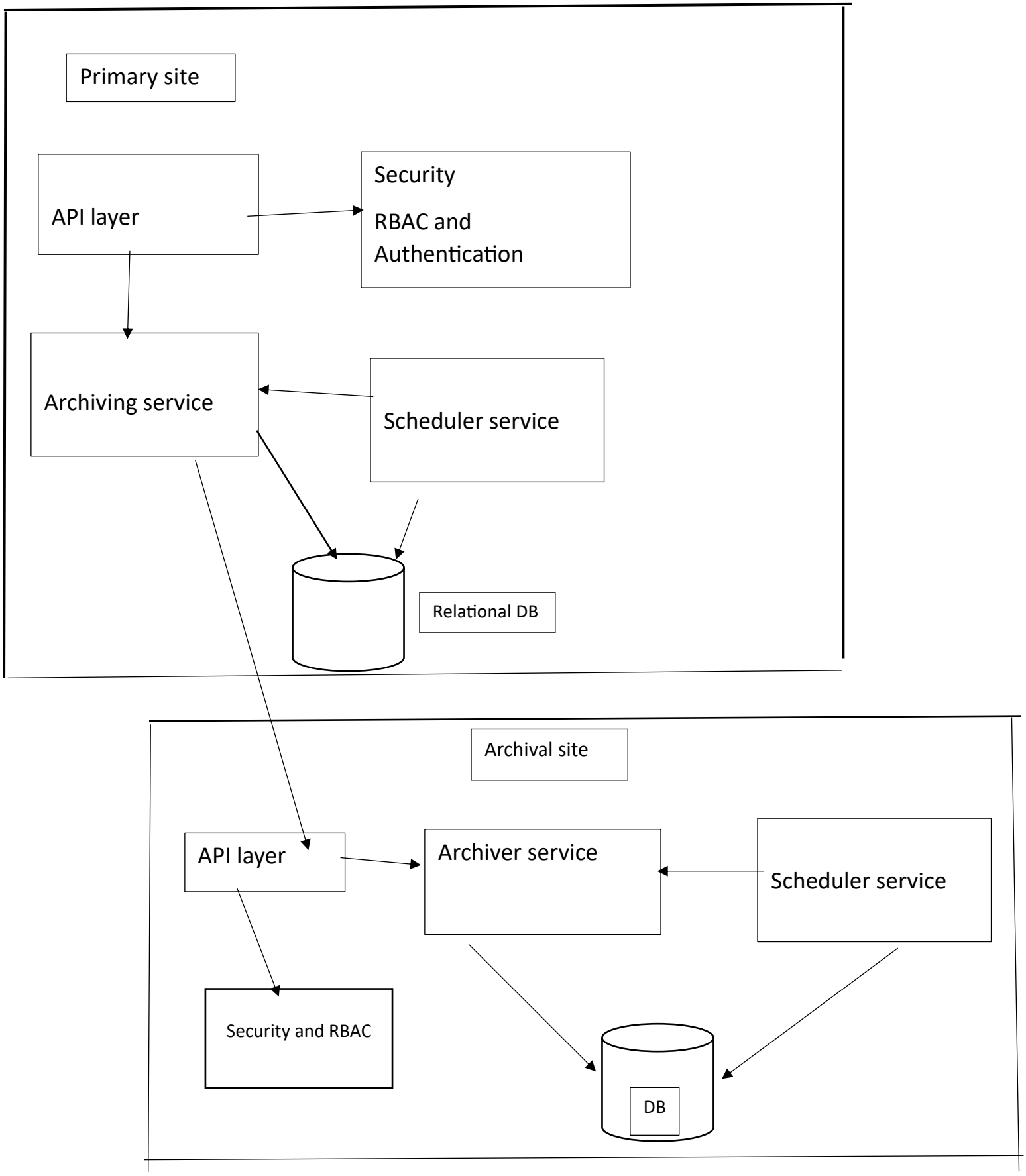
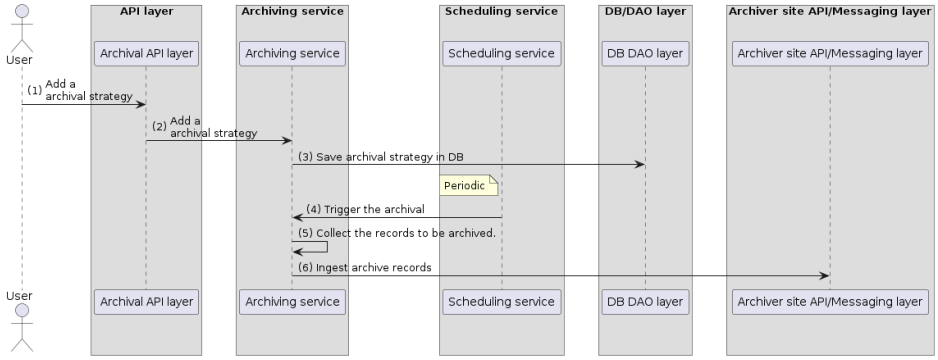
Design document – Archival service



