Test ID	Short Desc of Requirement	Category/Module (use for sorting purposes)	Priority	Requirement ID	Purpose	PreRequisite	Procedure	Expected Results	System test Passed	Actual Result
Test ID	Short Desc of Requirement	Category/Module (use for sorting purposes)	Priority	Requirement ID	Purpose	PreRequisite	Procedure	Expected Results	System test Passed	Actual Result
1	When launched, the application should wait for http connections	POST	1	1	Responds to http POST request	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux &	Step 1: Postman POST application/json '{"password":"angrymonkey"}' http://127.0.0.1:8088/hash	1. 200 and a job identifier, an integer	P	
2	It should answer on the PORT specified in the PORT environment variable.	POST	1	2	Responds from port specified via PORT, 8088, variable	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux &	Step 1: Postman POST application/json '{"password":"angryhuman"}' http://127.0.0.1:8088/hash	job identifier, an integer	P	
3	It should answer on the PORT specified in the PORT environment variable.	POST	1	2	Responds from port specified via PORT, 8089 variable	1. PORT=8089 2. Application is running: nohup ./broken-hashserve_linux &	Step 1: Postman POST application/json '{"password":"angryhuman"}' http://127.0.0.1:8089/hash	job identifier, an integer	P	
4	Port must be set	POST	1	0	Without a port service does not run	1. No port set	Step 1: Run application: nohup ./broken-hashserve_linux & Step 2: Check running processes: ps -ef | grep broken-hashserve	1. Application does not run 2. Application is not shown in a ps	P	
5	It should answer on the PORT specified in the PORT environment variable.	POST	3	0	Service can run on 65535	1. PORT=65535 2. Application is running: nohup ./broken-hashserve_linux &	Step 1: PORT=65535 Step 2: nohup ./broken-hashserve_linux & Step 3: Postman POST application/json '{"password":"angryhuman"}' http://127.0.0.1:1/hash Step 4: Postman GET "application/json" http://127.0.0.1:8088/hash/<job id>	2. Service starts up 3. Job id is returned 4. SHA512 password returned	P	
6	Password is SHA512	GET	1	3	Returned password is base64 SHA512	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux &	Step 1: Postman POST application/json '{"password":"angryhuman"}' http://127.0.0.1:8088/hash Step 2: Postman GET "application/json" http://127.0.0.1:8088/hash/<job id>	2. returned hash is 86 bytes or 88 bytes with padding	P	
7	Empty password is not allowed	POST	2	3	Empty passwords generate error messages	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux &	Step 1: Postman POST application/json '{"password":""}' http://127.0.0.1:8088/hash	1. 400 response	F	1. 200 is returned, along with a jobid. A subsequent get returns a hashed password that looks like all the others
8	Use username and empty string	POST	2	3	username and empty string are not valid	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux &	Step 1: Postman POST application/json '{"":"monkeymonkey"}' http://127.0.0.1:8088/hash Step 2: Postman POST application/json '{"username":"monkeymonkey"}' http://127.0.0.1:8088/hash Step 3: Postman POST application/json '{"":""}' http://127.0.0.1:8088/hash	1. 400 2. 400 3. 400 4. 400 5. 400	F	1. 200 response, job id is returned, subsequent GET with job id returns a hashed password 2. 200 response, job id is returned, subsequent GET with job id returns a hashed password 3. 200 response, job id is returned, subsequent GET with job id returns a hashed password

Test ID	Short Desc of Requirement	Category/Module (use for sorting purposes)	Priority	Requirement ID	Purpose	PreRequisite	Procedure	Expected Results	System test Passed	Actual Result
9	Password length tests	POST GET	1	3	Passwords can be long	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux &	Step 1: Postman POST application/json '{"password":"<16 chars>"}' http://127.0.0.1:8088/hash Step 2: Postman GET "application/json" http://127.0.0.1:8088/hash/<job id> Step 3: Repeat steps 1 and 2 with 1, 32, 64, 100, 1000 character length passwords	2. password is returned 3. password is returned	P	
