Scalable Project

Piazza

Media Engineering and Technology Faculty
German University in Cairo

Team Name: Brute Force

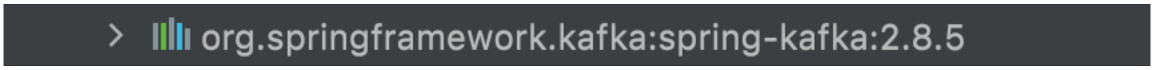
Submission Date: 3 June 2022

Table of Contents

Message Queues: Apache Kafka

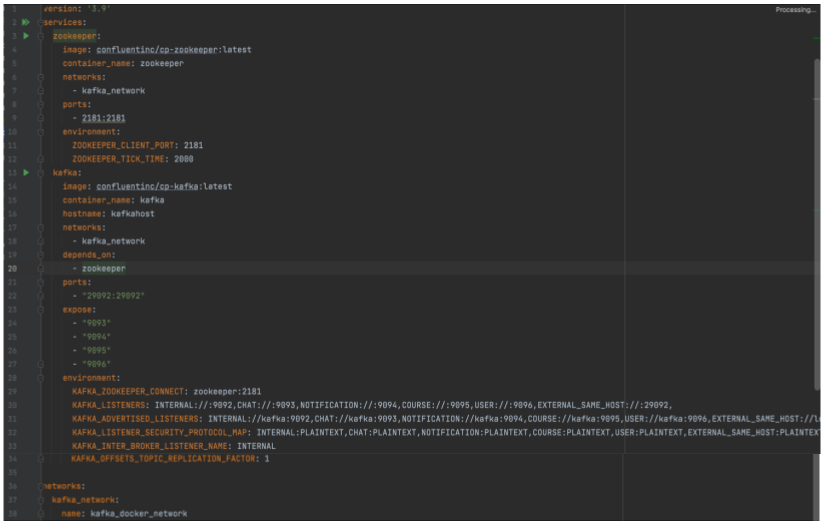
1489

Maven Dependencies:



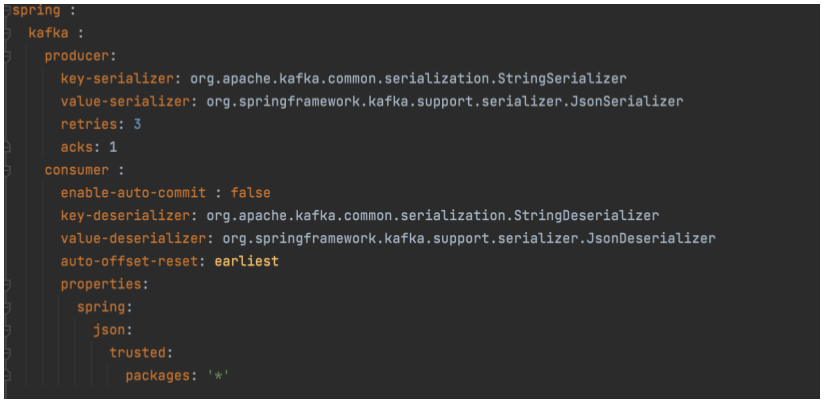
```
org.springframework.kafka:spring-kafka:2.8.5
```

Docker Image:



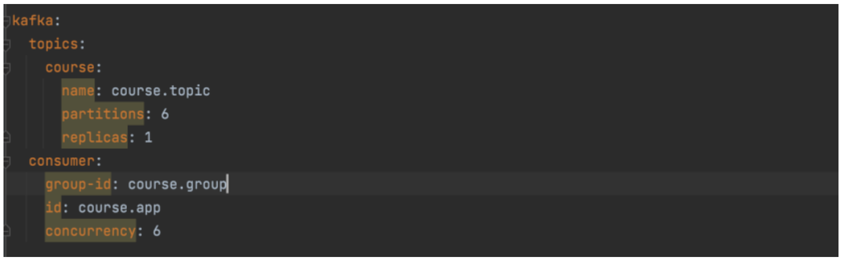
```yaml
version: '3.9'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    container_name: zookeeper
    networks:
      - kafka_network
    ports:
      - 2181:2181
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
  kafka:
    image: confluentinc/cp-kafka:latest
    container_name: kafka
    hostname: kafkahost
    networks:
      - kafka_network
    depends_on:
      - zookeeper
    ports:
      - "29092:29092"
    expose:
      - "9093"
      - "9094"
      - "9095"
      - "9096"
    environment:
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENERS: INTERNAL://:9092,CHAT://:9093,NOTIFICATION://:9094,COURSE://:9095,USER://:9096,EXTERNAL_SAME_HOST://:29092,
      KAFKA_ADVERTISED_LISTENERS: INTERNAL://kafka:9092,CHAT://kafka:9093,NOTIFICATION://kafka:9094,COURSE://kafka:9095,USER://kafka:9096,EXTERNAL_SAME_HOST://l
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INTERNAL:PLAINTEXT,CHAT:PLAINTEXT,NOTIFICATION:PLAINTEXT,COURSE:PLAINTEXT,USER:PLAINTEXT,EXTERNAL_SAME_HOST:PLAINTEX
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1

networks:
  kafka_network:
    name: kafka_docker_network
```

Kafka base configurations:



```yaml
spring :
  kafka :
    producer:
      key-serializer: org.apache.kafka.common.serialization.StringSerializer
      value-serializer: org.springframework.kafka.support.serializer.JsonSerializer
      retries: 3
      acks: 1
    consumer :
      enable-auto-commit : false
      key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
      value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer
      auto-offset-reset: earliest
      properties:
        spring:
          json:
            trusted:
              packages: '*'
```

Kafka configuration instance:



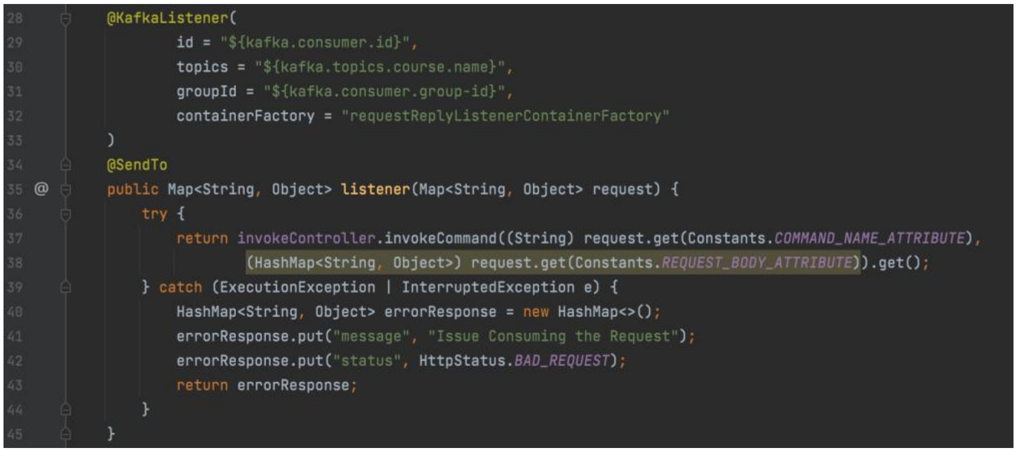
```
kafka:
  topics:
    course:
      name: course.topic
      partitions: 6
      replicas: 1
  consumer:
    group-id: course.group
    id: course.app
    concurrency: 6
```

Kafka producers:

- replyKafkaTemplate method that sends a request and asynchronously waits for a response. This method is mainly used by the server as it sends a request to a microservice and awaits for a response from them even if it is just a confirmation of execution of the request.
- KafkaTemplate method that sends a request and does not await for a response this is especially well suited when a microservice needs to simply inform another microservice of a specific event.

Kafka listener:

Each microservice listens for a specific topic name and any message with that topic name is received by the microservice. A command name is included in the message that used to direct the request to its destination endpoint.



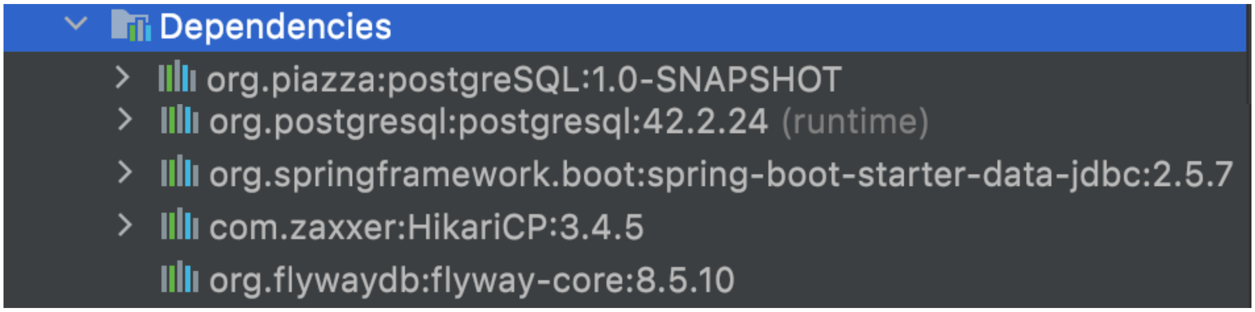
```
@KafkaListener(
        id = "${kafka.consumer.id}",
        topics = "${kafka.topics.course.name}",
        groupId = "${kafka.consumer.group-id}",
        containerFactory = "requestReplyListenerContainerFactory"
)
@SendTo
public Map<String, Object> listener(Map<String, Object> request) {
    try {
        return invokeController.invokeCommand((String) request.get(Constants.COMMAND_NAME_ATTRIBUTE),
                (HashMap<String, Object>) request.get(Constants.REQUEST_BODY_ATTRIBUTE)).get();
    } catch (ExecutionException | InterruptedException e) {
        HashMap<String, Object> errorResponse = new HashMap<>();
        errorResponse.put("message", "Issue Consuming the Request");
        errorResponse.put("status", HttpStatus.BAD_REQUEST);
        return errorResponse;
    }
}
```

Databases

[The first two heading levels get their own paragraph, as shown here. Headings 3, 4, and 5 are run-in headings used at the beginning of the paragraph.]

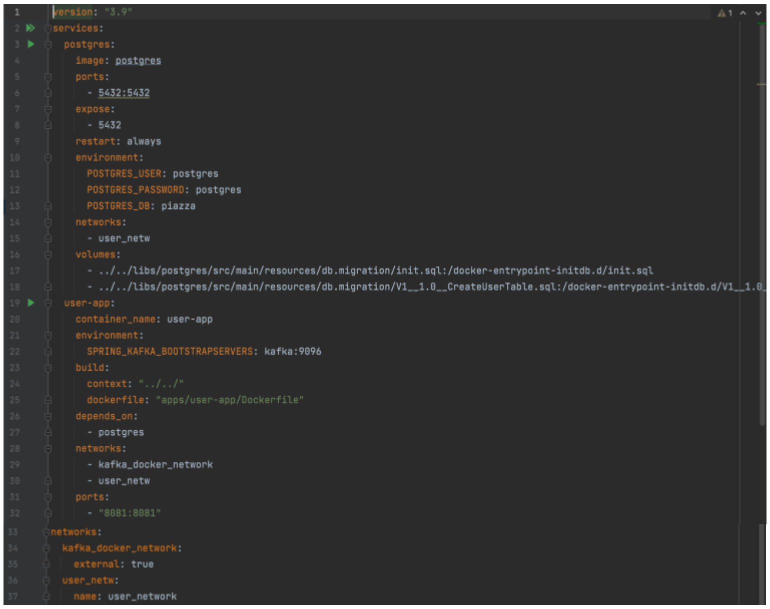
PostgreSQL

Maven Dependencies:



