**SYSTEM ARCHITECTURE FOR A SCHOOL MANAGEMENT SYSTEM.**

Users

1. Students
2. Teachers
3. Parents

Functional Requirements:

*Student*

1. Should be able to login.
2. Should be able to download assignments.
3. Should be able to upload assignments
4. Should be able to take online exam on the platform for a specifies timeframe eg. 1 hour
5. Should be able to view and print results.
6. Should be able to view available course

*Teacher*

1. Should be able to login.
2. Should be able to post assignments.
3. Should be able the access student’s performance, grade uploaded assignments and exams.
4. Should be able to upload course materials and lectures.

*Parent*

1. Should be able to login.
2. Should be able to view the performance of his/her ward.

Above are a some of the functionalities of a standard school management system. With a system as complex as this, it is expediate to create an architecture that can easily scale.   
For a system like this to scale easily,  
I use the microservice API gateway design pattern with caching.

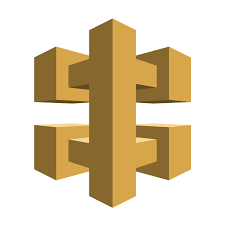
Why this?  
Microservice architecture breaks down the system into independent sections (services) that can still communicate with each other. Why microservice architecture?

* Loosely coupled
* Independently deployable
* Highly maintainable, testable and scalable.

These functionalities can be taken as the services, can be worked on independently and deployed separately. Clearly, this architecture has no single point of failure.  
Decentralized data storage is best for microservice architecture hence each of the 3 services here will have separate databases. There can be additional databases to serve as redundant databases.  
The API gateway design routes a request to the concerned microservice. In this school management system, if there is a client request for student grades, the request is routed to the students microservice and the desired request is granted.  
It can send the request to multiple services and similarly aggregate the results back to the client

Caching is implemented for this architecture because it reduces the number of trips a microservice needs to make to a database server and avoid redundant calls to other microservices. This reduces latency and gives the client a better experience.

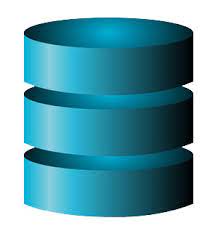
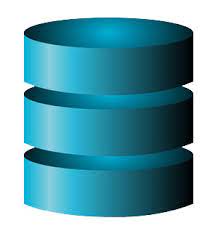
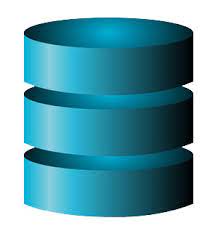
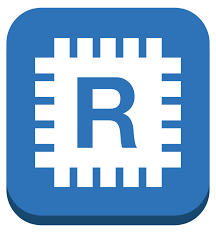
Client  
Request



API Gateway



Load Balancer



Database

Cache

Student  
Service

Parent  
Service

Teacher  
Service