Performance Optimization Report

1. Identified Bottlenecks

Before Optimization:

- **Data Retrieval Latency**: Frequent reads from the database for operations like fetching all products or retrieving products by ID could lead to increased response times, especially under high load.
- **Database Load**: Repeated queries for the same data resulted in high database read operations, which could degrade performance and scalability.
- **Query Performance**: Lack of indexes on frequently queried fields such as `name`, `price`, and `categoryId` could lead to slower query execution times.

2. Performance Improvements

Caching Implementation:

- **Reduction in Latency**: By introducing caching for the `getAllProducts()` and `getProductById(String id)` methods, response times are significantly reduced. Cached data is served from an in-memory store rather than querying the database repeatedly.
- **Decreased Database Load**: Caching reduces the number of database reads, alleviating the load on the database and improving its performance and scalability.
- **Consistency**: Used `@CachePut` to ensure that the cache is updated when products are saved, and `@CacheEvict` to remove products from the cache when deleted, maintaining data consistency.

Indexing Implementation:

- **Improved Query Performance**: Indexes on fields like `name`, `price`, and `categoryId` have optimized query performance. Indexes speed up the retrieval of documents based on these fields and improve the efficiency of queries.
- **Optimized Resource Utilization**: Indexing reduces the amount of time required to search and filter data, leading to faster response times and better user experience.

3. Report Comparing Before and After Optimization

Before Optimization:

- **Response Time for `getAllProducts()`**: Average response time was higher due to fetching data directly from the database on each request.

- **Response Time for `getProductById(String id)`**: Slower due to database reads without caching.
- **Query Execution Time**: Longer due to lack of indexes, especially for fields frequently used in queries.

After Optimization:

- **Response Time for `getAllProducts()`**: Reduced due to caching, with data being served from an in-memory cache.
- **Response Time for `getProductById(String id)`**: Significantly improved with caching, reducing the need for frequent database queries.
- **Query Execution Time**: Enhanced due to indexing, leading to faster execution of queries involving `name`, `price`, and `categoryId`.

4. Application's Adherence to 12-Factor Principles

1. Codebase

- **One codebase tracked in version control, many deploys**: The application is maintained in a single code repository, with different deployments (e.g., development, staging, production) derived from this codebase.

2. Dependencies

- **Explicitly declare and isolate dependencies**: All dependencies are declared in `pom.xml`, ensuring they are managed explicitly and are isolated from the application's codebase.

3. Config

- **Store configuration in the environment**: Configuration settings, such as database credentials and environment-specific settings, are stored in `application.properties`, which is used to manage environment-specific configurations outside of the codebase.

4. Backing Services

- **Treat backing services as attached resources**: Caching has been implemented as a backing service, improving performance by reducing the number of database reads and managing this service separately from the core application logic.

5. Build, Release, Run

- **Strictly separate build and run stages**: The processes of building, releasing, and running the application are distinct, ensuring a clear separation between these stages.

6. Processes

- **Execute the app as one or more stateless processes**: The application is designed to run as stateless processes, with state stored externally, such as in a database or cache.

7. Port Binding

- **Export services via port binding**: The application exposes its services through network ports, allowing it to run as a standalone service.

8. Concurrency

- **Scale out via the process model**: The application is designed to scale horizontally by running multiple processes to handle increased workload.

9. Disposability

- **Maximize robustness with fast startup and graceful shutdown**: Processes are able to start and stop quickly, and handle termination gracefully, ensuring robustness and scalability.

10. Dev/Prod Parity

- **Keep development, staging, and production as similar as possible**: Development, staging, and production environments are kept as similar as possible to avoid discrepancies and ensure consistent behavior.

11. Logs

- **Treat logs as event streams**: Logs are treated as a stream of events and managed externally, allowing for aggregation and processing by external tools.

12. Admin Processes

- **Run admin/management tasks as one-off processes**: Administrative tasks, such as database migrations, are run as separate one-off processes, ensuring they do not interfere with the regular application processes.
-

Conclusion

The optimization efforts, including caching and indexing, have led to significant improvements in response times and query performance. These changes address identified bottlenecks, enhance application performance, and align with the 12-factor principles, ensuring a robust and scalable application.