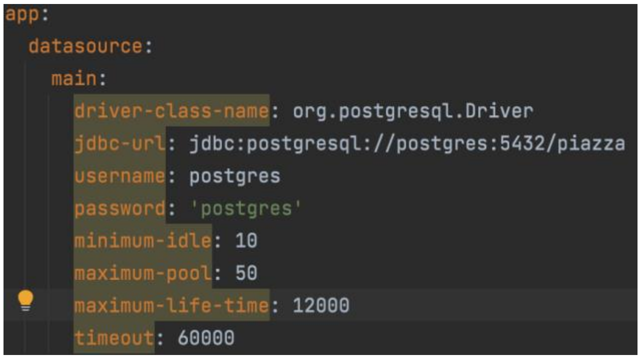
```
Dependencies
  org.piazza:postgreSQL:1.0-SNAPSHOT
  org.postgresql:postgresql:42.2.24 (runtime)
  org.springframework.boot:spring-boot-starter-data-jdbc:2.5.7
  com.zaxxer:HikariCP:3.4.5
  org.flywaydb:flyway-core:8.5.10
```

Docker Image:



```yaml
version: "3.9"
services:
  postgres:
    image: postgres
    ports:
      - 5432:5432
    expose:
      - 5432
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: piazza
    networks:
      - user_netw
    volumes:
      - ../../libs/postgres/src/main/resources/db.migration/init.sql:/docker-entrypoint-initdb.d/init.sql
      - ../../libs/postgres/src/main/resources/db.migration/V1__1.0__CreateUserTable.sql:/docker-entrypoint-initdb.d/V1__1.0_
  user-app:
    container_name: user-app
    environment:
      SPRING_KAFKA_BOOTSTRAPSERVERS: kafka:9096
    build:
      context: "../../"
      dockerfile: "apps/user-app/Dockerfile"
    depends_on:
      - postgres
    networks:
      - kafka_docker_network
      - user_netw
    ports:
      - "8081:8081"
networks:
  kafka_docker_network:
    external: true
  user_netw:
    name: user_network
```

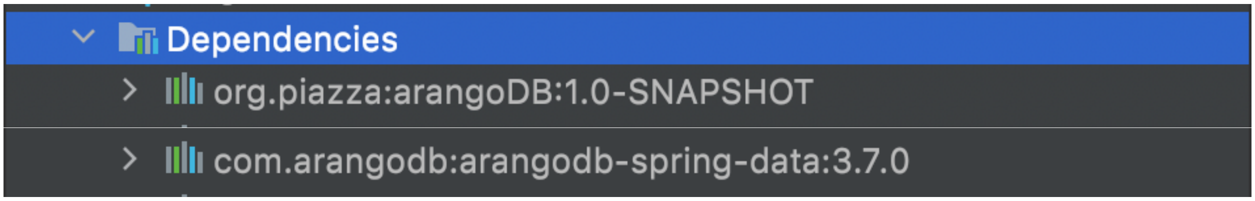
- Usage: The main use of the PostgreSQL database in our project was for the user-app microservice. We stored the user table in this database which consisted of email, password, firstName, lastName and role.
- Pooling: To handle the pooling for the PostgreSQL database a class called HikariCPDataSource can be found in the libs module, submodule PostgreSQL. Hikari is used for pooling with jdbc connections. The default setup is shown below.



```
app:
  datasource:
    main:
      driver-class-name: org.postgresql.Driver
      jdbc-url: jdbc:postgresql://postgres:5432/piazza
      username: postgres
      password: 'postgres'
      minimum-idle: 10
      maximum-pool: 50
      maximum-life-time: 12000
      timeout: 60000
```

ArangoDB

Maven Dependencies:

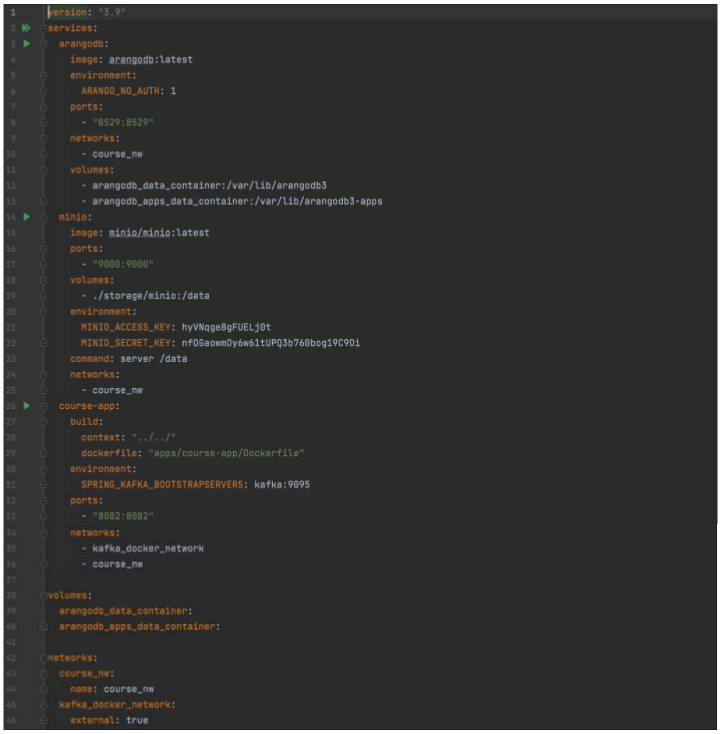


Dependencies
- org.piazza:arangoDB:1.0-SNAPSHOT
- com.arangodb:arangodb-spring-data:3.7.0

Collections: Users, Questions, Answers, Polls, Courses and Reports.

