Your task is to design and implement oauth token management system that must include two primary API endpoints and prepare supporting infrastructure, following the architecture discussed in the lecture that you can find in .excalidraw file.

The system will consist of two endpoints: `/token`, which issues new tokens with a two-hour lifetime(issue new one only if 75% of lifetime expired) and associates them with specific client-provided scopes and `/check`, which validates a token's status and returns its associated scopes . If the token has expired or is invalid, `/check` endpoint should return an error message.

System will serve approximately 1 000 users, where each user represents an external system relying on your service to send notifications (e.g., push or SMS) to their clients. These users are inefficient in their usage patterns and will frequently request new tokens before every `/check` call, regardless of the validity of their existing tokens.

You need to implement an oauth API in three programming languages: Python, Golang, and Java. For each language, you are required to carefully select a suitable framework. Additionally, you must document your decisions regarding programming languages and frameworks in the Architecture Design Record (ADR) format. Each ADR should include the context, decision, and consequences, providing a clear justification for why a particular language and framework were chosen based on suitable factors.

A few additional constraints and notes must be considered

1. You can use single-node database for this task, meaning there is no need to set up multiple nodes or replication. As a result, the VIP (Virtual IP) layer described in the diagram is not required
2. The connection pool for database interactions must be external to your application
3. To evaluate the performance of your implementations, you can use tools such as JMeter, Locust, or any custom scripts. If you choose to create custom scripts, note that they will be double-checked for correctness

The system should be designed to handle a consistent load of 20 000 requests per second. Your implementation should be tested under both single-instance and multi-instance setups.

Finally, prepare report summarizing your work. This should include performance benchmarks comparing three implementations. Be sure to clearly articulate how the design and framework decisions impacted the results and how they align with your thoughts in ADR.