

AI Issue Summary Documentation

Thank you for using our application! AI Issue Summary aims to quickly and easily generate AI-driven summaries of your ticket descriptions.

How to Use the App

1. Open any ticket you want to summarize. Tickets with more lengthy descriptions work best.

Projects /  Plugin Dev /  PD-9

Improve Performance of Product Search API by Implementing Server-Side Caching

+ Add @ Apps

Description

Currently, the `/api/products/search` endpoint experiences high latency (700–1200ms) under moderate load due to repeated computation of complex query filters and large-scale database access. This directly impacts user experience on both desktop and mobile when searching for products.

To address this, we propose introducing **server-side caching** for popular search queries and filters using **Redis**, which has already been integrated into our stack. Cached results should be stored for 10 minutes per unique query string to prevent overloading the database during high-traffic periods.

This initiative is in line with our Q3 goal of reducing page load times by 30% across critical flows.

Out of Scope:

- Frontend-level caching or local storage mechanisms.
- Changes to the database schema.

Technical Notes:

- Use the `query-string` as the Redis cache key.
- TTL (Time To Live) for each cache entry: 600 seconds.
- Use existing Redis connection pool via `redisClient` from `cache-utils.js`.
- Example cache key: `productSearch:query=boots&size=8&color=black`

Acceptance Criteria

- ✅ Implement Redis-based caching mechanism for `/api/products/search`.
- ✅ Cache is applied per unique query (including filters, keywords, sort, pagination).
- ✅ Cached results expire after 10 minutes or when explicitly invalidated by a related product change event (e.g., stock update).
- ✅ Include monitoring (e.g., Prometheus metrics or logs) for cache hit/miss ratio.
- ✅ Update unit and integration tests to cover caching logic and fallback behavior.
- ✅ No significant changes to existing API response format or client-side integration.
- ✅ Performance benchmark shows at least 40% reduction in average response time under simulated load.

Activity

Show: All Comments History Work log

Newest first ↕

1

Add a comment...

 Looks good!

 Need help?

 This is blocked...

 Can you clarify...?

 This is on track

Pro tip: press **M** to comment

2. At the top of the ticket, click the “Apps” button and select “AI Issue Summary”.

Projects / Plugin Dev / PD-9

Improve Performance of Product Search API by Implementing Server-Side Caching

+ Add

Apps

Description

AI Issue Summary

Currently, th

This directly

+ Add apps

point experiences high latency (700–1200ms) under moderate load due to repeated computation of complex query filters and large-scale database access. Both desktop and mobile when searching for products.

To address this, we propose introducing **server-side caching** for popular search queries and filters using **Redis**, which has already been integrated into our stack. Cached results should be stored for 10 minutes per unique query string to prevent overloading the database during high-traffic periods.

This initiative is in line with our Q3 goal of reducing page load times by 30% across critical flows.

Out of Scope:

- Frontend-level caching or local storage mechanisms.
- Changes to the database schema.

Technical Notes:

- Use the `query-string` as the Redis cache key.
- TTL (Time To Live) for each cache entry: 600 seconds.
- Use existing Redis connection pool via `redisClient` from `cache-utils.js`.
- Example cache key: `productSearch:query=boots&size=8&color=black`

Acceptance Criteria

- ✔ Implement Redis-based caching mechanism for `/api/products/search`.
- ✔ Cache is applied per unique query (including filters, keywords, sort, pagination).
- ✔ Cached results expire after 10 minutes or when explicitly invalidated by a related product change event (e.g., stock update).
- ✔ Include monitoring (e.g., Prometheus metrics or logs) for cache hit/miss ratio.
- ✔ Update unit and integration tests to cover caching logic and fallback behavior.
- ✔ No significant changes to existing API response format or client-side integration.
- ✔ Performance benchmark shows at least 40% reduction in average response time under simulated load.

Activity

Show:

All

Comments

History

Work log

Newest first ⌵

I

Add a comment...

Looks good!

Need help?

This is blocked...

Can you clarify...?

This is on track

Pro tip: press **M** to comment

3. The AI Issue Summary app will appear at the bottom of the ticket. Click the “Generate Summary” button.

Projects / Plugin Dev / PD-9

Improve Performance of Product Search API by Implementing Server-Side Caching

+ Add

@ Apps

Description

Currently, the `/api/products/search` endpoint experiences high latency (700–1200ms) under moderate load due to repeated computation of complex query filters and large-scale database access. This directly impacts user experience on both desktop and mobile when searching for products.

To address this, we propose introducing **server-side caching** for popular search queries and filters using **Redis**, which has already been integrated into our stack. Cached results should be stored for 10 minutes per unique query string to prevent overloading the database during high-traffic periods.

This initiative is in line with our Q3 goal of reducing page load times by 30% across critical flows.

Out of Scope:

- Frontend-level caching or local storage mechanisms.
- Changes to the database schema.

Technical Notes:

- Use the `query-string` as the Redis cache key.
- TTL (Time To Live) for each cache entry: 600 seconds.
- Use existing Redis connection pool via `redisClient` from `cache-utils.js`.
- Example cache key: `productSearch:query=boots&size=8&color=black`

Acceptance Criteria

☒ Implement Redis-based caching mechanism for `/api/products/search`.

☒ Cache is applied per unique query (including filters, keywords, sort, pagination).

☒ Cached results expire after 10 minutes or when explicitly invalidated by a related product change event (e.g., stock update).

☒ Include monitoring (e.g., Prometheus metrics or logs) for cache hit/miss ratio.

☒ Update unit and integration tests to cover caching logic and fallback behavior.

☒ No significant changes to existing API response format or client-side integration.

☒ Performance benchmark shows at least 40% reduction in average response time under simulated load.

AI Issue Summary DEV

☆ Generate Summary!

4. The summary of the ticket’s description will appear in real time on the ticket!

Projects / Plugin Dev / PD-9

To address this, we propose introducing **server-side caching** for popular search queries and filters using **Redis**, which has already been integrated into our stack. Cached results should be stored for 10 minutes per unique query string to prevent overloading the database during high-traffic periods.

This initiative is in line with our Q3 goal of reducing page load times by 30% across critical flows.

Out of Scope:

- Frontend-level caching or local storage mechanisms.
- Changes to the database schema.

Technical Notes:

- Use the `query-string` as the Redis cache key.
- TTL (Time To Live) for each cache entry: 600 seconds.
- Use existing Redis connection pool via `redisClient` from `cache-utils.js`.
- Example cache key: `productSearch:query=boots&size=8&color=black`

Acceptance Criteria

- Implement Redis-based caching mechanism for `/api/products/search`.
- Cache is applied per unique query (including filters, keywords, sort, pagination).
- Cached results expire after 10 minutes or when explicitly invalidated by a related product change event (e.g., stock update).
- Include monitoring (e.g., Prometheus metrics or logs) for cache hit/miss ratio.
- Update unit and integration tests to cover caching logic and fallback behavior.
- No significant changes to existing API response format or client-side integration.
- Performance benchmark shows at least 40% reduction in average response time under simulated load.

AI Issue Summary DEV

This Jira ticket addresses high latency (700–1200ms) on the `/api/products/search` endpoint caused by repeated complex queries and large database access. To improve performance, the plan is to implement Redis-based server-side caching for popular queries, with results stored for 10 minutes and invalidated by product updates. The goal aligns with Q3 objectives to reduce page load times by 30%. Success criteria include caching implementation, monitoring, testing coverage, minimal API changes, and a 40% reduction in response time under load.

Activity

Show: All Comments History Work log

Newest first 17