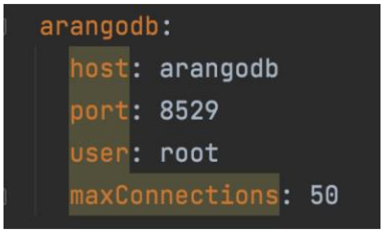
Edges: UserInCourse, UserAnswerPoll, UserCreatePoll, QuestionHasAnswer, QuestionMentionedUser, UserCreateReport, UserBannedFromCourseEdge, UserLikesQuestion, UserReported, UserMakeAnswer, UserLikesAnswerm, UserMakeQuestion, CourseHasPoll and CourseQuestion.

Docker Image:



```yaml
version: "3.9"
services:
  arangodb:
    image: arangodb:latest
    environment:
      ARANGO_NO_AUTH: 1
    ports:
      - "8529:8529"
    networks:
      - course_nw
    volumes:
      - arangodb_data_container:/var/lib/arangodb3
      - arangodb_apps_data_container:/var/lib/arangodb3-apps
  minio:
    image: minio/minio:latest
    ports:
      - "9000:9000"
    volumes:
      - ./storage/minio:/data
    environment:
      MINIO_ACCESS_KEY: hyVNqge8gFUELj0t
      MINIO_SECRET_KEY: nf0Gaowm0y6w61tUPQ3b760bog19C9Di
    command: server /data
    networks:
      - course_nw
  course-app:
    build:
      context: "../../"
      dockerfile: "apps/course-app/Dockerfile"
    environment:
      SPRING_KAFKA_BOOTSTRAPSERVERS: kafka:9095
    ports:
      - "8082:8082"
    networks:
      - kafka_docker_network
      - course_nw

volumes:
  arangodb_data_container:
  arangodb_apps_data_container:

networks:
  course_nw:
    name: course_nw
  kafka_docker_network:
    external: true
```

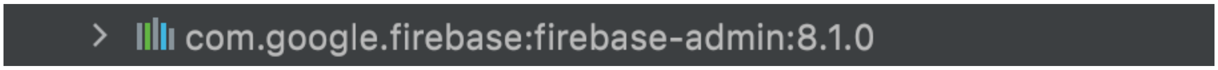
- Usage: The main use of the ArangoDB database in our project was for the course-app microservice. We stored the many tables, edges and graphs in it.
- Pooling: The pooling in Arango is handled by default and the default setup is shown below.



```yaml
arangodb:
  host: arangodb
  port: 8529
  user: root
  maxConnections: 50
```

Firebase

Maven Dependencies:



```
com.google.firebase:firebase-admin:8.1.0
```

Docker Image:

Firebase does not have a docker image as it is a cloud based database.

- Usage: The main use of the Firebase database in our project was for the chat-app and the notifications-app microservices.
- Pooling: The pooling in Firebase is handled remotely.

Caching

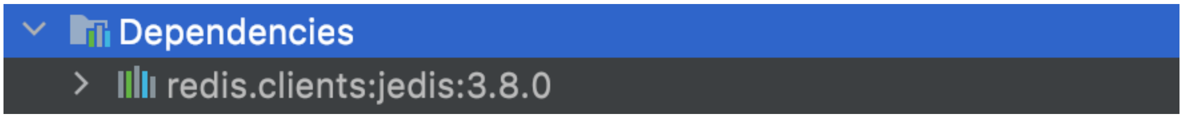
Usage

We used a Redis cache in the server controller module. The cache is used to verify logins. On login and refresh token routes the access token is extracted from the response and stored in the cache. On logout the token is removed from the cache.

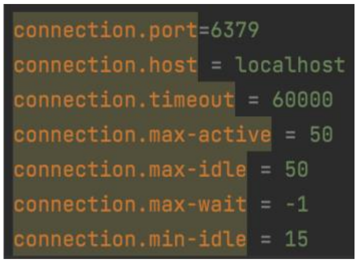
With the exception of the login, refresh and register routes, all other routes require a valid token. In order to verify the token the server first check if the token can be decrypted using the secret key, then it verifies if the token is not expired. If the token is valid the user email and role are passed as attributes in the request. The token is subjected to a final check, if the token is found in the cache then the request is allowed to continue, if it is not found the request is denied access to the server.

Configuration

Maven Dependencies:



Dependencies
redis.clients:jedis:3.8.0



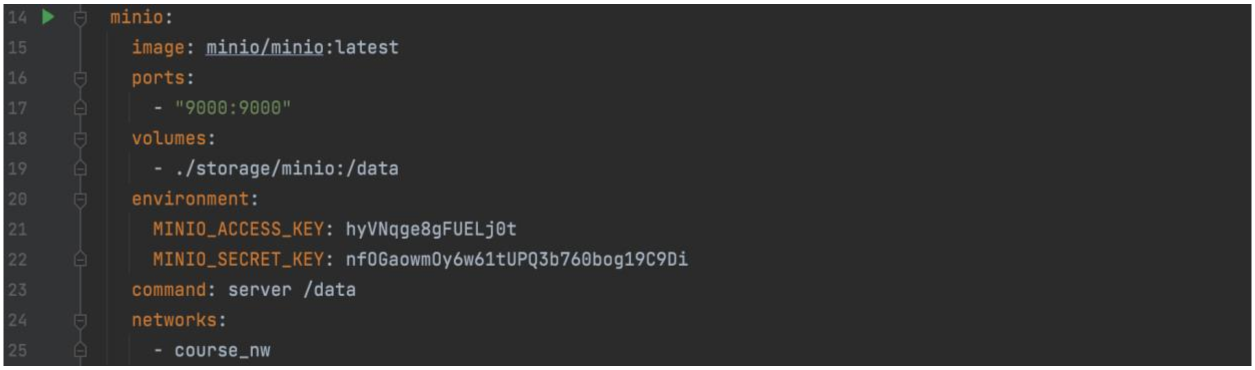
```
connection.port=6379
connection.host = localhost
connection.timeout = 60000
connection.max-active = 50
connection.max-idle = 50
connection.max-wait = -1
connection.min-idle = 15
```

Redis Connection Configuration

Media Storage

Since some images or videos can be used in questions or answers in Piazza. We use MiniIO server to store this media.

