DAFTAR PUSTAKA

Akbulut, A., & Perros, H. G. (2019). Performance Analysis of Microservices Design Patterns. JOURNAL OF IEEE INTERNET COMPUTING.

Aksakalli, I.K., Turgay C., Can A.B., Bedir, T. (2021). Deployment and communication patterns in microservice architectures: A systematic literature review. The Journal of Systems & Software 180 (2021) 111014

Amazon, Tampa tahun. Amazon Elastic Compute Cloud User Guide for Linux Instances (online). https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-ug.pdf diakses pada 11 Maret 2022.

Araújo, M., Maia, M., Rego, P., & Souza J.D. (2020). Performance analysis of computational offloading on embedded platforms using the gRPC framework. 8th International Workshop on AD-VANCEs in ICT Infrastructures and Services

Barakat, S. A. (2017). Monitoring and Analysis of Microservices Performance. Journal of Computer Science and Control Systems Volume 10, Number 1, May 2017

Belshe, M., BitGo, Peon, R., Google, Thomson, M. & Mozila 2015. Hypertext transfer protocol version 2 (http/2) (online). https://datatracker.ietf.org/doc/html/rfc7540 diakses 20 Maret 2022.

Berge, S. P., Hagaseth, M., & Bø, T. I. (2019). A Public API Supporting Autonomous Navigation. Journal of Physics: Conference Series, Volume 1357, International Maritime and Port Technology and Development Conference and International Conference on Maritime Autonomous Surface Ships 13–14 November 2019, Trondheim, Norway.

Cao, D. 2016. Mobile Benchmarks (online). https://grpc.io/blog/mobile-benchmarks/ diakses pada 10 Maret 2022.

Carlson, J.L. 2013. Redis in Action. Manning publication. New York. US.

Chamas, C.L., Cordeiro, D., & Eler, M.M. (2017). Comparing REST, SOAP, Socket and gRPC in computation offloading of mobile applications: an energy cost analysis. IEEE 9th Latin-American Conference on Communications.

Das, V. 2015 . Learning Redis -Design efficient web and business solutions with Redis. PACKT Publishing. Birmingham UK.

Doglio, F. 2018. REST API Development with Node.js - Manage and Understand the Full Capabilities of Successful REST Development Second Edition. Apress. Canelones, Uruguay.

Donovan A.A.A. & Kernighan B.W. 2016. The Go Programming Language. Addison-wesley. Indianapolis. United States.

Erinle, B. 2017. Performance Testing with JMeter 3. Third Edition. Packt Publishing. Birmingham UK.

Grigorik, I. & Surma. 2019. Introduction to Http/2 (online). https://developers.google.com/web/fundamentals/performance/http2 diakses 02 Maret 2021.

Gonzalez, I. (2019). Building Microservice APIs Using GRPC. Thesis. California State University, Northridge.

Indrasiri, K. & Kuruppu D. 2020. gRPC Up & Running Building Cloud Native Applications with Go and Java for Docker and Kubernetes. O'reilly. California, US.

Havebeke, M. 2018. Eloquent JavaScript third edition.

Masse, M. 2012. REST API Design Rulebook. O'reilly. California , United States.

Matam, S. & Jain, j. 2017 . Pro Apache JMeter: Web Application Performance Testing. Apress. New york. USA.

McPeak, J. 2015. Beginning JavaScript. John wiley & sons. Indianapolis, US.

Meyerson, J. 2014. The Go Programming Language. IEEE Software (Volume: 31, Issue: 5, Sept.-Oct. 2014)

Nelson, B.J. 1981. Remote procedure call. Xerox Palo Alto Research Center. California. US.

Prayogi, A.A., Niswar, M., Indrabayu & Rijal, M. (2019). Design and Implementation of REST API for Academic Information System. The 3rd EPI International Conference on Science and Engineering 2019 (EICSE 2019)

Raharjo, B. 2019. Pemrograman web dengan Node.js dan JavaScript. Penerbit informatika. Bandung. Indonesia.

Rebrošová, B.P. (2021). gRPC Layer for Content Delivery in Kentico Kontent. Thesis. Masaryk University. Brno, Ceko.

Richardson, C. 2019. Microservices patterns: with examples in Java. Manning Publications. New York. US.

Saini, R. & Behl R. (2020). An Introduction to AWS – EC2 (Elastic Compute Cloud). Proceedings of the International Conference on Research in Management & Technovation pp. 99–102. DOI: 10.15439/2020KM4

Shafabakhsh, B. (2020). Research on Interprocess Communication in Microservices Architecture. Thesis. KTH Royal Institute Of Technology School Of Electrical Engineering And Computer Science. Stockholm, Sweden.

Shafabakhsh, B., Lagerström, R., & Hacks S. (2020). Evaluating the Impact of Inter Process Communication in Microservice Architectures. 8th International Workshop on Quantitative Approaches to Software Quality

Stefanic, M. (2021), Developing the guidelines for migration from RESTful microservices to gRPC. Thesis. Masaryk University. Brno, Ceko.

LAMPIRAN

Lampiran 1 Hasil pengukuran *response time*

Fungsi Create *service* KRS yang menggunakan REST dan gRPC

Percobaan	Response time (ms)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	234.61	347.58	1056.62	1628.82	237.89	383.8	658.79	1627.26
2	222.14	431.92	1037.13	1543.16	216.04	417.78	876.91	1466.72
3	223.8	342.98	900.7	1735.58	210.47	343.08	817.4	1447.4
4	219.12	429.06	807.95	1787.05	206.38	377.14	881.84	1685.98
5	207.67	414.78	990.53	1668.92	225.72	381.83	818.87	1552.23
6	221.41	469.7	854.87	1606.02	201.54	404.08	746.86	1509.32
7	180.61	352.44	911.69	1765.48	191.96	376.04	867.89	1514.95
8	230.27	377.98	755.93	1692.12	214.4	364.69	750.53	1419.63
9	231.02	352.73	892.21	1604.77	221.71	320.96	752.92	1302.72
10	209.44	405.49	760.34	1476.13	195.61	404.74	828.97	1631
Rata-rata	**218**	**392.46**	**896.79**	**1650.8**	**212.17**	**377.41**	**800.09**	**1515.72**

Fungsi Read *service* KRS yang menggunakan REST dan gRPC

Percobaan	Response time (ms)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users

1	209.09	382.33	787.94	1546.2	197.96	386.69	791.24	1601.34
2	224.82	409.49	721.12	1542.44	236	381.99	921.24	1476.47
3	193.25	387.95	848.88	1556.03	212.91	356.21	678.65	1541.51
4	191.09	406.53	740.42	1462.98	225.14	397.22	799.88	1479.85
5	192.49	318.54	722.08	1367.88	193.91	319.29	809.87	1415.79
6	193.03	354.18	886.24	1644.49	226.84	295.91	659.79	1679.81
7	194.3	363.82	800.49	1705.12	207.53	312.4	828.4	1248.12
8	198.3	388.69	788.24	1531.2	207.82	400.57	809.84	1630.28
9	209.99	319.39	749.76	1579.4	221.83	343.33	729.88	1571.07
10	209.12	354.91	872.63	1809.04	213.96	401.49	740	1099.66
Rata-rata	**201.54**	**368.582**	**791.78**	**1574.47**	**214.38**	**359.51**	**776.87**	**1474.38**

Fungsi Update *service* KRS yang menggunakan REST dan gRPC

Percobaan	Response time (ms)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	224.56	399.47	903.07	1627.21	186.01	352.82	793.8	1429.28
2	219.44	400.16	803.68	1739	221.49	374.96	872.13	1497.26
3	240.1	427.24	794.78	3311.37	204.12	390.11	843	1565.34
4	191.85	434.63	846.94	1794.78	209.08	413.25	896.73	1428.67
5	221.42	440.77	892.07	1831.6	189.74	381.05	776.42	1495.89
6	191.74	427.08	788.41	1515.05	205.43	405.76	924.94	1524
7	270.45	408.51	863.33	1750.32	192.64	365.42	779.32	1415.38
8	201.05	408.57	928.81	3426.38	230.02	367.05	734.5	1519.79

9	227.52	379.39	770.97	1518.77	213.22	406.14	732.77	1476.57
10	232.47	425.16	920.17	1611.43	206.29	404.35	757.92	1582.26
Rata-rata	**222.06**	**415.09**	**851.22**	**1673.52**	**205.8**	**386.09**	**811.15**	**1493.44**

Fungsi Delete *service* KRS yang menggunakan REST dan gRPC

Percobaan	Response time (ms)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	197.96	427.84	734.22	1754	216.72	355.94	955.69	1401.8
2	236	338.89	884.47	1426.81	207.82	435.58	863.18	1678.48
3	212.91	343.09	955.06	1548.66	215.3	385.83	873.91	1619.52
4	225.14	432.45	790.34	1551.65	200.06	338.63	763.34	1349.35
5	193.91	440.68	754.99	1440.2	210.96	412.55	736.36	1500.1
6	226.84	445.91	745.8	1647.06	166.77	388.16	789.5	1285.09
7	207.53	407.64	794.76	1705.38	199.27	401.41	867.24	1520.28
8	207.82	427.13	810.32	1435.93	216.64	344.76	751.35	1697.87
9	221.83	348.88	1043.47	1620.94	229.24	415.71	852.33	1846.44
10	213.96	368.86	895.01	1597.09	181.83	361.33	974.28	1423.25
Rata-rata	**214.38**	**398.137**	**840.84**	**1572.77**	**204.46**	**383.98**	**842.71**	**1532.21**

Fungsi Login *service* Auth yang menggunakan REST dan gRPC

Percobaan	Response time (ms)	
	REST	gRPC

	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	205.7	403.39	950.75	1450.7	197.79	324.97	821.67	1247.05
2	217.51	357.69	1063.97	1499.61	209.16	388.34	698.5	1359.89
3	223.33	330.05	795.67	1532.2	200.33	381.34	683.45	1336.69
4	224.87	376.84	870.6	1693.97	175.66	405.38	791.36	1278.45
5	210.72	394.72	677.25	1455.76	212.28	377.58	842.9	1271.81
6	194.62	386.71	823.32	1284.23	201.83	337.62	704.83	1287.72
7	182.06	402.71	834.99	1589	200.04	330.94	695.39	1150.9
8	211.71	408.74	839.9	1527.01	183.34	364.31	612.44	1429.17
9	202.01	397	850.97	1553.14	243.71	374.67	680.42	1454.17
10	211.42	358.89	793.21	1658.73	187.52	381.2	765.47	1347.61
Rata-rata	**208.39**	**381.67**	**850.06**	**1524.43**	**201.166**	**366.63**	**729.64**	**1316.34**

Fungsi Login *service* Auth yang menggunakan REST dan gRPC

Percobaan	Response time (ms)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	194.91	428.31	882.11	1461.48	173.95	372.33	849.33	1368.28
2	185.37	381.67	861.05	1719.7	184.06	343.35	796.19	1495.32
3	205.31	377.84	672.38	2015.11	230.41	363.48	630.51	1219.16
4	199.94	399.05	779.1	1336.86	173.05	351.3	894.36	1542.37
5	182.92	388.15	903.49	1473.31	180.75	336.34	805.87	1077.05
6	201.61	406.56	759.87	1417.45	181.02	347.5	778.04	1226.77

7	174.6	338.12	751.58	1634.66	166.19	410.36	872.36	1489.95
8	203.62	376.98	877.31	1316.36	187.51	328.54	798.16	1171.04
9	218.09	330.6	841.79	1634.91	179.58	355.01	863.24	1315.29
10	227.06	393.96	850.93	1607.11	182.99	297.97	710.27	1198.18
Rata-rata	**199.3429**	**382.12**	**817.961**	**1561.69**	**183.951**	**350.61**	**799.83**	**1310.34**

Fungsi Pay service Payment yang menggunakan REST dan gRPC

Percobaan	Response time (ms)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	197.78	388.81	920.78	1416.89	219.65	367.47	686.61	1142.06
2	201.96	329	943.06	1772.07	187.86	271.66	703.27	1237.8
3	183.58	314.54	845.69	1626.25	186.63	295.77	904.93	1157.56
4	229.49	341.24	741.08	1645.55	204.9	303.34	767.55	1405.15
5	213.38	342.18	753.87	1421.16	160.73	329.13	645.47	1373.14
6	237.97	355.73	772.13	1589.45	205.4	376.12	699.81	1468.76
7	204.03	413.1	799.91	1536.16	180.59	303.77	677.72	1217.56
8	202.2	391.11	889.59	1666.94	213.31	380.07	688.49	1269.52
9	187.44	387.33	981.36	1682.9	213.85	417.3	851.09	1300.52
10	212.07	360.15	968.33	1587.64	188.39	384.39	730.32	1461.08
Rata-rata	**206.99**	**362.31**	**861.57**	**1594.5**	**196.131**	**342.9**	**735.52**	**1303.31**

