1. The Issue

We choose **blocking operations**: The main issue is that the system handles requests in a blocking way, meaning long database queries can tie up server workers and database connections for a long time, and there are no protections like timeouts or limits to stop a flood of requests. This is a problem because a few slow queries (or many repeated requests) can fill the database connection pool, make other requests wait in line, and cause response times to spike for everyone, potentially leading to timeouts and cascading failures as clients retry and add even more load. It becomes noticeable when traffic increases, when the database is slow due to heavy queries, missing indexes, or lock contention, or when request spam happens (accidental or malicious), because the system can't shed load and instead gets "stuck" serving slow work while everything else backs up.

2. The Solution

Our first idea is to implement rate limiting at the getaway level.
We plan to also use redis on heavy and often accessed routes to check cache before querying the database.

Changes

We plan to add a rate limiter for our backend to limit spam requests which could block the backend. We plan to do this by using a package from npm called "express-rate-limiter" which is a *middleware* that we can hook up to express.

We also plan to add redis to save cache on heavy and often accessed data. This can reduce database load and make requests faster.

Improvement

This would stop users from spamming the server with requests, which would decrease the possibility of server strain from spam. and redis would help increase server efficiency.

Trade-offs

Introducing redis increases the complexity of our architecture which could lead to more failure*. (We can limit this by not be depending on the redis service).* Redis could also introduce inconsistency for the requests because cache isn't always the news/right data.

3. Refactoring

Modification

We will only be looking at the backend, mostly in some of the routes that we want to add cache to. We will also add a middleware to the main express file which will handle the rate limitations.

No changes

We will not change our algorithm, our system that is not user interactive will mostly stay the same since they have nothing to do with server request speed.

Refactoring

- Add middleware in index.js
- Check which routes to add cache to
  - get user
  - get exercises
  - etc

4. Begin refactoring

Routes to add redis to

1. Get user
2. get exercise
3. get workout
4. leaderboard friends
5. stats