Docker Configuration:



```
minio:
  image: minio/minio:latest
  ports:
    - "9000:9000"
  volumes:
    - ./storage/minio:/data
  environment:
    MINIO_ACCESS_KEY: hyVNqge8gFUELj0t
    MINIO_SECRET_KEY: nfOGaowmOy6w61tUPQ3b760bog19C9Di
  command: server /data
  networks:
    - course_nw
```

Testing

Setup

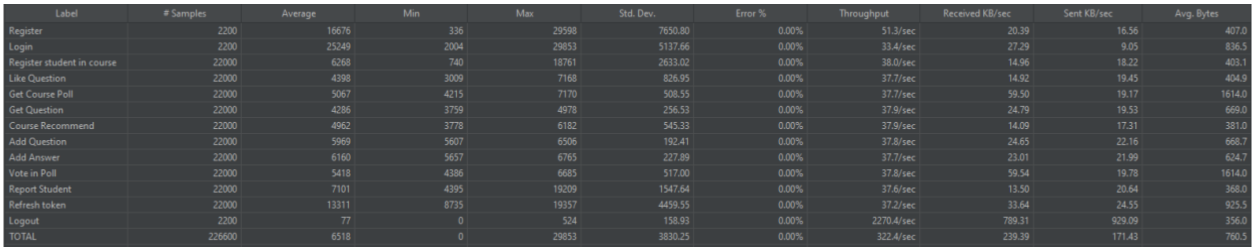
We conducted all of our testing using Apache JMeter in order to analyze how our system performs in a variety of situations. All our tests were conducted on a desktop PC with the configurations shown below.

Operating System	Windows 10
CPU	AMD Ryzen 5 5600X
RAM	32 GB

Endurance Testing

To test the endurance of our system we decided to register and login 2200 users. Then these 2200 users simultaneously send requests to 10 routes 50 times. Then all 2200 users logout.

Table 1.1 shows a summary report of all the routes executed along with their corresponding average, minimum, maximum response times, error rate and throughput among others. As can be seen there was a 0 % error rate and an average throughput of 322.4 requests/sec. The average observed response time was 6518 milliseconds. Tables 1.2, 1.3 and 1.4 show extra information and visualization on the running of the endurance tests.



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Register	2200	16676	336	29598	7650.80	0.00%	51.3/sec	20.39	16.56	407.0
Login	2200	25249	2004	29853	5137.66	0.00%	33.4/sec	27.29	9.05	836.5
Register student in course	22000	6268	740	18761	2633.02	0.00%	38.0/sec	14.96	18.22	403.1
Like Question	22000	4398	3009	7168	826.95	0.00%	37.7/sec	14.92	19.45	404.9
Get Course Poll	22000	5067	4215	7170	508.55	0.00%	37.7/sec	59.50	19.17	1614.0
Get Question	22000	4286	3759	4978	256.53	0.00%	37.9/sec	24.79	19.53	669.0
Course Recommend	22000	4962	3778	6182	545.33	0.00%	37.9/sec	14.09	17.31	381.0
Add Question	22000	5969	5607	6506	192.41	0.00%	37.8/sec	24.65	22.16	668.7
Add Answer	22000	6160	5657	6765	227.89	0.00%	37.7/sec	23.01	21.99	624.7
Vote in Poll	22000	5418	4386	6685	517.00	0.00%	37.8/sec	59.54	19.78	1614.0
Report Student	22000	7101	4395	19209	1547.64	0.00%	37.6/sec	13.50	20.64	368.0
Refresh token	22000	13311	8735	19357	4459.55	0.00%	37.2/sec	33.64	24.55	925.5
Logout	2200	77	0	524	158.93	0.00%	2270.4/sec	789.31	929.09	356.0
TOTAL	226600	6518	0	29853	3830.25	0.00%	322.4/sec	239.39	171.43	760.5