Fungsi Verify service Payment yang menggunakan REST dan gRPC

Percobaan	Response time (ms)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	187.57	360.29	844.15	1534.95	201.66	316.77	755.74	1284.57
2	211.69	381.44	824.92	1535.33	174.01	413.99	771.27	1482.8
3	163.92	398.17	943.95	1483.06	204.36	333.11	777.51	1106.98
4	189.31	402.38	742.97	1696.53	237.05	354.53	750.25	1414.85
5	198.77	372.5	844.16	1677.83	206.31	382.37	766.31	993.52
6	198.02	343.55	906.24	1340.59	204.81	337.34	685.21	1348.15
7	209.25	420.87	879.21	1627.32	180	308.67	849.56	1413.84
8	218.92	310	742.64	1665.06	200.17	312.29	869.96	1303.63
9	159.48	317.81	817.13	1515.17	218.85	370.5	894.54	1331.62
10	226.14	385.29	713.41	1451.05	182.9	408.3	814.45	1328.06
Rata-rata	**196.307**	**369.23**	**825.87**	**1552.68**	**201.012**	**353.78**	**793.48**	**1300.8**

Lampiran 2 Hasil pengukuran *throughput*

Fungsi Create service KRS yang menggunakan REST dan gRPC

Percobaan	Throughput (tps)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	403.23	500	444.05	508.39	395.26	470.59	652.74	555.86
2	389.11	435.73	456.62	528.26	446.43	431.03	505.56	625

3	395.26	484.26	490.68	481.46	434.78	533.33	534.76	647.67
4	406.5	352.11	551.27	530.5	431.03	497.51	525.76	550.06
5	414.94	439.56	478.01	587.54	413.22	512.82	530.22	553.4
6	343.64	346.02	536.48	529.66	427.35	470.59	657.89	593.12
7	369	505.05	505.05	472.81	350.88	480.77	496.03	609.76
8	406.5	492.61	508.13	550.96	436.68	497.51	589.62	623.83
9	338.98	466.2	491.16	542.3	425.53	602.41	610.5	581.4
10	401.61	443.46	547.65	572.41	458.72	449.44	549.45	580.05
Rata-rate	**386.87**	**446.5**	**500.91**	**530.42**	**421.98**	**494.59**	**565.25**	**592.01**

Fungsi Read service KRS yang menggunakan REST dan gRPC

Percobaan	Throughput (tps)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	374.53	437.64	538.21	555.56	450.45	503.78	570.78	582.07
2	427.35	452.49	584.8	549.75	446.43	483.09	525.21	626.57
3	467.29	471.7	546.45	539.37	478.47	515.46	586.17	563.7
4	476.19	445.43	589.62	534.76	377.36	410.68	536.48	635.73
5	408.16	539.08	595.95	611.62	421.94	549.45	576.04	649.35
6	377.36	496.28	482.63	517.6	473.93	558.66	667.56	545.85
7	346.02	516.8	567.54	485.44	395.26	506.33	574.71	682.59
8	377.36	453.51	524.66	558.35	571.43	423.73	581.4	562.11
9	344.83	434.78	484.5	539.37	431.03	549.45	624.22	585.82
10	452.49	507.61	531.35	521.38	395.26	475.06	586.85	773.4

Rata-rata	**405.158**	**475.53**	**544.57**	**541.31**	**444.156**	**497.569**	**582.94**	**620.718**

Fungsi Update service KRS yang menggunakan REST dan gRPC

Percobaan	Throughput (tps)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	366.3	406.5	497.02	529.94	438.6	554.02	587.54	628.54
2	343.64	414.94	519.75	527.15	413.22	481.93	542.89	613.87
3	325.73	436.68	554.32	255.62	418.41	428.27	572.74	609.76
4	420.17	433.84	544.07	536.19	414.94	462.96	534.76	615.38
5	348.43	363.64	462.53	493.1	354.61	480.77	612	593.82
6	322.58	437.64	511.77	550.96	460.83	438.6	507.1	619.2
7	289.02	468.38	508.13	496.77	476.19	491.4	572.74	613.5
8	380.23	427.35	469.04	279.1	398.41	493.83	566.89	618.81
9	416.67	477.33	575.37	621.5	436.68	462.96	546.45	579.04
10	389.11	369	507.1	521.65	414.94	462.96	519.75	586.17
Rata-rata	**360.18**	**423.52**	**514.91**	**534.65**	**422.68**	**475.77**	**556.28**	**607.8**

Fungsi Delete service KRS yang menggunakan REST dan gRPC

Percobaan	Throughput (tps)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users

1	446.43	441.5	461.68	546.75	427.35	529.1	484.03	593.47
2	387.6	469.48	475.29	607.53	408.16	433.84	544.07	537.35
3	436.68	477.33	466.85	620.35	456.62	487.8	522.47	559.6
4	411.52	403.23	533.05	528.26	440.53	540.54	542.3	651.47
5	460.83	431.03	570.13	596.3	374.53	462.96	591.72	546.15
6	404.86	396.04	603.14	543.48	448.43	485.44	555.56	604.59
7	378.79	460.83	530.79	505.31	362.32	455.58	540.54	595.59
8	386.1	432.9	588.93	429.18	423.73	501.25	553.71	543.18
9	347.22	439.56	439.75	565.29	416.67	447.43	547.05	513.35
10	333.33	461.89	508.65	531.07	383.14	516.8	478.93	595.24
Rata-rata	**399.33**	**441.379**	**517.82**	**547.35**	**414.14**	**486.07**	**536.03**	**573.99**

Fungsi Login service Auth yang menggunakan REST dan gRPC

Percobaan	Throughput (tps)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	358.42	447.43	458.72	617.67	436.68	508.91	573.39	577.03
2	429.18	424.63	443.26	479.62	403.23	461.89	498.5	541.71
3	322.58	431.97	492.61	563.7	377.36	470.59	622.67	621.12
4	393.7	377.36	462.96	549.75	389.11	457.67	561.17	661.81
5	411.52	425.53	551.88	562.11	398.41	425.53	535.33	654.02
6	413.22	450.45	460.41	623.83	389.11	547.95	585.48	680.74
7	480.77	457.67	443.66	530.79	431.03	431.97	591.02	736.92
8	361.01	423.73	517.06	436.87	483.09	456.62	672.04	537.06

9	342.47	437.64	500.5	490.44	374.53	492.61	645.16	580.05
10	348.43	460.83	548.85	496.77	393.7	498.75	513.87	627.75
Rata-rata	**386.13**	**433.72**	**487.99**	**535.15**	**407.62**	**475.24**	**579.86**	**621.82**

Fungsi Verify service Auth yang menggunakan REST dan gRPC

Percobaan	Throughput (tps)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	438.6	370.37	524.66	639.8	423.73	490.2	551.88	654.02
2	467.29	453.51	558.66	559.91	507.61	526.32	576.04	600.96
3	425.53	483.09	643.5	415.8	362.32	506.33	678.43	696.86
4	375.94	480.77	541.13	617.28	540.54	508.91	515.46	616.14
5	429.18	409.84	485.44	630.52	471.7	503.78	544.07	805.15
6	454.55	473.93	550.06	596.3	480.77	508.91	582.07	649.77
7	393.7	500	593.82	558.35	534.76	380.95	536.48	553.1
8	370.37	479.62	553.1	647.67	476.19	544.96	546.45	729.39
9	357.14	469.48	512.82	551.27	389.11	512.82	554.94	668.45
10	341.3	472.81	550.66	516.26	401.61	557.1	625	719.42
Rata-rata	**405.36**	**459.34**	**551.38**	**573.31**	**458.83**	**504.02**	**571.08**	**669.32**

Fungsi Pay service Payment yang menggunakan REST dan gRPC

Percobaan	Throughput (tps)	
	REST	gRPC

	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	467.29	487.8	473.93	672.04	381.68	475.06	648.51	805.15
2	395.26	595.24	478.01	522.47	458.72	550.96	650.2	590.67
3	480.77	579.71	518.67	569.15	392.16	518.13	512.82	671.59
4	408.16	491.4	440.53	536.19	458.72	555.56	555.56	668
5	431.03	433.84	615.76	678.43	404.86	550.96	701.26	696.86
6	358.42	522.19	580.05	538.21	450.45	404.04	636.94	648.51
7	429.18	459.77	537.63	634.12	465.12	464.04	561.8	763.94
8	347.22	462.96	530.79	541.71	448.43	491.4	647.67	706.21
9	478.47	486.62	457.04	530.5	448.43	457.67	545.85	736.92
10	425.53	461.89	457.04	567.54	492.61	405.68	634.52	648.09
Rata-rata	**422.13**	**498.14**	**508.94**	**579.03**	**440.11**	**487.35**	**609.51**	**693.593**

Fungsi Verify service Payment yang menggunakan REST dan gRPC

Percobaan	Throughput (tps)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	362.32	516.8	557.41	559.28	367.65	584.8	561.8	678.43
2	367.65	491.4	539.96	590.32	462.96	383.14	574.05	585.14
3	361.01	470.59	487.8	618.43	354.61	533.33	570.78	746.83
4	450.45	476.19	583.43	554.02	344.83	515.46	579.37	653.59
5	427.35	492.61	528.54	544.07	454.55	441.5	567.54	702.25
6	456.62	555.56	510.73	652.32	370.37	416.67	653.59	628.54

7	406.5	410.68	559.28	560.22	497.51	617.28	534.19	645.58
8	364.96	569.8	616.52	573.39	347.22	557.1	542.3	678.43
9	436.68	510.2	538.21	616.9	366.3	516.8	493.58	665.34
10	400	489	649.35	634.52	434.78	385.36	589.62	672.04
Rata-rata	**403.35**	**498.28**	**557.12**	**590.34**	**400.07**	**495.14**	**566.68**	**665.61**

Lampiran 3 Hasil pengukuran CPU *utilization*

Fungsi Create *service* KRS yang menggunakan REST dan gRPC

Percobaan	CPU utilization (%)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	40	46	86	86	13	30	20	46
2	43	73	86	93	20	33	60	60
3	46	53	75	87	26	46	33	46
4	70	73	86	86	13	13	20	73
5	13	80	86	86	26	33	26	66
6	26	20	86	93	13	50	60	60
7	33	33	73	99	26	13	20	66
8	46	33	99	99	26	6	53	66
9	20	70	80	99	26	33	33	46
10	13	33	93	99	33	20	53	73
Rata-rate	**35**	**51.4**	**85**	**92.7**	**22.2**	**27.7**	**37.8**	**60.2**

Fungsi Read *service* KRS yang menggunakan REST dan gRPC

Percobaan	CPU utilization (%)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	53	26	93	99	26	46	26	73
2	40	66	80	99	20	33	46	66
3	50	60	86	93	60	40	26	73
4	40	13	73	93	20	20	46	73
5	53	73	86	93	13	40	60	60
6	13	73	93	99	40	26	26	60
7	46	50	80	93	20	26	40	47
8	40	60	80	99	26	33	70	73
9	26	26	99	93	33	26	53	68
10	37	33	80	93	6	40	46	73
Rata-rata	**39.8**	**48**	**85**	**95.4**	**26.4**	**33**	**43.9**	**66.6**

Fungsi Update *service* KRS yang menggunakan REST dan gRPC

Percobaan	CPU utilization (%)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	80	73	80	93	40	30	60	60
2	13	60	93	93	50	60	46	53
3	46	53	86	93	20	26	53	60
4	46	66	73	86	40	50	53	60

5	63	60	86	93	30	70	40	80
6	33	60	99	86	20	40	40	62
7	20	53	86	93	20	20	60	60
8	40	80	80	93	20	6	66	46
9	50	72	66	93	20	40	33	60
10	60	53	86	80	30	60	60	66
Rata-rata	**45.1**	**63**	**83.5**	**90.3**	**29**	**40.2**	**51.1**	**60.7**

Fungsi Delete *service* KRS yang menggunakan REST dan gRPC

Percobaan	CPU utilization (%)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	26	80	86	80	20	60	53	73
2	20	33	93	93	26	40	10	80
3	40	26	99	93	20	40	53	73
4	33	53	86	81	20	46	66	62
5	40	53	93	87	26	50	53	53
6	13	66	86	81	40	6	33	53
7	50	53	99	99	10	25	26	62
8	46	53	73	99	20	53	60	73
9	6	86	86	93	26	26	60	73
10	40	56	80	99	40	33	80	66
Rata-rata	**31.4**	**55.9**	**88.1**	**90.5**	**24.8**	**37.9**	**53.7**	**66.8**

