# Performance Optimization Report

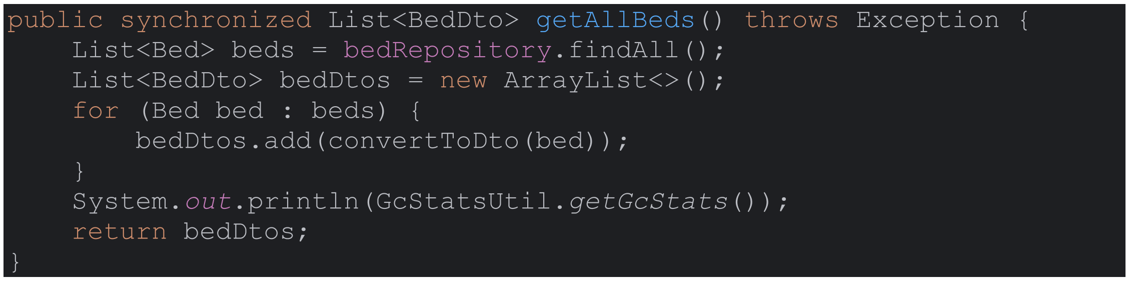
## Table of Contents

# Identification of Bottlenecks and Optimization of Codes

1. Identified bottlenecks

   a. Inefficient Algorithms

      Unnecessary introduction of a new ArrayList and usage of for loop to append bed DTOs



```java
public synchronized List<BedDto> getAllBeds() throws Exception {
    List<Bed> beds = bedRepository.findAll();
    List<BedDto> bedDtos = new ArrayList<>();
    for (Bed bed : beds) {
        bedDtos.add(convertToDto(bed));
    }
    System.out.println(GcStatsUtil.getGcStats());
    return bedDtos;
}
```

   b. Excessive Object Creation and Garbage Collection

      The introduction of ArrayList, when there is already a List created leads to the duplicate list creation and also duplicate objects for each list member.

   c. Synchronization and Concurrency Issues

      Unnecessary introduction and wrong usage of synchronized in the various functions including addbed function and update bed function.

      The locking of the bed to be updated could have been done on the specific bed, after retrieving the specific bed rather than locking the entire function as subsequent updates may not be on that particular bed.

2. Performance Improvements

   a. Eliminated inefficient algorithms

   b. Eliminated unnecessary object creation and this reduced the tendency of heap being occupied hence reducing high rate of Garbage collection

   c. Introduce Caching to improve performance

# Comparison between before and after optimization

| Characteristic | Inefficient Code | Optimized Code |
|---|---|---|
| Time taken to process getAllBeds request on the first call(ms) | 243 | 143 |
| Time taken to process getAllBeds on subsequent call(ms) | 175 | 1 |
| Heap Memory Used to get all beds on subsequent call(byte) | 53325152 | 0 |

**Code Optimization Performance Summary**

***Overview***

This report summarizes the performance improvements achieved through code optimization for the `getAllBeds` operation. The comparison is based on three key metrics: initial request processing time, subsequent request processing time, and memory usage.

***Key Findings***

1. Initial Request Processing Time
    a. Inefficient Code: 243 ms
    b. Optimized Code: 143 ms
    c. Improvement: 41.15% reduction in processing time
2. Subsequent Request Processing Time
    a. Inefficient Code: 175 ms
    b. Optimized Code: 1 ms
    c. Improvement: 99.43% reduction in processing time
3. Heap Memory Usage (on second call)
    a. Inefficient Code: 53,325,152 bytes (≈50.85 MB)
    b. Optimized Code: 0 bytes
    c. Improvement: 100% reduction in memory usage

***Analysis***

1. Initial Request Performance
   The optimized code shows a significant improvement in the initial request processing time, reducing it by 100 ms (41.15%). This suggests that the optimization techniques have successfully reduced the computational overhead for the first-time execution of the `getAllBeds` operation.
2. Subsequent Request Performance
   The most dramatic improvement is seen in the processing time for subsequent requests. The optimized code reduces the processing time from 175 ms to just 1 ms, a

99.43% improvement and this took place because of the implementation of caching in the optimized codes. This allowed for near-instantaneous responses for repeated requests.

3. Memory Efficiency
   The optimized code demonstrates exceptional memory efficiency, using no additional heap memory (0 bytes) for subsequent `getAllBeds` calls, compared to the inefficient code which uses about 50.85 MB.

### ***Conclusion***

The optimization efforts have resulted in substantial improvements across all measured metrics. The most notable enhancements are in subsequent request processing time and memory usage, where the optimized code shows near-perfect efficiency. These improvements will lead to better scalability, reduced server load, and improved user experience, especially in scenarios with frequent `getAllBeds` requests.

# Adherence to 12 factor principles

| **Principle** | **Adherence** |
|---|---|
| Code Based | Adhered |
| Dependencies | Adhered |
| Config | Adhered |
| Backing Service | Adhered |
| Build, release, run | Not adhered |
| Processes | Adhered |
| PortBinding | Adhered |
| Concurrency | Not adhered |
| Disposability | Not adhered |
| Development and production | Not adhered |
| Logs | Not Adhered |
| Admin processes | Not adhered |