10	Use username and password string	POST GET	2	3	Only a single JSON pair accepted	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux &	Step 1: Postman POST application/json '{"username":"lesshupe"," password":"monkeysee"}' http://127.0.0.1:8088/hashs	1. 400	F	1. 200 response, job id is returned, subsequent GET with job id returns a hashed password
11	Nonexistent ids should not return a hash	GET	1	3	Nonexistent ids should not return a hash	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux &	Step 1: Postman GET "application/json" http://127.0.0.1:8088/hash/<non existent job id>	1. 400, hash not found	P	
12	POST and GET with just URL does nothing	GET POST	2	3	POST and GET with just URL does nothing	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux &	Step 1: Postman POST application/json '{"password":"monkeysee"}' http://127.0.0.1:8088/ Step 2: Postman GET "application/json" http://127.0.0.1:8088/	1. 404	P	
13	POST and GET without JSON header fails	POST GET	2	3	POST and GET without JSON header fails	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux & 3. A valid job id already created.	Step 1: Postman POST '{"password":"monkeysee"}' http://127.0.0.1:8088/hash Step 2: Postman GET http://127.0.0.1:8088/<use vaild job id>	1. 404	F	1. 200 response, job id is returned, subsequent GET with job id returns a hashed password
14	Password encryption is consistent	POST GET	1	3	POST and GET creates same hash for same password	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux &	Step 1: Postman POST '{"password":"monkeysee"}' http://127.0.0.1:8088/hash Step 2: Postman POST '{"password":"monkeysee"}' http://127.0.0.1:8088/hash Step 3: Postman GET http://127.0.0.1:8088/<use vaild job id> for each job id created in steps 1 and 2	3. Hash returned is identical	P	
15	Password encryption can handle numbers and special characters	POST GET	1	3	Password encryption can handle numbers and special characters	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux &	Step 1: Postman POST '{"password":"12345678"}' http://127.0.0.1:8088/hash Step 2: Postman POST '{"password":"!@#$%^&*()_+<>,.;:+="}' http://127.0.0.1:8088/hash Step 3: Postman GET http://127.0.0.1:8088/<use vaild job id> for each job id created in steps 1 and 2	1. 200 2. 200 3. Hashed passwords returned.	P	

Test ID	Short Desc of Requirement	Category/Module (use for sorting purposes)	Priority	Requirement ID	Purpose	PreRequisite	Procedure	Expected Results	System test Passed	Actual Result
16	The software should be able to process multiple connections simultaneously	POST GET	1	4	Submit POST and GET requests nearly simultaneously	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux & 3. Shell script used, contents: curl -X POST -H 'application/json' -d '{"password":" angrymonkey"}' http://127.0.0.1:8088/hash & curl -X POST -H 'application/json' -d '{"password":" angrymonkey"}' http://127.0.0.1:8088/hash & ------- curl -H "application/json" http://127.0.0.1:8088/hash/1 & curl -H "application/json" http://127.0.0.1:8088/hash/3 & ------- curl -H "application/json" http://127.0.0.1:8088/hash/1 & curl -X POST -H 'application/json' -d '{"password":" angrymonkey"}' http://127.0.0.1:8088/hash &	Step 1: Postman POST two requests to http://127.0.0.1:8088/hash within a second using two Postman instances Step 2: Postman GET two requests to http://127.0.0.1:8088/hash within seconds using two Postman instances Step 3: Postman GET and POST requests to http://127.0.0.1:8088/hash within seconds using two Postman instances	1. POST successful, 200, job ids returned 2. GET successful, 200, hashed password returned 3. POST and GET successful, 200, jobid and hashed password returned	P	
