity assessment**: Clear, color-coded alert (green/yellow/red) based on AI's judgment.
- **IoC extraction**: URLs, IPs, and emails are automatically identified.
- **Mitigation suggestions**: Actionable steps for the user or IT team.
- **History modal**: Review or clear past analyses.
- **Chatbot**: Ask basic security questions.
- **Modern, responsive UI**: Built with Flask and Tailwind CSS.

---

3. Technology Stack

- **Backend**: Python, Flask, Google Gemini API, python-dotenv, requests
- **Frontend**: HTML, Tailwind CSS, JavaScript
- **Other**: requirements.txt for dependencies, .env for secrets

---

4. How It Works

4.1 User Flow

1. User pastes suspicious text and clicks "Analyze Threat".
2. The backend extracts IoCs (URLs, IPs, emails) using regex.
3. Simulated threat lookups are performed for each IoC.
4. All data is sent to Gemini LLM with a detailed prompt.
5. Gemini returns a structured threat report (summary, findings, severity, actions).
6. The frontend displays the report and a color-coded alert based on severity.
7. User can view history, clear history, or use the chatbot.

4.2 Severity Logic

- The backend parses the AI's report for severity (Critical, High, Medium, Low, Informational).
- The frontend alert color and message always match the AI's assigned severity.

---

5. Code Structure

- `app.py`: Flask app, routes, API logic, severity extraction, history, chatbot.
- `cyber_agent_core.py`: IoC extraction, simulated threat lookup, Gemini prompt/orchestration.
- `templates/index.html`: Main UI, alert logic, history modal, chatbot, JS.
- `requirements.txt`: Python dependencies.
- `.env`: API keys and secrets (not in version control).
- `test_green_scenarios.py` / `test_red_scenarios.py`: Automated tests for safe/threat scenarios.

---

6. Prompt Engineering

- The Gemini prompt is carefully crafted to:
  - Flag classic phishing (password reset, account compromised, urgent action with links) as HIGH/CRITICAL.
  - Flag double extensions (e.g., .pdf.exe) as HIGH/CRITICAL.
  - Treat standard password change confirmations (no links, no urgent action) as safe.
  - Require a structured report: summary, findings, severity, actions, disclaimer.

---

7. Security & Best Practices

- API keys/secrets are stored in `.env` (never in code or git).
- Input is validated and length-limited (max 5000 chars).
- User-friendly error messages for API failures.
- No sensitive data is logged in production.
- All user-facing messages are clear and consistent.

---

8. Extensibility & Future Ideas

- Integrate real threat intelligence APIs (VirusTotal, AbuseIPDB, etc.).
- Add user authentication and saved analysis history.
- Support for file uploads (e.g., log files, email .eml files).
- More advanced log parsing and anomaly detection.
- Deploy to cloud (e.g., Google Cloud Run, Azure App Service).

---

9. How to Run

1. Install dependencies:

```
pip install -r requirements.txt
```

2. Set up `.env` with your Gemini API key and Flask secret key.
3. Start the app:

```
python app.py
```

4. Open your browser to `http://localhost:5000`

---

10. Testing

- Run green (safe) scenario tests:

```
python test_green_scenarios.py
```

- Run red (threat) scenario tests:

```
python test_red_scenarios.py
```

---

11. Learning Outcomes

- Practical experience with LLM prompt engineering for security.
- Building robust, user-friendly AI web apps.
- Secure API key management and input validation.
- Automated testing of AI-driven security logic.

---

12. References

- Google Gemini API Docs
- Flask Documentation
- Tailwind CSS
- OWASP Phishing Guidance

---

13. Deployment to Render.com (Free Hosting)

- The project can be deployed for free using Render.com.
- Steps followed:
    1. Pushed the latest code and documentation to GitHub.
    2. Created a new Web Service on Render.com, connected to the GitHub repo.
    3. Set the **Build Command** to:

```
pip install -r requirements.txt
```

    4. Set the **Start Command** to:

```
python app.py
```

5. Added environment variables (`GEMINI_API_KEY`, `FLASK_SECRET_KEY`) in the Render dashboard (copied from `.env`).
6. Deployed and tested the app at the public Render URL.

- Troubleshooting:
  - If you see a build error about `pip install -r [requirements.txt](...)`, remove any brackets or Markdown formatting and use the plain filename as above.
- The app works on Render's free tier, with auto-sleep and public access for demos.

This documentation is up to date as of May 27, 2025, including deployment and troubleshooting steps for Render.com.