Fungsi Login *service* Auth yang menggunakan REST dan gRPC

Percobaan	CPU utilization (%)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	53	33	99	99	40	99	66	99
2	60	80	93	99	53	40	90	99
3	13	73	99	99	53	53	53	99
4	66	60	99	99	40	90	93	99
5	99	93	99	99	80	60	40	99
6	13	33	99	99	46	40	99	99
7	13	60	99	99	46	60	26	73
8	66	46	99	99	40	80	93	93
9	93	46	99	99	33	13	40	99
10	13	33	99	99	40	80	86	99
Rata-rata	**48.9**	**55.7**	**98.4**	**99**	**47.1**	**61.5**	**68.6**	**95.8**

Fungsi Verify *service* Auth yang menggunakan REST dan gRPC

Percobaan	CPU utilization (%)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	53	20	99	99	33	46	68	99
2	6	93	93	99	26	90	46	60
3	60	66	99	99	26	40	20	80
4	66	66	99	99	20	60	26	66

5	60	70	99	99	40	40	86	99
6	40	93	93	99	26	33	66	99
7	13	33	99	99	50	53	53	99
8	55	66	99	99	20	90	66	73
9	33	80	99	99	30	53	80	80
10	50	20	93	99	50	40	99	99
Rata-rata	**43.6**	**60.7**	**97.2**	**99**	**32.1**	**54.5**	**61**	**85.4**

Fungsi Pay *service* Payment yang menggunakan REST dan gRPC

Percobaan	CPU utilization (%)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	13	23	46	60	30	33	40	60
2	6	33	53	40	10	13	33	33
3	9	40	46	60	18	33	26	33
4	26	50	26	46	30	12	40	40
5	20	26	46	40	6	13	33	53
6	30	33	26	53	30	26	40	40
7	10	60	26	46	6	6	40	50
8	26	20	23	60	13	20	26	46
9	13	46	46	66	20	40	20	33
10	26	33	40	46	20	13	33	46
Rata-rata	**17.9**	**36.4**	**37.8**	**51.7**	**18.3**	**20.9**	**33.1**	**43.4**

Fungsi Verify *service* Payment yang menggunakan REST dan gRPC

Percobaan	CPU utilization (%)							
	REST				gRPC			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	13	20	46	46	6	30	33	33
2	20	20	53	46	20	30	18	33
3	20	40	40	53	13	30	46	33
4	6	30	53	46	13	20	30	20
5	13	20	33	53	13	6	13	53
6	13	33	40	46	20	6	50	40
7	6	6	46	46	6	30	40	40
8	20	33	40	53	13	30	40	40
9	13	33	33	53	20	20	30	40
10	13	13	40	53	30	20	26	43
Rata-rata	**13.7**	**24.8**	**42.4**	**49.5**	**15.4**	**22.2**	**32.6**	**37.5**

Lampiran 4 Hasil pengukuran *response time* antara Node.js dan Go

Fungsi Login *service* Auth menggunakan REST

Percobaan	Response time (ms)							
	Node.js				Go			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	205.7	403.39	950.75	1450.7	208.43	346.22	720.58	1396.52
2	217.51	357.69	1063.97	1499.61	206	371.74	858.18	1613.34

3	223.33	330.05	795.67	1532.2	175.44	400.29	832.11	1641.52
4	224.87	376.84	870.6	1693.97	209.16	355.34	827.63	1403.03
5	210.72	394.72	677.25	1455.76	183.9	388.35	854.82	1567.72
6	194.62	386.71	823.32	1284.23	198	325.45	694.68	1339.08
7	182.06	402.71	834.99	1589	188.11	374.5	741.55	1381.41
8	211.71	408.74	839.9	1527.01	189.13	387.12	918.75	1674.85
9	202.01	397	850.97	1553.14	207.65	317.21	963.61	1641.99
10	211.42	358.89	793.21	1658.73	193.05	401.64	956.62	1335.22
Rata-rata	**208.39**	**381.67**	**850.06**	**1524.43**	**195.887**	**366.785**	**836.853**	**1499.46**

Fungsi Verify *service* Auth menggunakan REST

Percobaan	Response time (ms)							
	Node.js				Go			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	194.91	428.31	882.11	1461.48	195.09	404.59	839.22	1354.91
2	185.37	381.67	861.05	1719.7	170.46	345.81	736.19	1505.11
3	205.31	377.84	672.38	2015.11	189.19	359.52	780.79	1653.74
4	199.94	399.05	779.1	1336.86	167.22	417.33	852.06	1644.44
5	182.92	388.15	903.49	1473.31	166.46	435.8	726.54	1360.54
6	201.61	406.56	759.87	1417.45	216.02	381.52	739.58	1457.22
7	174.6	338.12	751.58	1634.66	189.91	366.73	821.09	1426.85
8	203.62	376.98	877.31	1316.36	192.74	423.47	836.18	1577.65
9	218.09	330.6	841.79	1634.91	170.05	354.77	809.21	1474.68
10	227.06	393.96	850.93	1607.11	165.78	355.31	933.31	1475.18

Rata-rata	**199.342**	**382.12**	**817.961**	**1561.69**	**182.292**	**384.484**	**807.417**	**1493.03**

Fungsi Login *service* Auth menggunakan gRPC

Percobaan	Response time (ms)							
	Node.js				Go			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	197.79	324.97	821.67	1247.05	157	276.19	695.54	1477.87
2	209.16	388.34	698.5	1359.89	202.86	309.94	651.73	1592.77
3	200.33	381.34	683.45	1336.69	196.86	341.43	809.02	1296.9
4	175.66	405.38	791.36	1278.45	203.08	329.94	701.83	1230.59
5	212.28	377.58	842.9	1271.81	193.84	296.52	750.28	1386.28
6	201.83	337.62	704.83	1287.72	186.68	321.37	704.2	1465.43
7	200.04	330.94	695.39	1150.9	198.7	346.95	861.32	1520.86
8	183.34	364.31	612.44	1429.17	197.02	313.63	730.03	1482.71
9	243.71	374.67	680.42	1454.17	146.95	349.43	911.71	1458.58
10	187.52	381.2	765.47	1347.61	185.64	335.76	772.14	1277.89
Rata-rata	**201.166**	**366.63**	**729.64**	**1316.34**	**186.863**	**322.116**	**758.78**	**1418.98**

Fungsi Verify *service* Auth menggunakan gRPC

Percobaan	Response time (ms)							
	Node.js				Go			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users

1	173.95	372.33	849.33	1368.28	197.1	357.94	876.76	1272.16
2	184.06	343.35	796.19	1495.32	189.26	367.92	712.68	1487.34
3	230.41	363.48	630.51	1219.16	212.37	326.03	692.96	1204.87
4	173.05	351.3	894.36	1542.37	187.27	358.9	848.1	1363.88
5	180.75	336.34	805.87	1077.05	175.63	357.83	845.46	1598.49
6	181.02	347.5	778.04	1226.77	194.06	297.29	751.56	1322.4
7	166.19	410.36	872.36	1489.95	179.44	400.35	803.72	1474.04
8	187.51	328.54	798.16	1171.04	192.86	356.68	721.59	1102.36
9	179.58	355.01	863.24	1315.29	204.87	383.08	800.87	1372.6
10	182.99	297.97	710.27	1198.18	180.54	290.59	801.67	1309.86
Rata-rata	**183.951**	**350.61**	**799.83**	**1310.34**	**191.34**	**349.661**	**785.537**	**1350.8**

Lampiran 5 Hasil pengukuran *throughput* antara Node.js dan Go

Fungsi Login *service* Auth menggunakan REST

Percobaan	Throughput (ms)							
	Node.js				Go			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	358.42	447.43	458.72	617.67	367.65	510.2	618.81	628.93
2	429.18	424.63	443.26	479.62	444.44	493.83	539.96	563.7
3	322.58	431.97	492.61	563.7	487.8	460.83	511.25	565.61
4	393.7	377.36	462.96	549.75	460.83	487.8	553.71	604.59
5	411.52	425.53	551.88	562.11	421.94	490.2	478.01	535.91
6	413.22	450.45	460.41	623.83	429.18	453.51	496.52	584.8
7	480.77	457.67	443.66	530.79	373.13	428.27	576.04	706.71

8	361.01	423.73	517.06	436.87	361.01	468.38	490.68	542.01
9	342.47	437.64	500.5	490.44	444.44	519.48	477.1	565.93
10	348.43	460.83	548.85	496.77	462.96	430.11	485.91	621.89
Rata-rata	**386.13**	**433.72**	**487.99**	**535.15**	**425.337**	**474.260**	**522.799**	**592.008**

Fungsi Verify *service* Auth menggunakan REST

Percobaan	Throughput (ms)							
	Node.js				Go			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	438.6	370.37	524.66	639.8	446.43	461.89	531.35	653.59
2	467.29	453.51	558.66	559.91	390.63	490.2	572.74	621.12
3	425.53	483.09	643.5	415.8	469.48	508.91	509.16	573.07
4	375.94	480.77	541.13	617.28	537.63	437.64	531.35	565.61
5	429.18	409.84	485.44	630.52	389.11	366.97	618.05	586.17
6	454.55	473.93	550.06	596.3	366.3	476.19	646.83	599.52
7	393.7	500	593.82	558.35	480.77	496.28	508.13	639.39
8	370.37	479.62	553.1	647.67	400	415.8	561.8	581.73
9	357.14	469.48	512.82	551.27	438.6	492.61	560.54	565.29
10	341.3	472.81	550.66	516.26	518.13	475.06	504.54	550.96
Rata-rata	**405.36**	**459.34**	**551.38**	**573.31**	**443.707**	**462.155**	**554.449**	**593.645**

Fungsi Login *service* Auth menggunakan gRPC

Percobaan	Throughput (ms)							
	Node.js				Go			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	436.68	508.91	573.39	577.03	507.61	533.33	593.82	585.14
2	403.23	461.89	498.5	541.71	389.11	606.06	674.76	581.06
3	377.36	470.59	622.67	621.12	473.93	477.33	551.27	724.11
4	389.11	457.67	561.17	661.81	406.5	505.05	623.44	775.19
5	398.41	425.53	535.33	654.02	340.14	611.62	593.82	665.78
6	389.11	547.95	585.48	680.74	444.44	526.32	644.33	634.12
7	431.03	431.97	591.02	736.92	377.36	512.82	531.91	615.38
8	483.09	456.62	672.04	537.06	478.47	476.19	604.59	604.96
9	374.53	492.61	645.16	580.05	500	438.6	523.56	610.5
10	393.7	498.75	513.87	627.75	502.51	464.04	621.12	731.53
Rata-rata	**407.62**	**475.24**	**579.86**	**621.82**	**442.007**	**515.136**	**596.262**	**652.776**

Fungsi Verify *service* Auth menggunakan gRPC

Percobaan	Throughput (ms)							
	Node.js				Go			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	423.73	490.2	551.88	654.02	392.16	530.5	516.53	740.74
2	507.61	526.32	576.04	600.96	500	490.2	658.76	577.7
3	362.32	506.33	678.43	696.86	369	571.43	660.5	738.01

4	540.54	508.91	515.46	616.14	471.7	467.29	555.56	657.89
5	471.7	503.78	544.07	805.15	523.56	455.58	545.85	576.7
6	480.77	508.91	582.07	649.77	374.53	568.18	553.71	644.33
7	534.76	380.95	536.48	553.1	469.48	462.96	564.97	603.5
8	476.19	544.96	546.45	729.39	460.83	514.14	618.05	697.35
9	389.11	512.82	554.94	668.45	431.03	476.19	576.04	557.1
10	401.61	557.1	625	719.42	546.45	526.32	553.1	737.46
Rata-rata	**458.83**	**504.02**	**571.08**	**669.32**	**453.874**	**506.279**	**580.307**	**653.078**

Lampiran 6 Hasil pengukuran CPU *utilization* antara Node.js dan Go

Fungsi Login *service* Auth menggunakan REST

Percobaan	CPU utilization (%)							
	Node.js				Go			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	53	33	99	99	20	26	53	53
2	60	80	93	99	26	13	53	53
3	13	73	99	99	20	40	40	40
4	66	60	99	99	20	40	46	60
5	99	93	99	99	26	20	40	46
6	13	33	99	99	13	13	53	53
7	13	60	99	99	40	33	50	53
8	66	46	99	99	20	46	53	60
9	93	46	99	99	13	33	53	60
10	13	33	99	99	26	6	46	53

