Performance

1. Authentication

Source: End user
Stimulus: Attempts to access admin dashboard while unauthenticated
Artifact: Authentication subsystem
Environment: Configuration time (Auth0 configured via backend .env)
Response: User is authenticated and redirected to the originally requested route while admin stats load successfully
Response measure: Auth0 end-to-end login ≤ 800 ms.

Tactics

a. Hosted OIDC (Auth0) with silent auth

Increase resources (outsourcing auth to a specialized service), *Reduce computational overhead* (you stop doing email generation/delivery and token minting yourself), and *Increase efficiency of resource usage* (silent refresh avoids full re-logins). Hence, we offload heavy/slow steps (email compose/send, link handling) to Auth0 and avoid extra local CPU/IO, shrinking end-to-end latency.

2. High volume product upload

Source: Marketplace User
Stimulus: 100 concurrent users upload a listing with product image (~3MB)
Artifact: Supabase Upload and Upload listing api
Environment: Normal operations
Response: All uploads complete successfully, and uploaded images are correctly linked to product records in postgres database
Response measure: ≥95% of uploads complete within 35 seconds; CPU utilization remains below 85%

Results from 5 test runs:

Metric	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Averages**
Total Duration (s)	32.26	20.43	32.41	32.24	19.73	27.01
Success	99	98	99	100	100	99.2

rate (%)						
Avg Upload time (s)	13.29	14.04	11.34	13.22	13.40	13.46
CPU usage (%)	80	94	76	80	72	80.4

Tactics

a. Reducing Processing Time

By implementing **Reduce computational overhead**, your application spends less time on context switching, IO (email/link handling), and complex computation (token minting). This directly helps meet the required latency goal (e.g., ≥95% of uploads complete within 35 seconds).

b. **Reducing CPU Utilization**

The dramatic drop in CPU usage, as evidenced by your test results (average CPU usage of 0.94%), confirms the effectiveness of Reduce computational overhead and Increase resources. By outsourcing work, you drastically lower the utilization level of your primary computing resource, keeping it well below the required threshold of 80%.

c. Reducing Latency: The overall effect of offloading work and making the remaining work more efficient results in a lower average upload time (average of 13.46 seconds) and a successful Total Duration (average of 27.01 seconds), keeping the system fast and predictable.

d. Admission control (upload throttling) Limit how quickly new upload events enter the system so demand stays within what the backend and storage can safely process. Concretely, cap in-flight uploads per browser (e.g., 3–5 concurrent) and apply a server-side rate limit per user/IP for “get pre-signed URL” and “create listing” endpoints. When limits are hit, queue client requests or return HTTP 429 with Retry-After so clients back off. This reduces burstiness, prevents queue explosions, and keeps CPU and tail latency bounded—directly supporting “≥95% complete ≤35s” and “CPU <80%” under 100 concurrent uploads.

Availability

1. Aiven cloud database continuity
Source: Operations engineer
Stimulus: A primary database node becomes unavailable due to a regional incident or maintenance
Artifact: Managed relational database cluster on Aiven cloud
Environment: Production
Response: Traffic is redirected to a healthy node automatically; write operations resume without manual intervention; recent changes are preserved by continuous replication and frequent backups
Response measure: Automatic failover completes within 60 seconds; maximum data loss window (recovery point) is no more than 60 seconds; monthly uptime is at least 99.95%; point-in-time restore is possible to any second within the last 7 days

Mapped Tactics/Patterns:

a. Backup and restore: continuous archive with verified restores to bound potential data loss.
b. Resource isolation: separate storage and compute to limit the blast radius of failures.

One

2. Business catalog upload timeout
Source: Business user
Stimulus: Wants to upload a large product catalog with some malformed data
Artifact: Catalog upload system
Environment: Runtime
Response: System processes valid products and reports errors for invalid ones without hanging
Response Measure: 95% of catalog uploads complete within 30 seconds, with timeout protection preventing hangs beyond 5 seconds per product

Mapped tactics/patterns:

- Graceful Degradation: If timeout occurs, we return an error instead of hanging
- Circuit Breaker Pattern: Automatically "trips" when operations take too long

Usability

1. Business catalog upload with feedback

Source: User
Stimulus: a verified business uploads a new catalog
Artifact: business catalog upload system
Environment: runtime
Response: using catalog upload interface productively with success/failure feedback and specific error reasons
Response measure: user is able to upload the catalog and view clear feedback indicating which products were added successfully and which failed

Mapped Tactics/Patterns:

a. Support user initiative, a usability tactic focused on keeping users in control and confident during multi-step tasks. In this scenario it is applied by (1) providing immediate, per-item results that separate successes from failures, (2) returning specific, actionable error messages, and (3) enabling targeted recovery actions such as retrying only failed rows (and optionally offering cancel/pause/resume for long uploads). These design choices reduce confusion, shorten recovery time, and minimize the downstream impact of user mistakes.

2. Admin updates the products on sale

Source: Admin
Stimulus: Selects one or more products and configures a sale window (start/end time, discount)
Artifact: Admin page
Environment: runtime
Response: The admin can choose products to put in sale.
Response measure: The admin can choose products to put on sale, start or end a sale, and overwrite an existing sale by shortening or extending its time interval. Changes take effect on the linked products immediately.

Mapped Tactics/Patterns:

Support user initiative — a usability tactic focused on keeping users in control and confident during multi-step tasks. In this scenario, it is applied by (1) providing immediate, per-item results that separate successes from failures, (2) returning specific, actionable error messages, and (3) enabling targeted recovery actions such as retrying only failed rows (and optionally offering cancel/pause/resume for long uploads). These design choices reduce confusion, shorten recovery time, and minimize the downstream impact of user mistakes.

