

Security Architecture & Threat Model

The Platform

TheSecurityPiece is a cybersecurity education platform targeting entry-level to junior professionals who have certifications, degrees, or a genuine interest to join the field.

I designed the platform with security as a core principle and started by prioritizing defense-in-depth. Every design decision balanced security requirements against user experience because to remove friction that could drive users away.

Threat Surface Analysis

I identified attack vectors by thinking about what could go wrong with a public-facing platform handling user accounts, admin privileges, and user-generated content and landed on:

- Unauthorized privilege escalation from standard users to admin access
- Injection attacks through user-generated content like comments, challenge submissions, search boxes, and contact forms.
- Account compromise attempts through credential stuffing or session hijacking
- Platform feature abuse like point farming or resource exhaustion
- Automated scraping attempting to extract job data or article content

Understanding these threats shaped how I implemented security controls. Each threat required specific countermeasures layered throughout the architecture.

Defense-in-Depth Strategy

I implemented layered security controls so no single failure compromises the entire platform. Backend validation happens independently of frontend checks. The defense layers include:

- Backend access validation independent of frontend restrictions
- Input sanitization across all user touchpoints preventing injection attacks
- Rate limiting on API endpoints stopping resource abuse
- Secrets management keeping credentials out of source code
- Principle of least privilege for system components
- Firebase security rules enforcing data access controls at database level

The layering makes it difficult for attackers as they would need to bypass multiple controls to cause damage. If someone circumvents frontend validation, backend checks catch the attempt. If they manipulate client-side state, database rules prevent unauthorized data access.

Authentication & Access Control

I chose OAuth 2.0 through Google authentication to eliminate password management vulnerabilities entirely. This decision reduces my attack surface while improving user onboarding since most users already have Google accounts.

- Admin claim assignment happens server-side through Firebase Admin SDK, preventing users from granting themselves admin privileges.
- Frontend checks improve experience by hiding admin features from standard users, while backend validation actually enforces access restrictions.

User-Generated Content Protection

User comments represent the biggest injection attack surface since people can submit arbitrary text. I sanitize all inputs before storage and rendering to prevent XSS attacks, SQL injection attempts, and malicious script execution. The content protection includes:

- Input sanitization removing dangerous HTML and script tags
- Markdown parsing with safe rendering preventing code execution
- Content Security Policy headers blocking unauthorized script sources
- Rate limiting on comment submission preventing spam

The sanitization happens both at input time and render time as a defense-in-depth measure. If sanitization fails at one layer, the other layer catches malicious content before it affects users.

Data Protection Approach

PII collection was minimized because data not collected can't be breached. The platform only stores what's necessary for functionality:

- Google OAuth provides name and email without storing passwords
- User activity tracking limited to points system and article interactions
- No payment data since the platform uses a point-reward system model
- Job data stored without user association
- Firebase security rules preventing unauthorized data access

The data minimization principle reduces both regulatory burden and breach impact. If the database gets compromised, attackers gain limited personal information.