17	A GET to /hash should accept a job identifier. It should return the base64 encoded password hash for the corresponding POST request.	POST GET	1	4	Old job ids still return same hashed passwords	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux & 3. Multiple POST's have occurred adding at least 3 hashed passwords. Record the returned hashed passwords.	Step1: Go back to the first job id, 1, or just go back or go back a minimum of 3 job ids and compare hashed passwords	1. Hashed passwords that are returned are still there.	P	
18	A GET to /stats should accept no data. It should return a JSON data structure for the total hash requests since the server started and the average time of a hash request in milliseconds	GET	2	3	GET to /stats with no passwords hashed	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux & 3. No hash requests after startup	Step 1: Postman GET "application/json" http://127.0.0.1:8088/hash/stats	1. 200, {"TotalRequests":0," AverageTime":0}	P	
19	A GET to /stats should accept no data. It should return a JSON data structure for the total hash requests since the server started and the average time of a hash request in milliseconds	GET	1	3	GET to /stats with one request	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux & 3. One hash requests after startup	Step 1: Postman GET "application/json" http://127.0.0.1:8088/hash/stats	1. 200, {"TotalRequests":1," AverageTime": <somevalue>}, somevalue is <= 5 seconds	F	Single request resulted in average time of 343415. If in milliseconds that is 343 seconds. Too high to be valid.
20	A GET to /stats should accept no data. It should return a JSON data structure for the total hash requests since the server started and the average time of a hash request in milliseconds	GET	1	3	GET to /stats multiple requests	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux & 3. 10 hash requests after startup	Step 1: Postman GET "application/json" http://127.0.0.1:8088/hash/stats	1. 200, {"TotalRequests": 10,"AverageTime": <somevalue>} somevalue is < 5 seconds	F	Multiple requests resulted in average time of 127206. If in milliseconds that is 127 seconds. Too high to be valid.
21	A GET to /hash should accept a job identifier	GET	2	3	A GET to /hash with characters for id should fail	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux & 3. At least one hash request after startup	Step 1: Postman GET "application/json" http://127.0.0.1:8088/hash/id Step 2: Postman GET "application/json" http://127.0.0.1:8088/hash/id1	1. 400, error message about id 2. 400, error message about id	P	

Test ID	Short Desc of Requirement	Category/Module (use for sorting purposes)	Priority	Requirement ID	Purpose	PreRequisite	Procedure	Expected Results	System test Passed	Actual Result
22	The software should support a graceful shutdown request	POST	1	5	POST to shutdown is successful	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux & 3. Valid job id created by submitting password	Step 1: curl -X POST -d 'shutdown' http://127.0.0.1:8088/hash Step 2: Postman GET request to http://127.0.0.1:8088/hash with valid job id Step 3: ps -ef | grep hash	1. POST successful, 200 response, no other data from server 2. GET fails 3. process is not found	F	response from shutdown did not include a 200 Only got: curl: (52) Empty reply from server
23	No additional password requests should be allowed when shutdown is pending	POST	1	5	Send password request immediately after shutdown request	1. PORT=8088 2. Application is running: nohup ./broken-hashserve_linux & 3. Use shell script: #!/bin/bash OUT=$(curl -X POST -d "shutdown" http://127.0.0.1:8088/hash &) curl -X POST -H 'application/json' -d '{"password":"angrymonkey"}' http://127.0.0.1:8088/hash & echo $OUT	Step 1: Execute Shell script: #!/bin/bash OUT=$(curl -X POST -d "shutdown" http://127.0.0.1:8088/hash &) curl -X POST -H 'application/json' -d '{"password":"angrymonkey"}' http://127.0.0.1:8088/hash & echo $OUT	1. Shutdown successful, no reply from server 2. POST receives connection refused	P	