Table 1.1 Summary Report

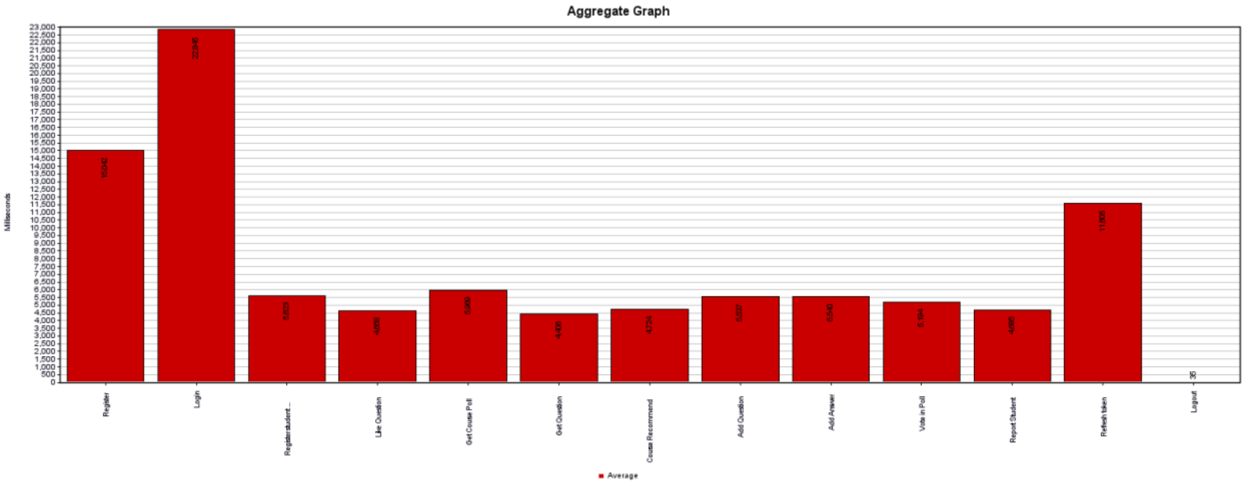


Table 1.2 Aggregate Report

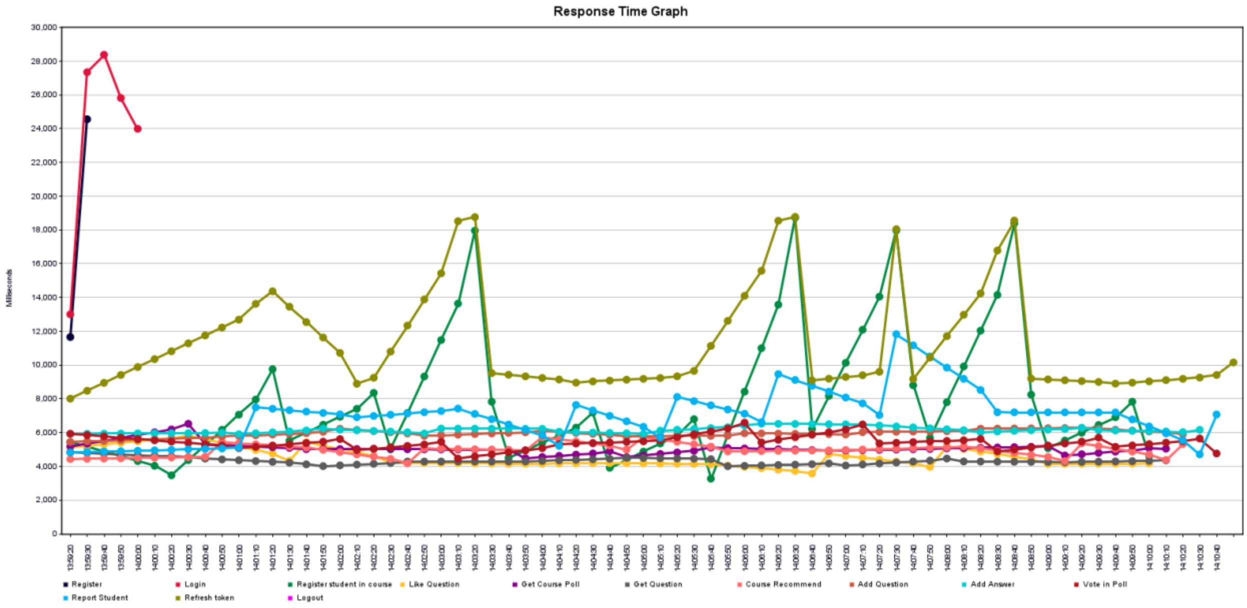
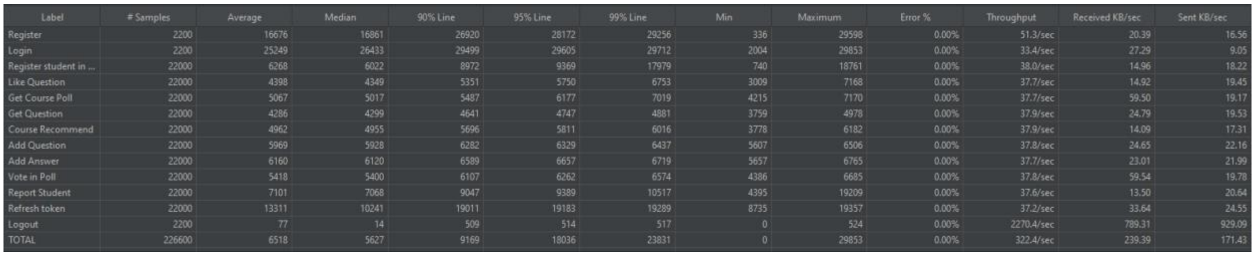


