1. Дизајн шеме базе података

Слика sema у репозиторијуму.

2. Партиционисање података

Партиционисањем података можемо повећати скалабилност система и побољшати перформансе. Применили бисмо хоризонтално и вертикално партиционисање. Како је претпоставка да постоји велики број корисника, за табелу Корисник предлажемо хоризонтално партиционисање према алфабету у блоковима од по два узаступна слова – гледа се почетно слово корисничког имена.Такође предлажемо хоризонтално партиционисање за табелу резервација на основу типа резервације... За све табеле у којима је било дозвољено логички обрисати ентитет, предлажемо хоризонтално партиционисање према вредности колоне активан.

3. Репликација базе и обезбеђивање отпорности на грешке

За репликацију базе података бисмо искористили Master/slave режим. Ова конфигурација подразумева да се све промене у подацима морају слати једном серверу – главни сервер, који ће те податке прослеђивати осталим slave серверима. У master базу података се подаци уписују и по потреби се могу и читати. Slave база података би постала master база података уколико сервер master базе података престане са радом, самим тим је потребно да master и slave базе података буду синхронизоване. Slave базе података бисмо искористили за читање података и на овај начин растеретили master базу података.

4. Кеширање података

Кеширање података се уводи како би приступ подацима био бржи. Користећи кеш, додатно бисмо побољшали перформансе апликације. У Spring-у

подразумевано кеширање је кеширање првог нивоа (L1, Session level). Кеширање првог нивоа омогућава да у оквиру једне сесије сваки захтев за објектом из базе враћа увек исту инстанцу објекта. Постоји и кеширање другог нивоа (L2, SessionFactory level). Ово кеширање подразумева да се подаци добијени из базе привремено чувају на серверу. Овакво кеширање подразумева да се при сваком новом упиту прво проверава да ли се тражени објекти већ налазе у кешу, а онда, уколико се не налазе, се приступа бази података. Коришћењем кеширања другог нивоа растеретили бисмо оптерећење над базом података. Како бисмо балансирали између оптерећења базе и ажурираности података у кешу, конфигурисали бисмо time to live.

5. Процена за хардверске ресурсе потребне за складиштење свих података у наредних 5 година

Претпоставимо да би на дневном нивоу корисник послао у базу података просечно 30KB података, самим тим добијамо:
(30KB * 30 дана) * 12 месеци * 5 година * 200 000 000 корисника = 10800000GB

6. Постављање load balancer-а

Распоређивање оптерећења на више серверских рачунара може да доведе до тога да наш систем може да обради више захтева у јединици времена. Алгоритам који бисмо употребили је Least Connections који подразумева слање захтева серверу који тренутно има најмањи број конекција ка клијентима. Још један алгоритам који бисмо искористили је Round Robin. Овај алгоритам подразумева да се захтеви серверима шаљу секвенцијално. Round Robin алгоритам бисмо могли побољшати додавањем тежина сваком од сервера којима се шаљу захтеви, где су тежине одређене у складу са капацитетом и перформансама сервера. На овај начин, више захтева за обраду би добијали сервери који имају боље перформансе, тј. сервери који имају већи тежински број.

7. Операције корисника које треба надгледати

Како бисмо одредили перформансе наше апликације, а на основу тога и побољшали саме перформансе система, посматрали бисмо одређене операције корисника, као што су број пристиглих захтева за регистрацију на дневном нивоу, број пристиглих захтева за резервације у секунди, сату и дану.

8. Цртеж дизајна архитектуре