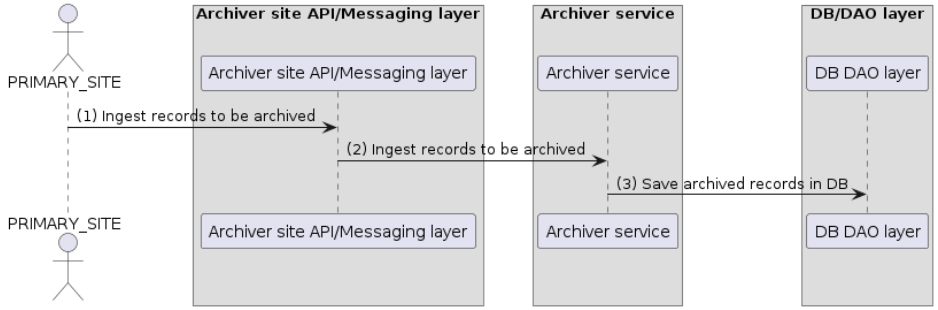
Sequence diagrams

Data archive

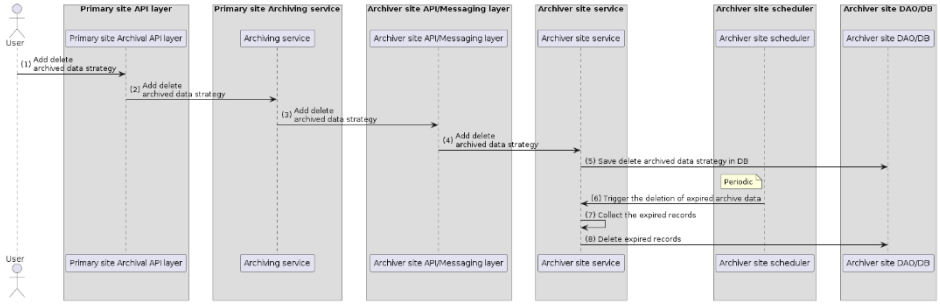
Primary site flow to archive the data as per archiving strategy suggested by user



Ingestion of archived data from primary site to archival site.



Deletion of archived data as per deletion strategy - happens at data archive site



Details of individual services.

API layer –

Interacts and serves the user requests.

APIs

1. **Login**

/api/auth/login – Takes parameters as username and password and returns a JSON web token (JWT) as response. The JWT is to be used as Authorization header in subsequent API requests.

2. **Add archival strategy.**

POST /archival-strategies

The API saves the user provided archival strategy in DB.

Input – The DB table for which archival strategy is to be applied, the archive Unit and archive value

For e.g. For statement, “Archive the data older than six months for student table” input will be as follows

{

“tableName”:”student”,

“archiveUnit”:”MONTHS”,

“archiveValue”:6

}

3. **Delete archived data strategy**

The API calls the remote site archiver service to configure this deletion strategy

Input – The DB table for which deletion strategy is to be applied, the archive Unit and archive value

For e.g. For statement, “Data older than 2 years can be deleted from archival storage for student table” input will be as follows

{

“tableName”:”student”,

“archiveUnit”:”YEARS”,

“archiveValue”:2

}

4. **Get student archived data**

GET /student-archives

Gets the archived records.

RBAC checks verify whether the given user role has access to the API/Student entity or not.

Scheduler Service

The main job of the service is to provide a periodic trigger for archival related activities.

On primary site, this service triggers the archival service periodically to send the data to remote archiver service as per user configured strategy.

On secondary site, this service calls the archiver service periodically to initiate deletion of the expired records.

The service works on cron expressions which triggers the required job in a periodic way.

Archiving service

On primary site, this service does the job of sending the data periodically to remote archiver service to store and manage it. Once sent to the remote site, the data is deleted from primary site.

On the remote site, this service ingests the data received from primary into the DB and does the periodic deletion of expired records.

The data can be dumped to a low-cost storage before it is deleted.

Design/Implementation decisions

Inter service communication – Data mobility.

Though the attached implementation demonstrates inter service communication via REST APIs, a MQ/Messaging based channel like RabbitMQ or Kafka can also be implemented.

The MQ will provide a better asynchronous approach to the communication also it will decouple the involved micro services.

Choice of database for archival storage

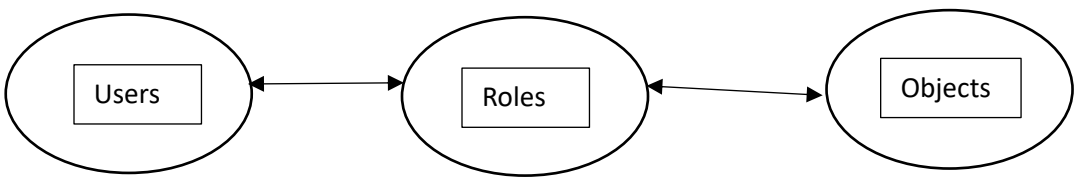
The attached implementation is a SQL based implementation for the archival storage. The given use case fairly defines a well-structured data (like Student records) which follow a specific schema and if the archival data needs to maintain ACID properties of the source site data, SQL can be a choice here.

A non-relational DB can also work in this case so that the archiver site does not need to worry about the DB schema creation in advance to persist the records to be archived. This eliminates a significant developer load to identify and implement/populate the required schemas on the archive site. NoSQL databases can handle large volumes of data at high speed which might be the case if archival rate is higher in applications.

Choice of technology – Spring boot

Most of the languages do provide frameworks to deploy applications in a containerized environment. Spring Boot gives an easier, quicker path to set up, configure, and run apps in a containerized environment which helps easier cross platform deployments.

Role based access control.



The implementation follows a standard RBAC model where a Role manages objects and permissions and Users are assigned with Roles.

For e.g. “Student” role has access to “Student” objects and a user named “Harry” is assigned with a role “Student”.