1. Keeping In mind the architectural design for the requirements, this microservice has been developed using REST / HTTP protocols.

There are a number of reasons for choosing REST over SOAP

* Lesser bandwidth
* JSON parsing is faster than XML parsing
* Development is easier and faster
* No additional changes required for an existing architecture

Controller: <http://localhost:8080/userReg>

POST: creates new user

Reqest body:

{

"fName" : "Ram",

"lName" : "STK",

"userID" : "ramstk",

"eMail" : "ramstk@test.com",

"password" : "Test@1234",

"cPassword": "Test@1234",

"completed" : "true"

}

Response: contains the user created details along with the status 200. Failure case such as passwords not matching causes 400 error along with the custom exception message.

GET: [http://localhost:8080/userReg/{id}](http://localhost:8080/userReg/%7bid%7d)

Fetches the user details with status 200 if found else returns a custom error message along with error 404 if the id is not present.

GET: http://localhost:8080/userReg

Fetches the entire list of users along with status 200. If there’s is no data present, an error 404 is thrown with no data message.

PATCH: http://localhost:8080/userReg

Is used for updating the user details. Since we don’t want to update the entire user info in a single request. The partial user info update return the changed user details along with status code 400 for improper data or 404 if the id is not present in the data.

Reqest body:

{

"id" : "1",

"fName" : "RamTest"

}

Response body:

{

"id" : "1",

"fName" : "RamTest",

"lName" : "STK",

"userID" : "ramstk",

"eMail" : "ramstk@test.com",

"password" : "Test@1234",

"cPassword": "Test@1234",

"completed" : "true"

}

DELETE: [http://localhost:8080/userReg/{id}](http://localhost:8080/userReg/%7bid%7d)

Is used for deleting a particular user. The response contains 200 when successfully deleted or 400 if user id isn’t found.

1. Considering the requirement for heavy traffic, the REST API is developed as a spring boot application. Spring boot can process close to 25000 requests per second with basic memory of 8gb and on a 4 cores processor. Since the server doesn’t store the state for a REST service, horizontal scaling is possible without any need for changing the current system’s specifications.
2. The controller can be changed to contain the version of the API. This would help us in building out new services with support to older API’s for existing applications. The components should be checked into a single source control repository so that the existing code can be imported when building a new service if required. By having a common Data source, data can be easily shared if required. Since the server doesn’t contain the server state, we can internally call the user service from the new service and validate the user credentials.
3. By protecting the service using a webseal or a gateway the end client would not know the destination of the request. The webseal can reroute the request based on the reverse proxy protocol. This helps in protecting the server identity leveraging the webseals security features. The requests can be grouped based on roles. Role based access for requests such as PUT/PATCH, DELETE ensure that only authorized users can make those requests. The passwords need to be encrypted with a secured encryption mechanism such as bcrypt. The password should never be sent as response. By using authentication tokens such as JWT or OAUTH2 we can successfully authorize the user to make data sensitive requests. Proper REST security needs to be followed when designing API’s such as POST for adding sensitive data. For user authentication, we should never reveal the password. The API should internally check for the hashed/encrypted password when authenticating instead of decrypting the password in the user database table. Instead of using the ID for updating/retrieving or deleting a different unique identity should be used and the internal ID should always remain in the system.
4. We can integrate our source control repository with Jenkins/Amazon CodeCommit for CI/CD purpose. Each push can trigger the tool to deploy the application when required. The tool should leverage the security features for deployment so that there is no source data leak when the code is transferred. The CI tool ensures that the latest code is deployed to different environments through which different phases of testing can be done.
5. Tech Stack for the Service:

* MySQL DB- open source, easy to configure and optimize, multiple instances can be controlled easily.
* Maven- can easily download any dependencies if required
* Spring boot- easy to implement, supports high scalability, provides built in security frameworks, supports latest security tools, process high frequency of requests, easy to implement reusable components.
* Apache tomcat (comes embedded with Spring Boot)
* JMX and JConsole for monitoring
* AWS or GCE based on the scope
* Webseal (for reverse proxy lookup)
* Bcrypt for password encryption