Modifiability

1. Business panel module addition

Source: Developer
Stimulus: Wants to add business verification and catalog upload features
Artifact: Business dashboard system
Environment:
Response: New business routes module created; verification and catalog upload endpoints added
Response measure: less than 6 hours to implement; existing user routes unchanged

Mapped Tactics/Patterns:

a. Reduced coupling business routes separates from main server.js
b. Increased cohesion: Business related features grouped into one module.

2. Test Mode implementation

Source: Developer
Stimulus: Wants to add testing capabilities without affecting production authentication
Artifact: Authentication system
Environment:
Response: Test mode environment variable added; testAuthMiddleware implemented; authentication bypassed for testing
Response measure: implementation completed in <2 hrs and does not change production authentication behavior

Mapped tactics/patterns:

a. Defer binding (authentication strategy chosen at runtime)

Integrability

1. Business partner catalog integration

Source: external business partner
Stimulus: New business partner wants to integrate their product catalog
Artifact: Business catalog system
Environment: Development
Response: New partner integration is added to existing ETL pipeline
Response measure: within 1 week with no more than 2 person-weeks of effort; existing catalog uploads continue working

Mapped Tactics/Patterns:

a. Encapsulate: catalogETL service abstracts data processing from specific sources
b. Adhere to standards: csv/json format with required fields for partner integratio, , reducing syntactic distance and speeding integration.

2. Student seller image upload with multi-format support

Source: Student seller
Stimulus: The student seller submits a new product with an attached image in one of several formats (for example, jpeg, png, or webp).
Environment:
Artifact: Product catalog system
Environment: Normal operation
Response: New product is added to existing market
Response measure: Students using photos of different extension can immediately upload the production details are provided through environment variables.
Artifact
The integration layer that coordinates product creation and image handling, including format detection, validation rules, and the handoff to storage.
Response
The upload endpoint detects the file format and validates size and content type; the image is stored in object storage; a public link is returned; the product record is created or updated with that link in the image field. If an image format is not supported, the request is rejected with a clear reason. Adding support for a new format requires only extending validation and storage mapping, without changes to the rest of the product flow.
Response Measure
Compatibility: accept at least jpeg, png, and webp at launch; adding a new format requires less than one developer-day.
Correctness: zero broken image links across a test set of at least one hundred items per format; one hundred percent of accepted uploads return a valid public link recorded on the product.
Safety: reject unsupported types and oversize files with clear error messages; no partial product writes when the image is rejected.
Performance: median upload and link write complete in less than fifteen seconds for images up to five megabytes; ninety-fifth percentile completes in less than thirty-five seconds.

Mapped Tactics/Patterns:

c. Encapsulate: catalogETL service abstracts data processing from specific sources
d. Adhere to standards: csv/json format with required fields for partner integration

Testability

1. Cart operations integration test

Source: Developer
Stimulus: Wants to test cart business logic with stock validation and ownership checks
Artifact: Cart operations system with mock database
Environment: Test environment
Response: System validates cart workflows, stock enforcement, and ownership validation through isolated integration tests
Response Measure: All cart test cases pass within 1 second; stock validation prevents overselling; ownership checks block unauthorized access; complete cart workflow executes successfully

Mapped Tactics/Patterns:

- Abstract Data Sources: the cart test uses a mockDatabase object that abstracts away the real database

2. Flash sale buying test

Source: Developer
Stimulus: Wants to test flash sales functionality under high concurrent load
Artifact: Flash sales system with surge testing
Environment: Test environment
Response: System validates flash sales behavior with 50 concurrent buyers, measures latency, and ensures stock integrity
Response Measure: Test completes within 30 seconds with 95% success rate; stock violations detected and reported; latency stays under 5 seconds per transaction
Mapped Tactics/Patterns:

- Test Isolation: cleanupExistingTestData() ensures each test run starts with clean state

Security

1. Unauthorized business access prevention

Source: Malicious user attempts to access business features
Stimulus: Attempts to upload product catalog without proper business verification
Artifact: business API endpoints
Environment: runtime
Response: system validates business access and blocks unauthorized access
Response measure: unverified businesses cannot upload catalog; the feature is blocked until status is verified

Mapped Tactics/Patterns:

- Authenticate actors (JWT verification)
  Manually validate the license plate number and the establishment card URL against trusted sources before granting business feature access, ensuring the requester represents a legitimate, verified entity.
- Authorize actors (RBAC, deny-by-default)
  Check required role/permission (e.g., business:upload) and reject missing/insufficient roles with 403, ensuring no privileged action is possible without explicit grant.
- Limit access (route scoping / endpoint restriction)
  Expose upload endpoints only under protected business scopes; reject traffic that doesn't meet both auth and verification policies, shrinking the attack surface.

2. Unauthorized admin panel access prevention

Source: Malicious user attempts to access admin features
Stimulus: Attempts to access admin panel without proper admin role verification
Artifact: Admin API endpoints and frontend components
Environment: Runtime
Response: System validates admin access and blocks unauthorized access
Response Measure: Non-admin users cannot access admin panel; admin features are blocked until role is verified as ADMIN

Mapped Tactics/Patterns:

- Authenticate actors (JWT verification)
- Authorize actors (RBAC, deny-by-default)
- Limit access (route scoping / endpoint restriction)