Rata-rata	**48.9**	**55.7**	**98.4**	**99**	**22.4**	**27**	**48.7**	**53.1**

Fungsi Verify *service* Auth menggunakan REST

Percobaan	CPU utilization (%)							
	Node.js				Go			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	53	20	99	99	20	12	26	46
2	6	93	93	99	20	20	40	53
3	60	66	99	99	13	20	53	40
4	66	66	99	99	13	40	46	40
5	60	70	99	99	20	30	33	40
6	40	93	93	99	6	6	40	46
7	13	33	99	99	6	26	33	53
8	55	66	99	99	20	33	46	53
9	33	80	99	99	6	13	46	46
10	50	20	93	99	18	26	46	53
Rata-rata	**43.6**	**60.7**	**97.2**	**99**	**14.2**	**22.6**	**42.55**	**47**

Fungsi Login *service* Auth menggunakan gRPC

Percobaan	CPU utilization (%)							
	Node.js				Go			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	40	99	66	99	13	40	53	40

2	53	40	90	99	20	13	13	31
3	53	53	53	99	13	26	26	46
4	40	90	93	99	20	20	40	53
5	80	60	40	99	20	13	53	53
6	46	40	99	99	6	26	26	60
7	46	60	26	73	20	33	46	50
8	40	80	93	93	20	20	20	73
9	33	13	40	99	13	50	26	33
10	40	80	86	99	6	33	20	60
Rata-rata	**47.1**	**61.5**	**68.6**	**95.8**	**15.1**	**27.4**	**32.3**	**49.9**

Fungsi Verify *service* Auth menggunakan gRPC

Percobaan	CPU utilization (%)							
	Node.js				Go			
	100 users	200 users	500 users	1000 users	100 users	200 users	500 users	1000 users
1	33	46	68	99	13	20	33	40
2	26	90	46	60	13	20	40	26
3	26	40	20	80	30	20	20	33
4	20	60	26	66	6	20	26	40
5	40	40	86	99	20	18	26	44
6	26	33	66	99	13	30	20	26
7	50	53	53	99	13	25	20	33
8	20	90	66	73	13	6	40	33
9	30	53	80	80	6	20	30	40

10	50	40	99	99	13	20	13	40
Rata-rata	**32.1**	**54.5**	**61**	**85.4**	**14**	**19.9**	**26.8**	**35.5**

Lampiran 7 *Source code* pendefinisian file proto

File krs.proto

```
syntax = "proto3";
package proto;
option go_package = "proto/";

message MataKuliah {
  string kode = 1;
  string nama = 2;
  int32 sks = 3;
  string dosen = 4;
  string semester = 5;
}

message CreateUpdateKRSRequest {
   string token = 1;
   int32 id_mahasiswa = 2;
   repeated MataKuliah mata_kuliahs = 3;
}

message ReadKRSRequest {
   string token = 1;
   int32 id_mahasiswa = 2;
}

message DeleteKRSRequest {
   string token = 1;
   int32 id_mahasiswa = 2;
   int32 id_mata_kuliah = 3;
}

message KRSResponse {
   repeated  MataKuliah mata_kuliahs = 1;
}

message DeleteKRSResponse {
   string status = 1;
}
```

```
service KrsService {
   rpc Read(ReadKRSRequest) returns (KRSResponse);
   rpc Create(CreateUpdateKRSRequest) returns (KRSResponse);
   rpc Update(CreateUpdateKRSRequest) returns (KRSResponse);
   rpc Delete(DeleteKRSRequest) returns (DeleteKRSResponse);
}
```

File auth.proto

```
syntax = "proto3";
package proto;
option go_package = "proto/";

message LoginRequest {
  string username = 1;
  string password = 2;
}

message VerifyRequest {
  string token = 1;
}

message LoginResponse {
  string token = 1;
}

message VerifyResponse {
  bool is_auth = 1;
}

service AuthService {
  rpc Login(LoginRequest) returns (LoginResponse){};
  rpc Verify(VerifyRequest) returns (VerifyResponse){};
}
```

File payment.proto

```
syntax = "proto3";

package proto;
```

```
option go_package = "proto/";

message CreatePaymentRequest {
   int32 id_mahasiswa = 1;
   float jumlah = 2;
   string metode = 3;
}

message VerifyPaymentRequest {
   int32 id_mahasiswa = 1;
}

message PaymentResponse {
   bool is_pay = 1;
}

service PaymentService {
   rpc Create(CreatePaymentRequest) returns (PaymentResponse);
   rpc Verify(VerifyPaymentRequest) returns (PaymentResponse);
}
```

Lampiran 8 *Source code service* KRS

File route.go

```
package rest

import (
       "net/http"
       "github.com/dinel13/thesis-ac/krs/domain"
       "github.com/julienschmidt/httprouter"
)

func Routes(ch domain.KrsRestHandlers) http.Handler {
       r := httprouter.New()
       r.HandlerFunc(http.MethodGet, "/krs/:id", ch.Read)
       r.HandlerFunc(http.MethodPost, "/krs", ch.Create)
       r.HandlerFunc(http.MethodPut, "/krs/:id", ch.Update)
       r.HandlerFunc(http.MethodDelete, "/krs/:id", ch.Delete)
       return r
}
```

File redis.go

```
package repository

import (
	"context"
	"encoding/json"
	"strconv"
	"time"
	"github.com/dinel13/thesis-ac/krs/domain"
	"github.com/go-redis/redis/v8"
)

func NewKrsRepoRedisImpl(dbClient *redis.Client) domain.KrsRepository {
	return krsRepositoryRedisImpl{Rds: dbClient}
}

type krsRepositoryRedisImpl struct {
	Rds *redis.Client
}

func (m krsRepositoryRedisImpl) Create(ctx context.Context, krs
*domain.Krs) (*domain.Krs, error) {
	krsJson, err := json.Marshal(krs)
	if err != nil {
		return nil, err
	}

	err = m.Rds.Set(ctx, strconv.Itoa(krs.IdMahasiswa), krsJson,
24*time.Hour).Err()
	if err != nil {
		return nil, err
	}
	return krs, nil
}

func (m krsRepositoryRedisImpl) Read(ctx context.Context, id int)
(*domain.Krs, error) {
	val, err := m.Rds.Get(ctx, strconv.Itoa(id)).Bytes()
	if err != nil {
		return nil, err
	}
	krs := domain.Krs{}
	err = json.Unmarshal(val, &krs)
	if err != nil {
		return nil, err
	}
	return &krs, nil
```

```
}

func (m krsRepositoryRedisImpl) Update(ctx context.Context, krs
*domain.Krs) (*domain.Krs, error) {
	krsJson, err := json.Marshal(krs)
	if err != nil {
		return nil, err
	}

	err = m.Rds.Set(ctx, strconv.Itoa(krs.IdMahasiswa), krsJson,
24*time.Hour).Err()
	if err != nil {
		return nil, err
	}
	return krs, nil
}

func (m krsRepositoryRedisImpl) Delete(ctx context.Context, id int)
error {
	err := m.Rds.Del(ctx, strconv.Itoa(id)).Err()
	if err != nil {
		return err
	}
	return nil
}
```

File service.go

```
package service

import (
	"context"
	"time"

	"github.com/dinel13/thesis-ac/krs/domain"
)

func NewKrsService(Repo domain.KrsRepository) domain.KrsService {
	return DefaultKrsService{Repo}
}

type DefaultKrsService struct {
	Repo domain.KrsRepository
}
```

```
// GetKrs returns a krs by id
func (s DefaultKrsService) Read(token string, id int) (*domain.Krs,
error) {
	ctx, cancel := context.WithTimeout(context.Background(),
3*time.Second)
	defer cancel()

	c, err := s.Repo.Read(ctx, id)
	if err != nil {
		return nil, err
	}
	return c, nil
}

//	AddKrs adds a new krs
func (s DefaultKrsService) Create(krs *domain.Krs) (*domain.Krs, error)
{
	ctx, cancel := context.WithTimeout(context.Background(),
3*time.Second)
	defer cancel()

	c, err := s.Repo.Create(ctx, krs)
	if err != nil {
		return nil, err
	}
	return c, nil
}

//	UpdateKrs
func (s DefaultKrsService) Update(krs *domain.Krs) (*domain.Krs, error)
{
	ctx, cancel := context.WithTimeout(context.Background(),
3*time.Second)
	defer cancel()

	c, err := s.Repo.Update(ctx, krs)
	if err != nil {
		return nil, err
	}
	return c, nil
}

//	DeleteKrs
func (s DefaultKrsService) Delete(token string, id int) error {
	ctx, cancel := context.WithTimeout(context.Background(),
```

```
3*time.Second)
	defer cancel()

	err := s.Repo.Delete(ctx, id)
	if err != nil {
		return err
	}
	return nil
}
```

File handler.go untuk REST

```
package rest

import (
	"errors"
	"net/http"
	"strconv"

	"github.com/dinel13/thesis-ac/krs/domain"
	"github.com/julienschmidt/httprouter"
)

func NewCoursRestHandlers(s domain.KrsService) domain.KrsRestHandlers {
	return krsHandlers{s}
}

type krsHandlers struct {
	service domain.KrsService
}

// GetKrs is handler for GET /krs/{id}
func (h krsHandlers) Read(w http.ResponseWriter, r *http.Request) {
	// get the krs id from request param
	params := httprouter.ParamsFromContext(r.Context())
	id, err := strconv.Atoi(params.ByName("id"))
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}

	// GET TOKEN FROM HEADER USE BEARER TOKEN
	token := r.Header.Get("Authorization")
```

```
	isAuth, err := verifyToken(token)
	if err != nil {
		WriteJsonError(w, err, http.StatusBadRequest)
		return
	}
	if !isAuth {
		WriteJsonError(w, errors.New("token tidak valid"),
http.StatusBadRequest)
		return
	}

	isPay, err := verifyPayment(id)
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}
	if !isPay {
		WriteJsonError(w, errors.New("belum melakukan
pembayaran"), http.StatusBadRequest)
		return
	}

	// get the krs from service
	krs, err := h.service.Read(token, id)
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}

	// write the krs to response
	WriteJson(w, http.StatusOK, krs, "krs")
}

// Create is handler for POST /krs to create COurse
func (h krsHandlers) Create(w http.ResponseWriter, r *http.Request) {
	token := r.Header.Get("Authorization")

	isAuth, err := verifyToken(token)
	if err != nil {
		WriteJsonError(w, err, http.StatusBadRequest)
		return
	}
	if !isAuth {
		WriteJsonError(w, errors.New("token tidak valid"),
http.StatusBadRequest)
		return
```

```
	}

	// get the krs from request body
	krs := &domain.Krs{}
	err = ReadJson(r, krs)
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}

	isPay, err := verifyPayment(krs.IdMahasiswa)
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}
	if !isPay {
		WriteJsonError(w, errors.New("belum melakukan
pembayaran"), http.StatusBadRequest)
		return
	}

	// create the krs
	c, err := h.service.Create(krs)
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}

	// write the krs to response
	WriteJson(w, http.StatusCreated, c, "krs")
}

// Update is handler for PUT /krs to update Krs
func (h krsHandlers) Update(w http.ResponseWriter, r *http.Request) {
	// get the krs id from request param
	params := httprouter.ParamsFromContext(r.Context())
	id, err := strconv.Atoi(params.ByName("id"))
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}

	token := r.Header.Get("Authorization")

	isAuth, err := verifyToken(token)
	if err != nil {
```

```
			WriteJsonError(w, err, http.StatusBadRequest)
			return
		}
		if !isAuth {
			WriteJsonError(w, errors.New("token tidak valid"),
http.StatusBadRequest)
			return
		}

		isPay, err := verifyPayment(id)
		if err != nil {
			WriteJsonError(w, err, http.StatusInternalServerError)
			return
		}
		if !isPay {
			WriteJsonError(w, errors.New("belum melakukan
pembayaran"), http.StatusBadRequest)
			return
		}

		// get the krs from request body
		krs := &domain.Krs{}
		err = ReadJson(r, krs)
		if err != nil {
			WriteJsonError(w, err, http.StatusInternalServerError)
			return
		}
		krs.IdMahasiswa = id

		// update the krs
		c, err := h.service.Update(krs)
		if err != nil {
			WriteJsonError(w, err, http.StatusInternalServerError)
			return
		}

		// write the krs to response
		WriteJson(w, http.StatusOK, c, "krs")
}

// Delete is handler for DELETE /krs/{id} to delete Krs
func (h krsHandlers) Delete(w http.ResponseWriter, r *http.Request) {
		// get the krs id from request param
		params := httprouter.ParamsFromContext(r.Context())
		id, err := strconv.Atoi(params.ByName("id"))
		if err != nil {
```

```
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}

	token := r.Header.Get("Authorization")
	isAuth, err := verifyToken(token)
	if err != nil {
		WriteJsonError(w, err, http.StatusBadRequest)
		return
	}
	if !isAuth {
		WriteJsonError(w, errors.New("token tidak valid"),
http.StatusBadRequest)
		return
	}

	isPay, err := verifyPayment(id)
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}
	if !isPay {
		WriteJsonError(w, errors.New("belum melakukan
pembayaran"), http.StatusBadRequest)
		return
	}

	// delete the krs
	err = h.service.Delete(token, id)
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}

	// write the krs to response
	WriteJson(w, http.StatusOK, "success", "krs")
}
```

