Subteam 21.1

Part 1 – Planning

1. Identifying the Problem/Improvement

Problem:
Currently, *a12n-server* provides a lightweight authentication and authorization solution but does not proactively monitor user login behavior for potential anomalies (e.g., multiple failed logins from new countries, unusual login times, or high-risk IP addresses). Admins have no automated way to detect or be warned of suspicious login attempts—relying purely on logs can be slow and prone to human error or oversight.

Improvement via AI:
We propose adding an AI-driven anomaly detection system that flags suspicious login attempts in real time or near real time. A simple example user story:

> "As an admin, I want to be alerted when a12n-server sees an unusual login pattern for a particular user, so that I can react quickly and reduce the likelihood of a security breach."

2. How the AI Component Enhances the Project

1. Automation: Rather than requiring an admin to manually sift through logs, an AI model can continuously monitor relevant login metrics—IP addresses, device fingerprints, geolocation, timestamps—and highlight "unusual" events.
2. Better Security & Decision Support: Admins gain immediate insights into high-risk login activity, decreasing the time between suspicious behavior and a security response.
3. Adaptive: Over time, the model can learn typical login patterns for individual users and reduce false positives by recognizing "legitimate anomalies" (e.g., someone traveling once in a while).

3. Criteria for Choosing a Solution

We must ensure the solution:

1. Integrates with Existing Tech: Minimal friction with the Node.js/TypeScript stack and synergy with the existing endpoints in a12n-server.
2. Performance: The approach should ideally have low latency or be easily offloaded to a separate process or queue for analysis.
3. Feasibility: We should have or be able to collect the needed data—login timestamps, IP addresses, geolocation, or device info—to train or run an anomaly detector.
4. Maintainability: Clear guidelines on how the model is updated or re-trained, plus easy error and performance monitoring.

4. Potential AI Models, Frameworks, and Libraries

Local Outlier Factor (LOF), Isolation Forest, or Autoencoders:

- These are classic anomaly-detection algorithms.
- Could be implemented in a Node/TypeScript environment via a library like TensorFlow.js (for an autoencoder) or via Python microservices that your Node server calls asynchronously.

Third-Party Services (e.g., ML services on AWS, GCP, or Azure):

- Managed anomaly detection solutions that can ingest streaming data.
- Training and configuration is modularized and separated, through services like GCP and Vertex AI, we can simply call an API to use
- Downside: requires external cloud configuration and cost overhead, but it offloads the training complexity.

In-Process vs. External Service

- In-process: Use a library such as brain.js or scikit.js (the JS port of scikit-learn).
- External: Host a minimal Python service with scikit-learn or PyTorch. Then a12n-server or the Admin UI just calls an HTTP endpoint to evaluate suspiciousness.

Comparing Alternatives:

Approach	Ease of Integration	Performance	Complexity	Recommended Use
Local Outlier Factor	Medium	Real-time feasible	Medium (distance-based)	Good if you want straightforward anomaly scores without heavy training.
Isolation Forest	Medium	Real-time feasible	Medium (tree-based)	Often used for malicious request detection in security contexts.
Autoencoder	Higher	Potentially GPU-intensive	Higher (requires neural net)	Best for advanced anomaly detection if you have large volumes of data.
Third-Party ML	High (integration overhead)	Scalable but depends on network calls	Low (outsourced training)	Great if you want minimal custom code but can pay for external services.

Given that a12n-server is meant to be "lightweight," a typical approach is using something like Isolation Forest or Local Outlier Factor in an external microservice, or simply integrate scikit.js for small scale.

5. Dataset Requirements

Data We Need:

- Past login attempts and status (success/failure).
- IP addresses, geolocation data (if available).
- User agent or device metadata.
- Timestamp (time of day, day of week).

Data Collection Approach:

- a12n-server can log login attempts to the existing database (Postgres/MySQL/SQLite).
- We would add columns for user agent, IP, or geolocation if not already present.

Data Preprocessing:

- *IP Address → Geolocation* (via a GeoIP library).
- Normalizing or categorizing times (e.g., typical vs. atypical hours).
- Possibly one-hot encoding for device type or country of origin.
- Re-label known suspicious attempts if we have historical examples, though that may be scarce initially.

Bias Mitigation & Ethical Considerations:

- IP origin can correlate with specific countries—avoid marking certain geolocations as always suspicious, or you risk bias.
- Keep user privacy in mind; store only minimal metadata (hash IP or trim user agent) to respect GDPR or other privacy regulations.

6. Justification & Alignment with Project Goals

- Goal: a12n-server aims to be a simple, secure alternative to heavier solutions like Keycloak.
- Why AI is needed: A purely rules-based approach (e.g., "block IP after 5 attempts") is easy to bypass. AI anomaly detection adapts to each user's patterns and can reduce false positives.
- Alignment: Security is paramount in authentication. AI anomaly detection directly strengthens the solution's security offering, differentiating it from "lighter" but less proactive servers.

Part 2 – Impact Analysis

1. How the Product/Process Improves with AI

- Before: Admins rely on manual log inspections, might only notice suspicious patterns late (if at all).
- After:
    1. Real-Time Alerts: As soon as a user's login deviates significantly from their normal pattern, a12n-server can log an "anomaly event" or push a notification to the Admin UI.
    2. Adaptive Security: Over time, the model adjusts to each user's typical behavior, improving detection accuracy.
    3. Proactive Response: Admins can lock or force MFA re-enrollment for suspect accounts *before* a potential breach escalates.
    4. Have

2. Testing the Improvement

1. Offline Accuracy Evaluation
    - Gather historical login data (or synthetic test data) with some known malicious attempts inserted.
    - Split data into train/test sets, measure *precision*, *recall*, and *F1 score* for flagged anomalies.
2. Unit Tests
    - Ensure the anomaly detection module receives the correct fields (IP, timestamp, user ID, etc.) and returns a consistent anomaly score.
3. Integration Tests
    - Spin up the a12n-server with a mock AI microservice. Simulate login attempts from a variety of IP addresses and times.
    - Verify the server's "anomaly detection pipeline" triggers the correct alerts in the Admin UI.
4. A/B Testing
    - If it's a large deployment, you could enable the AI detection for half the user base to compare security-related metrics (e.g., incident response time, number of compromised accounts).

3. Trade-Offs

- Resource Consumption:
    - Training or running an anomaly model will use CPU or GPU cycles. *Possible Mitigation*: Offload the model to a separate process or service, so the main a12n-server remains lightweight.
- Latency:
    - Real-time checks could add overhead to the login flow. *Possible Mitigation*: Evaluate anomalies asynchronously; if an anomaly is detected, then challenge the user with additional MFA steps or log it for the admin in near real time.
- False Positives:
    - Admins might get alerted too often, leading to alert fatigue. *Solution*: Tweak thresholds or incorporate additional context (e.g., user's travel frequency, known new device sign-ins).

- Maintenance:
    - Over time, the model might need retraining or hyperparameter tuning to remain accurate.
    - The team must maintain a stable data pipeline to keep user login logs consistent and clean. Feed data continually to adapt to new circumstances (Verified location changes, blocking/ignoring suspicious IP addresses)
    - Create agents that monitor and moderate the server so less interference and maintenance is needed
- Scalability vs Cost
    - A lightweight in-process model minimizes infrastructure costs but may not scale well for high traffic. An external ML service scales better but incurs cloud costs. *Possible Mitigation:* Use a hybrid approach—run a simple model locally and escalate complex cases to a cloud service when necessary.