24	The software should support a graceful shutdown request. Meaning, it should allow any in-flight password hashing to complete, reject any new requests, respond with a 200 and shutdown	POST	1	5	it should allow any in-flight password hashing to complete, reject any new requests, respond with a 200 and shutdown.	1. PORT=8088 2. Application is started up with no hashes submitted: nohup ./broken-hashserve_linux & 3. Use a shell script: #!/bin/bash curl -X POST -H 'application/json' -d '{"password":"angrymonkey"}' http://127.0.0.1:8088/hash & OUT=$(curl -X POST -d "shutdown" http://127.0.0.1:8088/hash &) echo $OUT	Step 1: Run shell script. Process is shutdown but job id is still returned.	1. POST successful, id is returned, 200 2. Shutdown successful 3. GET unsuccessful, 200	P	
25	A POST to /hash should accept a password. It should return a job identifier immediately. It should then wait 5 seconds and compute the password hash.	POST GET	2	3	Job id cannot be used to get Hash less than five seconds after request	1. PORT=8088 2. Application is started up and then one hash submitted: nohup ./broken-hashserve_linux & 3. The job id referenced in the shell script is the one undergoing processing by broken-hash 3. Shell script used: #!/bin/bash curl -X POST -H 'application/json' -d '{"password":"angrymonkey"}' http://127.0.0.1:8088/hash & curl -H "application/json" http://127.0.0.1:8088/hash/2 &	Step 1: Execute Shell script: #!/bin/bash curl -X POST -H 'application/json' -d '{"password":"angrymonkey"}' http://127.0.0.1:8088/hash & curl -H "application/json" http://127.0.0.1:8088/hash/2 &	1. Hash not found, then job id returned for that hash	P	

Test ID	Short Desc of Requirement	Category/Module (use for sorting purposes)	Priority	Requirement ID	Purpose	PreRequisite	Procedure	Expected Results	Sys tem test Pas sed	Actual Result
26	capacity of 100 can be stored test	POST	3	N/A	100 job ids can be stored	1. PORT=8088 2. Shell script used: #!/bin/bash COUNTER=0 while [$COUNTER -lt 100]; do echo The counter is $COUNTER curl -X POST -H 'application/json' -d '{"password":"$RANDOM"}' http://127.0.0.1:8088/hash let COUNTER=COUNTER+1 done	Step 1: Execute Shell script Step 2: retrieve hash job id 100	1. Shell script completes 2. Hash id returned for job id 100	P	
27	load test	POST	3	N/A	Send multiple requests in succession to see how many near simultaneous requests can be handled	1. PORT=8088 2. Shell script used: #!/bin/bash COUNTER=0 while [$COUNTER -lt 100]; do echo The counter is $COUNTER curl -X POST -H 'application/json' -d '{"password":"$RANDOM"}' http://127.0.0.1:8088/hash & curl -X POST -H 'application/json' -d '{"password":"$RANDOM"}' http://127.0.0.1:8088/hash & curl -X POST -H 'application/json' -d '{"password":"$RANDOM"}' http://127.0.0.1:8088/hash & curl -X POST -H 'application/json' -d '{"password":"$RANDOM"}' http://127.0.0.1:8088/hash & curl -X POST -H 'application/json' -d '{"password":"$RANDOM"}' http://127.0.0.1:8088/hash & sleep 2 let COUNTER=COUNTER+1 done	Step 1: Execute Shell script, GET a job id and confirm hash returned Step 2: Modify shell script to run 10 at time and limit the counter to 20, GET a job id and confirm hash returned Step 3: Modify shell script to run 30 at a time with limit 20, GET a job id and confirm hash returned Step 4: Modify shell script to run 40 at a time with limit 20 and sleep only 1 second, GET a job id and confirm hash returned Step 5: Modify shell script for 40 at a time, no sleeps, loop 20 times, GET a job id and confirm hash returned	1. Shell script completes and job ids returned, spot check of job ids returns password hash 2. Shell script completes and job ids returned, spot check of job ids returns password hash 3. Shell script completes and job ids returned, spot check of job ids returns password hash 4. Shell script completes and job ids returned, spot check of job ids returns password hash 5. Shell script completes and job ids returned, spot check of job ids returns password hash	P	