File handler.go untuk gRPC

```
package grpc

import (
	"context"
```

```
	"errors"
	"log"

	"github.com/dinel13/thesis-ac/krs/domain"
	"github.com/dinel13/thesis-ac/krs/proto"
)

func NewGrpcHandler(cu proto.AuthServiceClient, cp
proto.PaymentServiceClient, s domain.KrsService) domain.KrsGrpcHandler
{
	return grpcHandler{clientAuth: cu, clientPay: cp, service: s}
}

type grpcHandler struct {
	clientAuth proto.AuthServiceClient
	clientPay  proto.PaymentServiceClient
	service    domain.KrsService
}

// read is a method that implements the Read method of the
KrsGrpcHandler interface
func (h grpcHandler) Read(ctx context.Context, req
*proto.ReadKRSRequest) (*proto.KRSResponse, error) {
	token := req.GetToken()
	id_mahasiswa := req.GetIdMahasiswa()
	idMahasiswa := int(id_mahasiswa)

	isAuth, err := VerifyToken(ctx, h.clientAuth, token)
	if err != nil {
		log.Println(err)
		return nil, err
	}
	if !isAuth {
		log.Println("token is not valid")
		return nil, errors.New("token is not valid")
	}

	isPay, err := VerifyPayment(ctx, h.clientPay, idMahasiswa)
	if err != nil {
		log.Println(err)
		return nil, err
	}
	if !isPay {
		log.Println("belum melakukan pembayaran")
		return nil, errors.New("belum melakukan pembayaran")
	}
```

```
	// parse int32 to int64
	krs, err := h.service.Read(token, idMahasiswa)
	if err != nil {
		log.Println(err)
		return nil, err
	}

	// loop krs.MataKuliah to convert to proto.MataKuliah
	var mataKuliahs []*proto.MataKuliah
	for _, mataKuliah := range krs.MataKuliahs {
		mataKuliahs = append(mataKuliahs, &proto.MataKuliah{
			Kode:     mataKuliah.Kode,
			Nama:     mataKuliah.Nama,
			Sks:      int32(mataKuliah.Sks),
			Dosen:    mataKuliah.Dosen,
			Semester: mataKuliah.Semester,
		})
	}

	res := &proto.KRSResponse{
		MataKuliahs: mataKuliahs,
	}

	return res, nil
}

// Create is a method that implements the Create method of the
KrsGrpcHandler interface
func (h grpcHandler) Create(ctx context.Context, req
*proto.CreateUpdateKRSRequest) (*proto.KRSResponse, error) {
	token := req.GetToken()
	idMahasiswa := int(req.GetIdMahasiswa())

	isAuth, err := VerifyToken(ctx, h.clientAuth, token)
	if err != nil {
		log.Println(err)
		return nil, err
	}
	if !isAuth {
		log.Println("token is not valid")
		return nil, errors.New("token is not valid")
	}

	isPay, err := VerifyPayment(ctx, h.clientPay, idMahasiswa)
	if err != nil {
```

```
		log.Println(err)
		return nil, err
	}
	if !isPay {
		log.Println("belum melakukan pembayaran")
		return nil, errors.New("belum melakukan pembayaran")
	}

	krs := &domain.Krs{
		IdMahasiswa: idMahasiswa,
		MataKuliahs: nil,
	}

	// loop proto.MataKuliah to convert to domain.MataKuliah untuk
ambil data yg dikirm
	for _, mataKuliah := range req.GetMataKuliahs() {
		krs.MataKuliahs = append(krs.MataKuliahs,
&domain.MataKuliah{
			Kode:     mataKuliah.Kode,
			Nama:     mataKuliah.Nama,
			Sks:      int(mataKuliah.Sks),
			Dosen:    mataKuliah.Dosen,
			Semester: mataKuliah.Semester,
		})
	}

	krs, err = h.service.Create(krs)
	if err != nil {
		log.Println(err)
		return nil, err
	}

	// loop krs.MataKuliah to convert to proto.MataKuliah
	var mataKuliahs []*proto.MataKuliah
	for _, mataKuliah := range krs.MataKuliahs {
		mataKuliahs = append(mataKuliahs, &proto.MataKuliah{
			Kode:     mataKuliah.Kode,
			Nama:     mataKuliah.Nama,
			Sks:      int32(mataKuliah.Sks),
			Dosen:    mataKuliah.Dosen,
			Semester: mataKuliah.Semester,
		})
	}

	res := &proto.KRSResponse{
		MataKuliahs: mataKuliahs,
```

```
	}

	return res, nil
}

// Update is a method that implements the Update method of the
KrsGrpcHandler interface
func (h grpcHandler) Update(ctx context.Context, req
*proto.CreateUpdateKRSRequest) (*proto.KRSResponse, error) {
	token := req.GetToken()
	idMahasiswa := int(req.GetIdMahasiswa())

	isAuth, err := VerifyToken(ctx, h.clientAuth, token)
	if err != nil {
		log.Println(err)
		return nil, err
	}
	if !isAuth {
		log.Println("token is not valid")
		return nil, errors.New("token is not valid")
	}

	isPay, err := VerifyPayment(ctx, h.clientPay, idMahasiswa)
	if err != nil {
		log.Println(err)
		return nil, err
	}
	if !isPay {
		log.Println("belum melakukan pembayaran")
		return nil, errors.New("belum melakukan pembayaran")
	}

	// parse proto.KRS to domain.Krs
	krs := &domain.Krs{
		IdMahasiswa: idMahasiswa,
		MataKuliahs: nil,
	}

	// loop proto.MataKuliah to convert to domain.MataKuliah
	for _, mataKuliah := range req.GetMataKuliahs() {
		krs.MataKuliahs = append(krs.MataKuliahs,
&domain.MataKuliah{
			Kode:     mataKuliah.Kode,
			Nama:     mataKuliah.Nama,
			Sks:      int(mataKuliah.Sks),
			Dosen:    mataKuliah.Dosen,
```

```
                        Semester: mataKuliah.Semester,
                })
        }

        // parse int32 to int64
        krs, err = h.service.Update(krs)
        if err != nil {
                log.Println(err)
                return nil, err
        }

        // loop krs.MataKuliah to convert to proto.MataKuliah
        var mataKuliahs []*proto.MataKuliah
        for _, mataKuliah := range krs.MataKuliahs {
                mataKuliahs = append(mataKuliahs, &proto.MataKuliah{
                        Kode:     mataKuliah.Kode,
                        Nama:     mataKuliah.Nama,
                        Sks:      int32(mataKuliah.Sks),
                        Dosen:    mataKuliah.Dosen,
                        Semester: mataKuliah.Semester,
                })
        }

        res := &proto.KRSResponse{
                MataKuliahs: mataKuliahs,
        }

        return res, nil
}

// Delete is a method that implements the Delete method of the
KrsGrpcHandler interface
func (h grpcHandler) Delete(ctx context.Context, req
*proto.DeleteKRSRequest) (*proto.DeleteKRSResponse, error) {
        token := req.GetToken()
        idMahasiswa := int(req.GetIdMahasiswa())

        isAuth, err := VerifyToken(ctx, h.clientAuth, token)
        if err != nil {
                log.Println(err)
                return nil, err
        }
        if !isAuth {
                log.Println("token is not valid")
                return nil, errors.New("token is not valid")
        }
```

```
	isPay, err := VerifyPayment(ctx, h.clientPay, idMahasiswa)
	if err != nil {
		log.Println(err)
		return nil, err
	}
	if !isPay {
		log.Println("belum melakukan pembayaran")
		return nil, errors.New("belum melakukan pembayaran")
	}

	err = h.service.Delete(token, idMahasiswa)
	if err != nil {
		log.Println(err)
		return nil, err
	}

	res := &proto.DeleteKRSResponse{
		Status: "success",
	}

	return res, nil
}
```

File krs.go

```
package domain

import (
	"context"
	"net/http"

	"github.com/dinel13/thesis-ac/krs/proto"
)

type MataKuliah struct {
	Kode     string `json:"kode"`
	Nama     string `json:"nama"`
	Sks      int    `json:"sks"`
	Dosen    string `json:"dosen"`
	Semester string `json:"semester"`
}

type Krs struct {
```

```
	IdMahasiswa int           `json:"id_mahasiswa"`
	MataKuliahs []*MataKuliah `json:"mata_kuliahs"`
}

type KrsRepository interface {
	Create(context.Context, *Krs) (*Krs, error)
	Read(context.Context, int) (*Krs, error)
	Update(context.Context, *Krs) (*Krs, error)
	Delete(context.Context, int) error
}

type KrsService interface {
	Read(string, int) (*Krs, error)
	Create(*Krs) (*Krs, error)
	Update(*Krs) (*Krs, error)
	Delete(string, int) error
}

type KrsRestHandlers interface {
	Read(http.ResponseWriter, *http.Request)
	Create(http.ResponseWriter, *http.Request)
	Update(http.ResponseWriter, *http.Request)
	Delete(http.ResponseWriter, *http.Request)
}

type KrsGrpcHandler interface {
	Read(context.Context, *proto.ReadKRSRequest)
(*proto.KRSResponse, error)
	Create(context.Context, *proto.CreateUpdateKRSRequest)
(*proto.KRSResponse, error)
	Update(context.Context, *proto.CreateUpdateKRSRequest)
(*proto.KRSResponse, error)
	Delete(context.Context, *proto.DeleteKRSRequest)
(*proto.DeleteKRSResponse, error)
}
```

File main.go

```
package main

import (
	"context"
	"fmt"
	"log"
```

```
	"net"
	"net/http"
	"os"
	"time"

	"github.com/dinel13/thesis-ac/krs/domain"
	"github.com/dinel13/thesis-ac/krs/proto"
	"github.com/dinel13/thesis-ac/krs/repository"
	"github.com/dinel13/thesis-ac/krs/rest"
	"github.com/dinel13/thesis-ac/krs/service"
	"github.com/go-redis/redis/v8"
	"google.golang.org/grpc"

	mygrpc "github.com/dinel13/thesis-ac/krs/grpc"
)

var urlAuth = os.Getenv("URL_AUTH")
var urlPay = os.Getenv("URL_PAYMENT")

// startRestServer starts the REST server
func startRestServer() {
	port := ":8080"
	fmt.Printf("Staring REST server on port %s\n", port)
	dbClient, krsRepo := startRepoRedis()

	defer dbClient.Close()

	// cs :=
rest.NewCoursRestHandlers(service.NewKrsService(krsRepo))
	ks := service.NewKrsService(krsRepo)
	cs := rest.NewCoursRestHandlers(ks)

	srv := &http.Server{
		Addr:    ":8080",
		Handler: rest.Routes(cs),
	}
	err := srv.ListenAndServe()
	if err != nil {
		log.Fatal(err)
	}
}

// startGRPCServer starts the gRPC server
func startGRPCServer() {
	port := ":9090"
	fmt.Printf("Staring gRPC server on port %s\n", port)
```

```
	dbClient, krsRepo := startRepoRedis()

	connAuth, err := grpc.Dial(fmt.Sprintf("%s:9091", urlAuth),
grpc.WithInsecure())
	if err != nil {
		log.Fatal(err)
	}

	connPayment, err := grpc.Dial(fmt.Sprintf("%s:9092", urlPay),
grpc.WithInsecure())
	if err != nil {
		log.Fatal(err)
	}

	defer func() {
		dbClient.Close()
		connAuth.Close()
		connPayment.Close()
	}()

	clientAuth := proto.NewAuthServiceClient(connAuth)
	clientPayment := proto.NewPaymentServiceClient(connPayment)

	ks := service.NewKrsService(krsRepo)

	cs := mygrpc.NewGrpcHandler(clientAuth, clientPayment, ks)

	// create gRPC server
	lis, err := net.Listen("tcp", port)
	if err != nil {
		log.Fatalf("failed to listen: %v", err)
	}

	s := grpc.NewServer()
	proto.RegisterKrsServiceServer(s, cs)

	if err := s.Serve(lis); err != nil {
		log.Fatalf("failed to serve: %v", err)
	}
}

func startRepoRedis() (*redis.Client, domain.KrsRepository) {
	dbClient := connectRedis()
	crDb := repository.NewKrsRepoRedisImpl(dbClient)
	return dbClient, crDb
}
```

```
func connectRedis() *redis.Client {
	log.Println("Connecting to redis...")
	client := redis.NewClient(&redis.Options{
		Addr: "localhost:6379",
	})
	_, err := client.Ping(context.Background()).Result()
	if err != nil {
		log.Fatal("Cannot connect to redis! Dying...")
	}
	log.Println("Connected to redis!")
	return client
}

func main() {
	go startRestServer()
	go startGRPCServer()
	time.Sleep(113880 * time.Hour)
}
```

Lampiran 9 *Source code service* AUTH menggunakan Node.js

File app.js untuk REST

```
const express = require("express");
const { login, verify, signup } = require("./controller");

const  startRestServer = () => {
  const app = express();

  app.use(express.json()); // for parsing application/json

  app.post("/signup", signup);
  app.post("/login", login);
  app.post("/verify", verify);
  app.listen(8081, () => console.log("REST server running at port
8081"));
}

module.exports = startRestServer;
```

File app.js untuk gRPC

```
const grpc = require("@grpc/grpc-js");
const PROTO_PATH = __dirname + "/../proto/auth.proto";
const protoLoader = require("@grpc/proto-loader");
const { Login, Verify } = require("./controller");

const packageDefinition = protoLoader.loadSync(PROTO_PATH, {
  keepCase: true,
  longs: String,
  enums: String,
  defaults: true,
  oneofs: true,
});

const protoDescriptor = grpc.loadPackageDefinition(packageDefinition);
// The protoDescriptor object has the full package hierarchy
const authservice = protoDescriptor.proto;

const server = new grpc.Server();

server.addService(authservice.AuthService.service, {
  Login: Login,
  Verify: Verify,
});

const startGrpcServer = () => {
  server.bindAsync(
    "0.0.0.0:9091",
    grpc.ServerCredentials.createInsecure(),
    () => {
      server.start();
      console.log("GRPC server running at port 9091");
    }
  );
};

module.exports = startGrpcServer;
```

File redis.js

```
const Redis = require("ioredis");
const redis = new Redis();

module.exports = redis;
```

File handler.js untuk REST

```
const jwt = require("jsonwebtoken");
const redis = require("../database/redis");

const signup = async (req, res) => {
  try {
    const { username, password } = req.body;
    const token = jwt.sign({ username }, "secretKey@123", { expiresIn:
"1d" });
    await redis.set(
      username,
      JSON.stringify({ username, password }),
      "EX",
      60 * 60 * 24
    );
    res.status(201).json({ token });
  } catch (error) {
    console.log(error);
    res.status(500).json({
      message: error,
    });
  }
};

const login = async (req, res) => {
    try {
      const { username, password } = req.body;
      let user = await redis.get(username);
      if (!user) {
        return res.status(400).json({
          message: "Username tidak ditemukan",
        });
      }
      user = JSON.parse(user);
      if (password !== user.password) {
        return res.status(400).json({
          message: "Password salah",
        });
      }
      token = jwt.sign(
        {
          username,
        },
        "secretKey@123",
        { expiresIn: "1d" }
```

```
      );
      res.status(200).json({
        token,
      });
    } catch (error) {
      console.log(error);
      res.status(500).json({
        message: error,
      });
    }
};

const verify = async (req, res) => {
  try {
    const token = req.headers.authorization.split(" ")[1];
    if (!token) {
      return res.status(401).json({
        isAuth: false,
      });
    }

    const decoded = jwt.verify(token, "secretKey@123");
    if (!decoded.username) {
      return res.status(401).json({
        isAuth: false,
      });
    }
    res.status(200).json({
      isAuth: true,
    });
  } catch (err) {
    return res.status(500).json({
      isAuth: false,
    });
  }
};

module.exports = {
  signup,
  login,
  verify,
};
```

File handler.js untuk gRPC

```
const jwt = require("jsonwebtoken");
const redis = require("../database/redis");

async function Login(_, callback) {
  let token;
  try {
    let userExits = await redis.get(_.request.username);
    if (!userExits) {
      throw new Error("User not exits");
    }
    userExits = JSON.parse(userExits);

    if (userExits.password != _.request.password) {
      throw new Error("Password not match");
    }
    token = jwt.sign(
      {
        username: userExits.username,
      },
      "secretKey@123",
      { expiresIn: "1d" }
    );
  } catch (error) {
    console.log(error);
    return callback(null, { token: "" });
  }
  return callback(null, { token });
}

async function Verify(_, callback) {
  let respon;
  try {
    const user = await jwt.verify(_.request.token, "secretKey@123");
    if (user) {
      respon = {
        is_auth: true,
      };
    } else {
      respon = {
        is_auth: false,
      };
    }
  } catch (error) {
    respon = {
      is_auth: false,
    };
  }
  return callback(null, respon);
}

module.exports = {
  Login,
```

```
  Verify,
};
```

File index.js

```
const startGrpc  = require("./grpc/app");
const startRest = require("./rest/app");

startGrpc();
startRest();
```

File package.json

```
{
  "name": "auth-service",
  "version": "1.0.0",
  "description": "auth service for my acedemic thesis",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "github.com/dinel13/thesis-ac"
  },
  "author": "salahuddin",
  "license": "ISC",
  "dependencies": {
    "@grpc/grpc-js": "^1.4.5",
    "@grpc/proto-loader": "^0.6.7",
    "express": "^4.17.2",
    "google-protobuf": "^3.19.0",
    "ioredis": "^4.28.2",
    "jsonwebtoken": "^8.5.1"
  }
}
```

Lampiran 10 *Source code service* AUTH menggunakan Go

File route.go

```
package rest

import (
	"net/http"
	"github.com/dinel13/thesis-ac/auth/domain"
	"github.com/julienschmidt/httprouter"
)

func Routes(ah domain.AuthRestHandlers) http.Handler {
	r := httprouter.New()
	r.HandlerFunc(http.MethodPost, "/verify", ah.Verify)
	r.HandlerFunc(http.MethodPost, "/login", ah.Login)
	r.HandlerFunc(http.MethodPost, "/signup", ah.Signup)
	return r
}
```

File redis.go

```
package repository

import (
	"context"
	"encoding/json"
	"errors"

	"github.com/dinel13/thesis-ac/auth/domain"
	"github.com/go-redis/redis/v8"
)

func NewAuthRepoRedisImpl(dbClient *redis.Client) domain.AuthRepository
{
	return authRepositoryImpl{Rds: dbClient}
}

type authRepositoryImpl struct {
	Rds *redis.Client
}

func (m authRepositoryImpl) Login(ctx context.Context, auth
*domain.LoginSignupRequest) error {
	val, err := m.Rds.Get(ctx, auth.Username).Bytes()
	if err == redis.Nil {
```

```
		return errors.New("user not found")
	} else if err != nil {
		return err
	} else {
		user := domain.LoginSignupRequest{}
		err = json.Unmarshal(val, &user)
		if err != nil {
			return err
		}
		if user.Password != auth.Password {
			return errors.New("wrong password")
		}
		return nil
	}
}

// Signup adds a new auth
func (m authRepositoryImpl) Signup(ctx context.Context, auth
*domain.LoginSignupRequest) error {
	// set user to redis
	val, err := json.Marshal(auth)
	if err != nil {
		return err
	}
	err = m.Rds.Set(ctx, auth.Username, val, 0).Err()
	if err != nil {
		return err
	}

	return nil
}
```

File service.go

```
package service

import (
	"context"
	"time"

	"github.com/dinel13/thesis-ac/auth/domain"
	"github.com/dinel13/thesis-ac/auth/helper"
)
```

```
func NewAuthService(repo domain.AuthRepository) domain.AuthService {
	return authService{repo}
}

type authService struct {
	repo domain.AuthRepository
}

// GetAuth returns a auth by id
func (p authService) Verify(tokenString string) error {
	err := helper.ParseToken(tokenString, "secretKey@123")
	if err != nil {
		return err
	}
	return nil
}

//	AddAuth adds a new auth
func (p authService) Login(auth *domain.LoginSignupRequest)
(*domain.LoginSignupResponse, error) {
	ctx, cancel := context.WithTimeout(context.Background(),
3*time.Second)
	defer cancel()

	err := p.repo.Login(ctx, auth)
	if err != nil {
		return nil, err
	}

	// create token
	token, err := helper.CreateToken(auth.Username, "secretKey@123")
	if err != nil {
		return nil, err
	}

	res := &domain.LoginSignupResponse{
		Token: token,
	}

	return res, nil
}

func (p authService) Signup(auth *domain.LoginSignupRequest)
(*domain.LoginSignupResponse, error) {
	ctx, cancel := context.WithTimeout(context.Background(),
3*time.Second)
```

```
	defer cancel()

	err := p.repo.Signup(ctx, auth)
	if err != nil {
		return nil, err
	}

	// create token
	token, err := helper.CreateToken(auth.Username, "secretKey@123")
	if err != nil {
		return nil, err
	}

	res := &domain.LoginSignupResponse{
		Token: token,
	}

	return res, nil
}
```

File handler.go untuk REST

```
package rest

import (
	"errors"
	"net/http"
	"strings"

	"github.com/dinel13/thesis-ac/auth/domain"
)

func NewAuthRestHandlers(s domain.AuthService) domain.AuthRestHandlers
{
	return authHandlers{s}
}

type authHandlers struct {
	service domain.AuthService
}

// GetAuth is handler for GET /auth/{id}
func (h authHandlers) Verify(w http.ResponseWriter, r *http.Request) {
	authorizationHeader := r.Header.Get("Authorization")
```

```
	if !strings.Contains(authorizationHeader, "Bearer") {
		WriteJsonError(w, errors.New("invalid token"),
http.StatusBadRequest)
		return
	}

	tokenString := strings.Replace(authorizationHeader, "Bearer ",
"", -1)
	err := h.service.Verify(tokenString)
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}

	WriteJson(w, http.StatusOK, true, "isAuth")
}

// Create is handler for POST /auth to create COurse
func (h authHandlers) Login(w http.ResponseWriter, r *http.Request) {
	// get the auth from request body
	auth := &domain.LoginSignupRequest{}
	err := ReadJson(r, auth)
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}

	// create the auth
	c, err := h.service.Login(auth)
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}

	// write the auth to response
	WriteJson(w, http.StatusCreated, c.Token, "token")
}

func (h authHandlers) Signup(w http.ResponseWriter, r *http.Request) {
	// get the auth from request body
	auth := &domain.LoginSignupRequest{}
	err := ReadJson(r, auth)
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}
```

```
	// create the auth
	c, err := h.service.Signup(auth)
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}

	// write the auth to response
	WriteJson(w, http.StatusCreated, c.Token, "token")
}
```

File handler.go untuk gRPC

```
package grpc

import (
	"context"
	"log"

	"github.com/dinel13/thesis-ac/auth/domain"
	"github.com/dinel13/thesis-ac/auth/proto"
)

func NewGrpcHandler(s domain.AuthService) domain.AuthGrpcHandler {
	return grpcHandler{s}
}

type grpcHandler struct {
	service domain.AuthService
}

// read is a method that implements the Read method of the
AuthGrpcHandler interface
func (h grpcHandler) Verify(ctx context.Context, req
*proto.VerifyRequest) (*proto.VerifyResponse, error) {
	token := req.GetToken()

	// parse int32 to int64
	err := h.service.Verify(token)
	if err != nil {
		log.Println(err)
		return nil, err
	}
```

```
	res := &proto.VerifyResponse{
		IsAuth: true,
	}

	return res, nil
}

// Create is a method that implements the Create method of the
AuthGrpcHandler interface
func (h grpcHandler) Login(ctx context.Context, req
*proto.LoginRequest) (*proto.LoginResponse, error) {

	username := req.GetUsername()
	password := req.GetPassword()

	auth := &domain.LoginSignupRequest{
		Username: username,
		Password: password,
	}

	authRespon, err := h.service.Login(auth)
	if err != nil {
		log.Println(err)
		return nil, err
	}

	res := &proto.LoginResponse{
		Token: authRespon.Token,
	}

	return res, nil
}
```

File token.go

```
package helper

import (
	"errors"
	"time"

	"github.com/golang-jwt/jwt"
)
```

```
type MyClaims struct {
	jwt.StandardClaims
	Username string `json:"username"`
}

func CreateToken(username string, secretKey string) (string, error) {
	claims := MyClaims{
		StandardClaims: jwt.StandardClaims{
			Issuer:    "auth",
			ExpiresAt: time.Now().Add(time.Duration(1000) *
time.Hour).Unix(),
		},
		Username: username,
	}

	token := jwt.NewWithClaims(
		jwt.SigningMethodHS256,
		claims,
	)

	signedToken, err := token.SignedString([]byte(secretKey))
	if err != nil {
		return "", err
	}
	return signedToken, nil
}

func parseTokenJwt(tokenString string, secretKey string) (*jwt.Token,
error) {
	token, err := jwt.Parse(tokenString, func(token *jwt.Token)
(interface{}, error) {
		if method, ok := token.Method.(*jwt.SigningMethodHMAC);
!ok {
			return nil, errors.New("signing method invalid")
		} else if method != jwt.SigningMethodHS256 {
			return nil, errors.New("signing method invalid")
		}

		return []byte(secretKey), nil
	})

	return token, err
}

func ParseToken(tokenString string, secretKey string) error {
```

```
	token, err := parseTokenJwt(tokenString, secretKey)

	if err != nil {
		return err
	}

	claims, ok := token.Claims.(jwt.MapClaims)
	if !ok || !token.Valid {
		return errors.New("token invalid")
	}

	// look the containts of claims
	expires_at := int(claims["exp"].(float64))

	// convert expires_at to time.Time
	expires_at_time := time.Unix(int64(expires_at), 0)

	// cek if token expired
	if time.Now().Unix() > expires_at_time.Unix() {
		return errors.New("token expired")
	}

	return nil
}
```

File auth.go

```
package domain

import (
	"context"
	"net/http"

	"github.com/dinel13/thesis-ac/auth/proto"
)

type LoginSignupRequest struct {
	Username string `json:"username"`
	Password string `json:"password"`
}

type LoginSignupResponse struct {
	Token string `json:"token"`
}
```

```
type VerifyResponse struct {
	IsAuth bool `json:"isAuth"`
}

type AuthRepository interface {
	Login(context.Context, *LoginSignupRequest) error
	Signup(context.Context, *LoginSignupRequest) error
}

type AuthService interface {
	Signup(*LoginSignupRequest) (*LoginSignupResponse, error)
	Login(*LoginSignupRequest) (*LoginSignupResponse, error)
	Verify(string) error
}

type AuthRestHandlers interface {
	Login(http.ResponseWriter, *http.Request)
	Verify(http.ResponseWriter, *http.Request)
	Signup(http.ResponseWriter, *http.Request)
}

type AuthGrpcHandler interface {
	Login(context.Context, *proto.LoginRequest)
(*proto.LoginResponse, error)
	Verify(context.Context, *proto.VerifyRequest)
(*proto.VerifyResponse, error)
}
```

File main.go

```
package main

import (
	"context"
	"fmt"
	"log"
	"net"
	"net/http"
	"time"

	"google.golang.org/grpc"

	"github.com/dinel13/thesis-ac/auth/domain"
```

```
	"github.com/dinel13/thesis-ac/auth/repository"
	"github.com/go-redis/redis/v8"

	mygrpc "github.com/dinel13/thesis-ac/auth/grpc"

	"github.com/dinel13/thesis-ac/auth/proto"
	"github.com/dinel13/thesis-ac/auth/rest"
	"github.com/dinel13/thesis-ac/auth/service"
)

// StartRestServer starts the REST server
func StartRestServer() {
	port := ":8081"
	fmt.Printf("Staring REST server on port %s\n", port)

	dbClient, authRepo := startRepoRedis()
	defer dbClient.Close()

	cs := rest.NewAuthRestHandlers(service.NewAuthService(authRepo))

	srv := &http.Server{
		Addr:    port,
		Handler: rest.Routes(cs),
	}
	err := srv.ListenAndServe()
	if err != nil {
		log.Fatal(err)
	}
}

// StartGRPCServer starts the gRPC server
func StartGRPCServer() {
	port := ":9091"
	fmt.Printf("Staring gRPC server on port %s\n", port)

	dbClient, authRepo := startRepoRedis()
	defer dbClient.Close()
	cs := mygrpc.NewGrpcHandler(service.NewAuthService(authRepo))

	// create gRPC server
	lis, err := net.Listen("tcp", port)
	if err != nil {
		log.Fatalf("failed to listen: %v", err)
	}

	s := grpc.NewServer()
```

```
	proto.RegisterAuthServiceServer(s, cs)

	if err := s.Serve(lis); err != nil {
		log.Fatalf("failed to serve: %v", err)
	}
}

func startRepoRedis() (*redis.Client, domain.AuthRepository) {
	dbClient := connectRedis()
	crDb := repository.NewAuthRepoRedisImpl(dbClient)
	return dbClient, crDb
}

func connectRedis() *redis.Client {
	// connect to redis
	log.Println("Connecting to redis...")
	client := redis.NewClient(&redis.Options{
		Addr: "localhost:6379",
	})
	_, err := client.Ping(context.Background()).Result()
	if err != nil {
		log.Fatal("Cannot connect to redis! Dying...")
	}
	log.Println("Connected to redis!")
	return client
}

func main() {
	go StartRestServer()
	go StartGRPCServer()
	time.Sleep(113880 * time.Hour)
}
```

Lampiran 11. *Source code service* Payment

File route.go

```
package rest

import (
	"net/http"

	"github.com/dinel13/thesis-ac/payment/domain"
	"github.com/julienschmidt/httprouter"
```

```
)

func Routes(ph domain.PaymentRestHandlers) http.Handler {
	r := httprouter.New()

	r.HandlerFunc(http.MethodGet, "/verify/:id", ph.Verify)
	r.HandlerFunc(http.MethodPost, "/pay", ph.Create)

	return r
}
```

File redis.go

```
package repository

import (
	"context"
	"encoding/json"
	"strconv"
	"time"

	"github.com/dinel13/thesis-ac/payment/domain"
	"github.com/go-redis/redis/v8"
)

func NewPaymentRepoRedisImpl(dbClient *redis.Client)
domain.PaymentRepository {
	return paymentRepositoryImpl{Rds: dbClient}
}

type paymentRepositoryImpl struct {
	Rds *redis.Client
}

//	CreatePayment creates a new payment
func (m paymentRepositoryImpl) Create(ctx context.Context, payment
*domain.PaymentRequest) (*domain.PaymentResponse, error) {
	paymentJson, err := json.Marshal(payment)
	if err != nil {
		return nil, err
	}

	// "pay" + payment.ID so that not same in krs service
	err = m.Rds.Set(ctx, "pay"+strconv.Itoa(payment.IdMahasiswa),
```

```
paymentJson, 24*time.Hour).Err()
	if err != nil {
		return nil, err
	}
	res := &domain.PaymentResponse{IsPay: true}
	return res, nil
}

// GetPayment returns a payment by id
func (m paymentRepositoryImpl) Verify(ctx context.Context, id int)
(*domain.PaymentResponse, error) {
	err := m.Rds.Get(ctx, "pay"+strconv.Itoa(id)).Err()
	if err != nil {
		if err.Error() == "redis: nil" {
			return &domain.PaymentResponse{
				IsPay: false,
			}, nil
		}
		return nil, err
	}
	payment := domain.PaymentResponse{
		IsPay: true,
	}
	if err != nil {
		return nil, err
	}
	return &payment, nil
}
```

File service.go

```
package service

import (
	"context"
	"time"

	"github.com/dinel13/thesis-ac/payment/domain"
)

func NewPaymentService(repo domain.PaymentRepository)
domain.PaymentService {
	return paymentService{repo}
}
```

```
type paymentService struct {
	repo domain.PaymentRepository
}

// GetPayment returns a payment by id
func (p paymentService) Verify(id int) (*domain.PaymentResponse, error)
{
	ctx, cancel := context.WithTimeout(context.Background(),
3*time.Second)
	defer cancel()

	c, err := p.repo.Verify(ctx, id)
	if err != nil {
		return nil, err
	}
	return c, nil
}

//	AddPayment adds a new payment
func (p paymentService) Create(payment *domain.PaymentRequest)
(*domain.PaymentResponse, error) {
	ctx, cancel := context.WithTimeout(context.Background(),
3*time.Second)
	defer cancel()

	c, err := p.repo.Create(ctx, payment)
	if err != nil {
		return nil, err
	}
	return c, nil
}
```

File handler.go untuk REST

```
package rest

import (
	"net/http"
	"strconv"

	"github.com/dinel13/thesis-ac/payment/domain"
	"github.com/julienschmidt/httprouter"
)
```

```
func NewPaymentRestHandlers(s domain.PaymentService)
domain.PaymentRestHandlers {
	return paymentHandlers{s}
}

type paymentHandlers struct {
	service domain.PaymentService
}

// GetPayment is handler for GET /payment/{id}
func (h paymentHandlers) Verify(w http.ResponseWriter, r *http.Request)
{
	// get the payment id from request param
	params := httprouter.ParamsFromContext(r.Context())
	id, err := strconv.Atoi(params.ByName("id"))
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
	}

	// get the payment from service
	payment, err := h.service.Verify(id)
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}

	// write the payment to response
	WriteJson(w, http.StatusOK, payment, "payment")
}

// Create is handler for POST /payment to create COurse
func (h paymentHandlers) Create(w http.ResponseWriter, r *http.Request)
{
	// get the payment from request body
	payment := &domain.PaymentRequest{}
	err := ReadJson(r, payment)
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
		return
	}

	// create the payment
	c, err := h.service.Create(payment)
	if err != nil {
		WriteJsonError(w, err, http.StatusInternalServerError)
```

```
		return
	}

	// write the payment to response
	WriteJson(w, http.StatusCreated, c, "payment")
}
```

File handler.go untuk gRPC

```
package grpc

import (
	"context"
	"log"

	"github.com/dinel13/thesis-ac/payment/domain"
	"github.com/dinel13/thesis-ac/payment/proto"
)

func NewGrpcHandler(s domain.PaymentService) domain.PaymentGrpcHandler
{
	return grpcHandler{s}
}

type grpcHandler struct {
	service domain.PaymentService
}

// read is a method that implements the Read method of the
PaymentGrpcHandler interface
func (h grpcHandler) Verify(ctx context.Context, req
*proto.VerifyPaymentRequest) (*proto.PaymentResponse, error) {
	id_mahasiswa := req.GetIdMahasiswa()
	idMahasiswa := int(id_mahasiswa)

	// parse int32 to int64
	payment, err := h.service.Verify(idMahasiswa)
	if err != nil {
		log.Println(err)
		return nil, err
	}

	res := &proto.PaymentResponse{
		IsPay: payment.IsPay,
```

```
	}

	return res, nil
}

// Create is a method that implements the Create method of the
PaymentGrpcHandler interface
func (h grpcHandler) Create(ctx context.Context, req
*proto.CreatePaymentRequest) (*proto.PaymentResponse, error) {

	IdMahasiswa := int(req.GetIdMahasiswa())
	Jumlah := float64(req.GetJumlah())
	Metode := req.GetMetode()

	payment := &domain.PaymentRequest{
		IdMahasiswa: IdMahasiswa,
		Jumlah:      Jumlah,
		Metode:      Metode,
	}

	paymentRespon, err := h.service.Create(payment)
	if err != nil {
		log.Println(err)
		return nil, err
	}

	res := &proto.PaymentResponse{
		IsPay: paymentRespon.IsPay,
	}

	return res, nil
}
```

File krs.go

```
package domain

import (
	"context"
	"net/http"
	"github.com/dinel13/thesis-ac/payment/proto"
)

type PaymentRequest struct {
```

```
	IdMahasiswa int     `json:"id_mahasiswa"`
	Jumlah      float64 `json:"jumlah"`
	Metode      string  `json:"metode"`
}
type VerifyRequest struct {
	IdMahasiswa int `json:"id_mahasiswa"`
}
type PaymentResponse struct {
	IsPay bool `json:"isPay"`
}

type PaymentRepository interface {
	Create(context.Context, *PaymentRequest) (*PaymentResponse,
error)
	Verify(context.Context, int) (*PaymentResponse, error)
}

type PaymentService interface {
	Create(*PaymentRequest) (*PaymentResponse, error)
	Verify(int) (*PaymentResponse, error)
}

type PaymentRestHandlers interface {
	Create(http.ResponseWriter, *http.Request)
	Verify(http.ResponseWriter, *http.Request)
}

type PaymentGrpcHandler interface {
	Create(context.Context, *proto.CreatePaymentRequest)
(*proto.PaymentResponse, error)
	Verify(context.Context, *proto.VerifyPaymentRequest)
(*proto.PaymentResponse, error)
}
```

File main.go

```
package main

import (
	"context"
	"fmt"
	"log"
	"net"
	"net/http"
```

```
	"time"

	"google.golang.org/grpc"

	"github.com/dinel13/thesis-ac/payment/domain"
	"github.com/dinel13/thesis-ac/payment/repository"
	"github.com/go-redis/redis/v8"

	mygrpc "github.com/dinel13/thesis-ac/payment/grpc"

	"github.com/dinel13/thesis-ac/payment/proto"
	"github.com/dinel13/thesis-ac/payment/rest"
	"github.com/dinel13/thesis-ac/payment/service"
)

// StartRestServer starts the REST server
func StartRestServer() {
	port := ":8082"
	fmt.Printf("Staring REST server on port %s\n", port)

	dbClient, paymentRepo := startRepoRedis()
	defer dbClient.Close()

	cs :=
rest.NewPaymentRestHandlers(service.NewPaymentService(paymentRepo))

	srv := &http.Server{
		Addr:    port,
		Handler: rest.Routes(cs),
	}
	err := srv.ListenAndServe()
	if err != nil {
		log.Fatal(err)
	}
}

// StartGRPCServer starts the gRPC server
func StartGRPCServer() {
	port := ":9092"
	fmt.Printf("Staring gRPC server on port %s\n", port)

	dbClient, paymentRepo := startRepoRedis()
	defer dbClient.Close()
	cs :=
mygrpc.NewGrpcHandler(service.NewPaymentService(paymentRepo))
```

```
	// create gRPC server
	lis, err := net.Listen("tcp", port)
	if err != nil {
		log.Fatalf("failed to listen: %v", err)
	}

	s := grpc.NewServer()
	proto.RegisterPaymentServiceServer(s, cs)

	if err := s.Serve(lis); err != nil {
		log.Fatalf("failed to serve: %v", err)
	}
}

func startRepoRedis() (*redis.Client, domain.PaymentRepository) {
	dbClient := connectRedis()
	crDb := repository.NewPaymentRepoRedisImpl(dbClient)
	return dbClient, crDb
}

func connectRedis() *redis.Client {
	// connect to redis
	log.Println("Connecting to redis...")
	client := redis.NewClient(&redis.Options{
		Addr: "localhost:6379",
	})
	_, err := client.Ping(context.Background()).Result()
	if err != nil {
		log.Fatal("Cannot connect to redis! Dying...")
	}
	log.Println("Connected to redis!")
	return client
}

func main() {
	go StartRestServer()
	go StartGRPCServer()
	time.Sleep(113880 * time.Hour)
}
```

File json.go untuk service KRS, Auth dan Payment

```
package rest
```

```
import (
	"encoding/json"
	"fmt"
	"log"
	"net/http"
	"os"
	"runtime/debug"
)

func ReadJson(request *http.Request, result interface{}) error {
	decoder := json.NewDecoder(request.Body)
	err := decoder.Decode(result)
	if err != nil {
		return err
	}
	return nil
}

func WriteJson(w http.ResponseWriter, status int, data interface{},
wrap string) error {
	wraper := make(map[string]interface{})

	wraper[wrap] = data

	js, err := json.Marshal(wraper)
	if err != nil {
		return err
	}

	w.Header().Set("Content-Type", "application/json")
	w.WriteHeader(status)
	w.Write(js)

	return nil
}

func WriteJsonError(w http.ResponseWriter, err error, status ...int) {
	statusCode := http.StatusBadRequest
	if len(status) > 0 {
		statusCode = status[0]
	}

	logErorr(err)

	type jsonError struct {
		Message string `json:"message"`
```

```
	}

	theError := jsonError{
		Message: err.Error(),
	}

	WriteJson(w, statusCode, theError, "error")
}

var errorLog *log.Logger

func logErorr(err error) {
	errorLog = log.New(os.Stdin, "ERROR: ",
log.Ldate|log.Ltime|log.Lshortfile)
	trace := fmt.Sprintf("%s\n%s", err, debug.Stack())
	errorLog.Println(trace)
}
```

Lampiran 12**.** File jmx sebagai definisi pengujian

```
<?xml version="1.0" encoding="UTF-8"?>
<jmeterTestPlan version="1.2" properties="5.0" jmeter="5.4.1">
  <hashTree>
    <TestPlan guiclass="TestPlanGui" testclass="TestPlan"
testname="grpc-auth-login" enabled="true">
      <stringProp name="TestPlan.comments"></stringProp>
      <boolProp name="TestPlan.functional_mode">false</boolProp>
      <boolProp name="TestPlan.tearDown_on_shutdown">true</boolProp>
      <boolProp name="TestPlan.serialize_threadgroups">false</boolProp>
      <elementProp name="TestPlan.user_defined_variables"
elementType="Arguments" guiclass="ArgumentsPanel" testclass="Arguments"
testname="User Defined Variables" enabled="true">
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
      <stringProp name="TestPlan.user_define_classpath"></stringProp>
    </TestPlan>
    <hashTree>
      <ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup"
testname="user" enabled="true">
        <stringProp
name="ThreadGroup.on_sample_error">continue</stringProp>
        <elementProp name="ThreadGroup.main_controller"
elementType="LoopController" guiclass="LoopControlPanel"
testclass="LoopController" testname="Loop Controller" enabled="true">
```

```
          <boolProp
name="LoopController.continue_forever">false</boolProp>
          <stringProp name="LoopController.loops">1</stringProp>
        </elementProp>
        <stringProp name="ThreadGroup.num_threads">100</stringProp>
        <stringProp name="ThreadGroup.ramp_time">0</stringProp>
        <boolProp name="ThreadGroup.scheduler">false</boolProp>
        <stringProp name="ThreadGroup.duration"></stringProp>
        <stringProp name="ThreadGroup.delay"></stringProp>
        <boolProp
name="ThreadGroup.same_user_on_next_iteration">false</boolProp>
      </ThreadGroup>
      <hashTree>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui"
testclass="HTTPSamplerProxy" testname="create" enabled="true">
          <boolProp name="HTTPSampler.postBodyRaw">true</boolProp>
          <elementProp name="HTTPsampler.Arguments"
elementType="Arguments">
            <collectionProp name="Arguments.arguments">
              <elementProp name="" elementType="HTTPArgument">
                <boolProp
name="HTTPArgument.always_encode">false</boolProp>
                <stringProp name="Argument.value">{&#xd;
   &quot;username&quot;: &quot;salahuddin&quot;,&#xd;
   &quot;password&quot;: &quot;Password123&quot;&#xd;
}</stringProp>
                <stringProp name="Argument.metadata">=</stringProp>
              </elementProp>
            </collectionProp>
          </elementProp>
          <stringProp name="HTTPSampler.domain">127.0.0.1</stringProp>
          <stringProp name="HTTPSampler.port">8085</stringProp>
          <stringProp name="HTTPSampler.protocol"></stringProp>
          <stringProp name="HTTPSampler.contentEncoding"></stringProp>
          <stringProp name="HTTPSampler.path">/grpc/login</stringProp>
          <stringProp name="HTTPSampler.method">POST</stringProp>
          <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
          <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
          <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
          <boolProp
name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
          <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
          <stringProp name="HTTPSampler.connect_timeout"></stringProp>
          <stringProp name="HTTPSampler.response_timeout"></stringProp>
        </HTTPSamplerProxy>
        <hashTree>
```

```
          <ResultCollector guiclass="ViewResultsFullVisualizer"
testclass="ResultCollector" testname="View Results Tree"
enabled="true">
            <boolProp
name="ResultCollector.error_logging">false</boolProp>
            <objProp>
              <name>saveConfig</name>
              <value class="SampleSaveConfiguration">
                <time>true</time>
                <latency>true</latency>
                <timestamp>true</timestamp>
                <success>true</success>
                <label>true</label>
                <code>true</code>
                <message>true</message>
                <threadName>true</threadName>
                <dataType>true</dataType>
                <encoding>false</encoding>
                <assertions>true</assertions>
                <subresults>true</subresults>
                <responseData>false</responseData>
                <samplerData>false</samplerData>
                <xml>false</xml>
                <fieldNames>true</fieldNames>
                <responseHeaders>false</responseHeaders>
                <requestHeaders>false</requestHeaders>
                <responseDataOnError>false</responseDataOnError>

<saveAssertionResultsFailureMessage>true</saveAssertionResultsFailureMe
ssage>
                <assertionsResultsToSave>0</assertionsResultsToSave>
                <bytes>true</bytes>
                <sentBytes>true</sentBytes>
                <url>true</url>
                <threadCounts>true</threadCounts>
                <idleTime>true</idleTime>
                <connectTime>true</connectTime>
              </value>
            </objProp>
            <stringProp name="filename"></stringProp>
          </ResultCollector>
          <hashTree/>
          <ResultCollector guiclass="SummaryReport"
testclass="ResultCollector" testname="Summary Report" enabled="true">
            <boolProp
name="ResultCollector.error_logging">false</boolProp>
```

```
            <objProp>
              <name>saveConfig</name>
              <value class="SampleSaveConfiguration">
                <time>true</time>
                <latency>true</latency>
                <timestamp>true</timestamp>
                <success>true</success>
                <label>true</label>
                <code>true</code>
                <message>true</message>
                <threadName>true</threadName>
                <dataType>true</dataType>
                <encoding>false</encoding>
                <assertions>true</assertions>
                <subresults>true</subresults>
                <responseData>false</responseData>
                <samplerData>false</samplerData>
                <xml>false</xml>
                <fieldNames>true</fieldNames>
                <responseHeaders>false</responseHeaders>
                <requestHeaders>false</requestHeaders>
                <responseDataOnError>false</responseDataOnError>

<saveAssertionResultsFailureMessage>true</saveAssertionResultsFailureMe
ssage>
                <assertionsResultsToSave>0</assertionsResultsToSave>
                <bytes>true</bytes>
                <sentBytes>true</sentBytes>
                <url>true</url>
                <threadCounts>true</threadCounts>
                <idleTime>true</idleTime>
                <connectTime>true</connectTime>
              </value>
            </objProp>
            <stringProp name="filename"></stringProp>
          </ResultCollector>
          <hashTree/>
        </hashTree>
      </hashTree>
    </hashTree>
  </hashTree>
</jmeterTestPlan>
```

Lampiran 13. Program pengukur CPU Utilization

```
package main

import (
	"bytes"
	"flag"
	"fmt"
	"log"
	"os"
	"os/exec"
	"strconv"
	"strings"
	"time"
)

type Process struct {
	pid     string
	cpu     float64
	mem     float64
	command string
}

func main() {
	output := flag.String("o", "", "output file")
	pid := flag.String("p", "", "pid")
	flag.Parse()
	if *pid == "" {
		log.Fatal("pid is empty")
	}
	if *output == "" {
		log.Fatal("output file is empty")
	}
	f, err := os.Create(*output)
	if err != nil {
		log.Fatal(err)
	}
	defer f.Close()

	f.WriteString("CPU\t\tMEM\t\tTIME\t\t\t\tPID\t\tCOMMAND\n")

	for {
		process := GetProcessInfoUseTop(*pid)
		if process == nil || process.cpu <= 0.3 {
			continue
		}
```

```
			go printProcessInfo(process)

			f.WriteString(strconv.FormatFloat(process.cpu, 'f', 2, 64)
+ "\t\t")
			f.WriteString(strconv.FormatFloat(process.mem, 'f', 2, 64)
+ "\t\t")
			f.WriteString(time.Now().Format("2006-01-02 15:04:05") +
"\t\t")
			f.WriteString(process.pid + "\t\t")
			f.WriteString(process.command + "\n")

			// time.Sleep(500 * time.Millisecond)
		}

}

func printProcessInfo(process *Process) {
		log.Printf("pid: %s, cpu: %f, mem: %f, command: %s\n",
process.pid, process.cpu, process.mem, process.command)
}

func GetProcessInfoUseTop(pid string) *Process {
		// top -b -n 2 -d 0.2 -p 18368 | tail -1 | awk '{print
$9,$10,$12}'
		cmd := exec.Command("top", "-b", "-n", "2", "-d", "0.1", "-p",
pid)
		var outb, errb bytes.Buffer
		cmd.Stdout = &outb
		cmd.Stderr = &errb
		err := cmd.Run()
		if err != nil {
			log.Fatal(fmt.Sprint(err) + ": " + errb.String())
		}
		for {
			line, err := outb.ReadString('\n')
			if err != nil {
				break
			}
			tokens := strings.Split(line, " ")
			ft := make([]string, 0)
			for _, t := range tokens {
				if t != "" && t != "\t" {
					ft = append(ft, t)
				}
			}
```

```
            if pid != ft[0] {
                  continue
            }
            cpu, err := strconv.ParseFloat(ft[8], 64)
            if err != nil {
                  log.Fatal(err)
            }
            if cpu == 0 {
                  continue
            }
            mem, err := strconv.ParseFloat(ft[9], 64)
            if err != nil {
                  log.Fatal(err)
            }
            return &Process{pid, cpu, mem, ft[11]}
      }
      return nil
}
```