Table 1.3 Response Time Graph



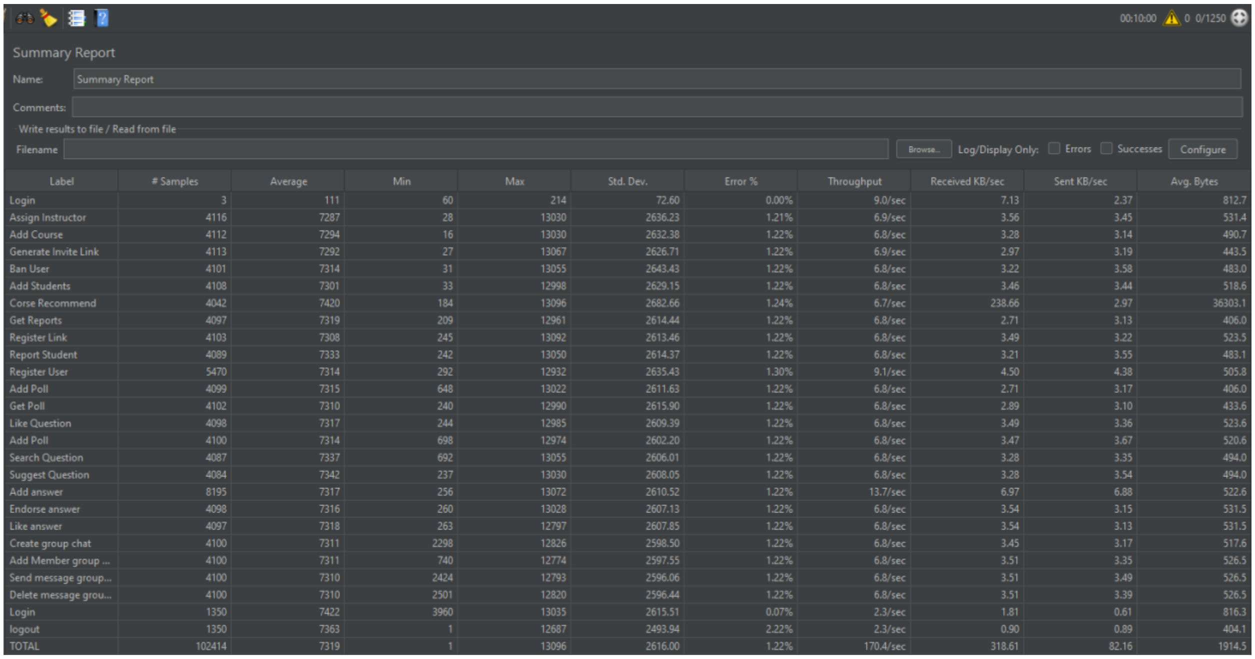
Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
Register	2200	16676	16861	26920	28172	29256	336	29598	0.00%	51.3/sec	20.39	16.56
Login	2200	25249	26433	29499	29605	29712	2004	29853	0.00%	33.4/sec	27.29	9.05
Register student in ...	22000	6268	6022	8972	9369	17979	740	18761	0.00%	38.0/sec	14.96	18.22
Like Question	22000	4398	4349	5351	5750	6753	3009	7168	0.00%	37.7/sec	14.92	19.45
Get Course Poll	22000	5067	5017	5487	6177	7019	4215	7170	0.00%	37.7/sec	59.50	19.17
Get Question	22000	4286	4299	4641	4747	4881	3759	4978	0.00%	37.9/sec	24.79	19.53
Course Recommend	22000	4962	4955	5696	5811	6016	3778	6182	0.00%	37.9/sec	14.09	17.31
Add Question	22000	5969	5928	6282	6329	6437	5607	6506	0.00%	37.8/sec	24.65	22.16
Add Answer	22000	6160	6120	6589	6657	6719	5657	6765	0.00%	37.7/sec	23.01	21.99
Vote in Poll	22000	5418	5400	6107	6262	6574	4386	6685	0.00%	37.8/sec	59.54	19.78
Report Student	22000	7101	7068	9047	9389	10517	4395	19209	0.00%	37.6/sec	13.50	20.64
Refresh token	22000	13311	10241	19011	19183	19289	8735	19357	0.00%	37.2/sec	33.64	24.55
Logout	2200	77	14	509	514	517	0	524	0.00%	2270.4/sec	789.31	929.09
TOTAL	226600	6518	5627	9169	18036	23831	0	29853	0.00%	322.4/sec	239.39	171.43

Table 1.4 Aggregate Report

Load Testing

To test the load of our system we decided to run 25 requests on 25 routes using 50 users simultaneously. These routes are from all our microservices and when the requests are running on an infinite loop form 10 minutes. The goal from this is that all these endpoints are always sustaining 50 requests for 10 minutes to see when and if the system fails.

Table 2.1 shows a summary report of all the routes executed along with their corresponding average, minimum, maximum response times, error rate and throughput among others. As can be seen there was a 1.22 % error rate and an average throughput of 170.4 requests/sec. The average observed response time was 7318 milliseconds. Figures 2.2, 2.3 and 2.4 show extra information and visualization on the running of the endurance tests. What we can deduce from our observations is that after 9:58 minutes the load on the system became too great and the system dropped.



Summary Report

Name: Summary Report

Comments:

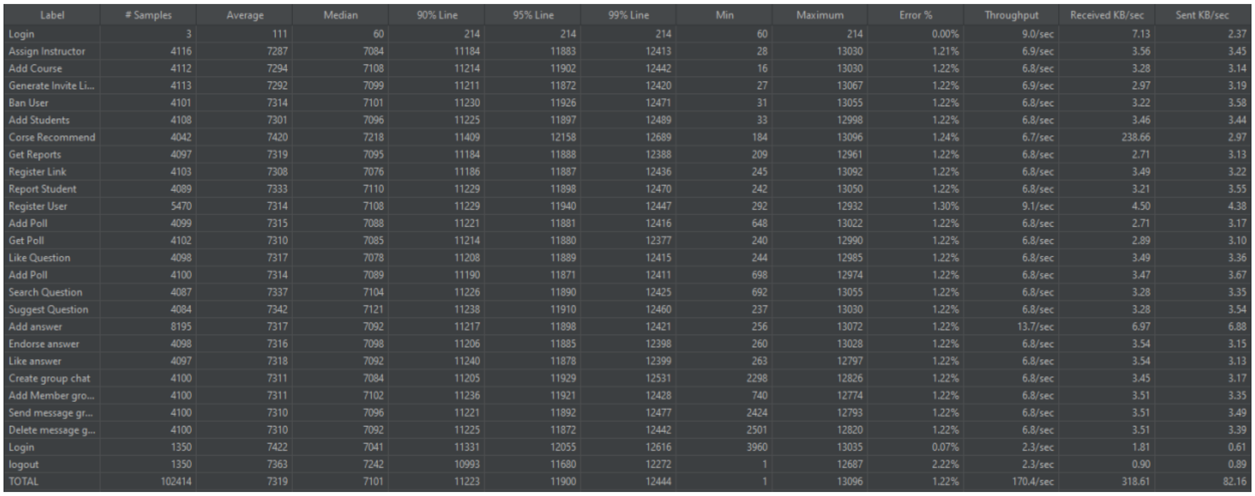
Write results to file / Read from file

Filename

Log/Display Only: □ Errors □ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Login	3	111	60	214	72.60	0.00%	9.0/sec	7.13	2.37	812.7
Assign Instructor	4116	7287	28	13030	2636.23	1.21%	6.9/sec	3.56	3.45	531.4
Add Course	4112	7294	16	13030	2632.38	1.22%	6.8/sec	3.28	3.14	490.7
Generate Invite Link	4113	7292	27	13067	2626.71	1.22%	6.9/sec	2.97	3.19	443.5
Ban User	4101	7314	31	13055	2643.43	1.22%	6.8/sec	3.22	3.58	483.0
Add Students	4108	7301	33	12998	2629.15	1.22%	6.8/sec	3.46	3.44	518.6
Corse Recommend	4042	7420	184	13096	2682.66	1.24%	6.7/sec	238.66	2.97	36303.1
Get Reports	4097	7319	209	12961	2614.44	1.22%	6.8/sec	2.71	3.13	406.0
Register Link	4103	7308	245	13092	2613.46	1.22%	6.8/sec	3.49	3.22	523.5
Report Student	4089	7333	242	13050	2614.37	1.22%	6.8/sec	3.21	3.55	483.1
Register User	5470	7314	292	12932	2635.43	1.30%	9.1/sec	4.50	4.38	505.8
Add Poll	4099	7315	648	13022	2611.63	1.22%	6.8/sec	2.71	3.17	406.0
Get Poll	4102	7310	240	12990	2615.90	1.22%	6.8/sec	2.89	3.10	433.6
Like Question	4098	7317	244	12985	2609.39	1.22%	6.8/sec	3.49	3.36	523.6
Add Poll	4100	7314	698	12974	2602.20	1.22%	6.8/sec	3.47	3.67	520.6
Search Question	4087	7337	692	13055	2606.01	1.22%	6.8/sec	3.28	3.35	494.0
Suggest Question	4084	7342	237	13030	2608.05	1.22%	6.8/sec	3.28	3.54	494.0
Add answer	8195	7317	256	13072	2610.52	1.22%	13.7/sec	6.97	6.88	522.6
Endorse answer	4098	7316	260	13028	2607.13	1.22%	6.8/sec	3.54	3.15	531.5
Like answer	4097	7318	263	12797	2607.85	1.22%	6.8/sec	3.54	3.13	531.5
Create group chat	4100	7311	2298	12826	2598.50	1.22%	6.8/sec	3.45	3.17	517.6
Add Member group ...	4100	7311	740	12774	2597.55	1.22%	6.8/sec	3.51	3.35	526.5
Send message group...	4100	7310	2424	12793	2596.06	1.22%	6.8/sec	3.51	3.49	526.5
Delete message grou...	4100	7310	2501	12820	2596.44	1.22%	6.8/sec	3.51	3.39	526.5
Login	1350	7422	3960	13035	2615.51	0.07%	2.3/sec	1.81	0.61	816.3
logout	1350	7363	1	12687	2493.94	2.22%	2.3/sec	0.90	0.89	404.1
TOTAL	102414	7319	1	13096	2616.00	1.22%	170.4/sec	318.61	82.16	1914.5

Table 2.1 Summary Report



Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
Login	3	111	60	214	214	214	60	214	0.00%	9.0/sec	7.13	2.37
Assign Instructor	4116	7287	7084	11184	11883	12413	28	13030	1.21%	6.9/sec	3.56	3.45
Add Course	4112	7294	7108	11214	11902	12442	16	13030	1.22%	6.8/sec	3.28	3.14
Generate Invite Li...	4113	7292	7099	11211	11872	12420	27	13067	1.22%	6.9/sec	2.97	3.19
Ban User	4101	7314	7101	11230	11926	12471	31	13055	1.22%	6.8/sec	3.22	3.58
Add Students	4108	7301	7096	11225	11897	12489	33	12998	1.22%	6.8/sec	3.46	3.44
Corse Recommend	4042	7420	7218	11409	12158	12689	184	13096	1.24%	6.7/sec	238.66	2.97
Get Reports	4097	7319	7095	11184	11888	12388	209	12961	1.22%	6.8/sec	2.71	3.13
Register Link	4103	7308	7076	11186	11887	12436	245	13092	1.22%	6.8/sec	3.49	3.22
Report Student	4089	7333	7110	11229	11898	12470	242	13050	1.22%	6.8/sec	3.21	3.55
Register User	5470	7314	7108	11229	11940	12447	292	12932	1.30%	9.1/sec	4.50	4.38
Add Poll	4099	7315	7088	11221	11881	12416	648	13022	1.22%	6.8/sec	2.71	3.17
Get Poll	4102	7310	7085	11214	11880	12377	240	12990	1.22%	6.8/sec	2.89	3.10
Like Question	4098	7317	7078	11208	11889	12415	244	12985	1.22%	6.8/sec	3.49	3.36
Add Poll	4100	7314	7089	11190	11871	12411	698	12974	1.22%	6.8/sec	3.47	3.67
Search Question	4087	7337	7104	11226	11890	12425	692	13055	1.22%	6.8/sec	3.28	3.35
Suggest Question	4084	7342	7121	11238	11910	12460	237	13030	1.22%	6.8/sec	3.28	3.54
Add answer	8195	7317	7092	11217	11898	12421	256	13072	1.22%	13.7/sec	6.97	6.88
Endorse answer	4098	7316	7098	11206	11885	12398	260	13028	1.22%	6.8/sec	3.54	3.15
Like answer	4097	7318	7092	11240	11878	12399	263	12797	1.22%	6.8/sec	3.54	3.13
Create group chat	4100	7311	7084	11205	11929	12531	2298	12826	1.22%	6.8/sec	3.45	3.17
Add Member gro...	4100	7311	7102	11236	11921	12428	740	12774	1.22%	6.8/sec	3.51	3.35
Send message gr...	4100	7310	7096	11221	11892	12477	2424	12793	1.22%	6.8/sec	3.51	3.49
Delete message g...	4100	7310	7092	11225	11872	12442	2501	12820	1.22%	6.8/sec	3.51	3.39
Login	1350	7422	7041	11331	12055	12616	3960	13035	0.07%	2.3/sec	1.81	0.61
logout	1350	7363	7242	10993	11680	12272	1	12687	2.22%	2.3/sec	0.90	0.89
TOTAL	102414	7319	7101	11223	11900	12444	1	13096	1.22%	170.4/sec	318.61	82.16

Table 2.2 Aggregate Report

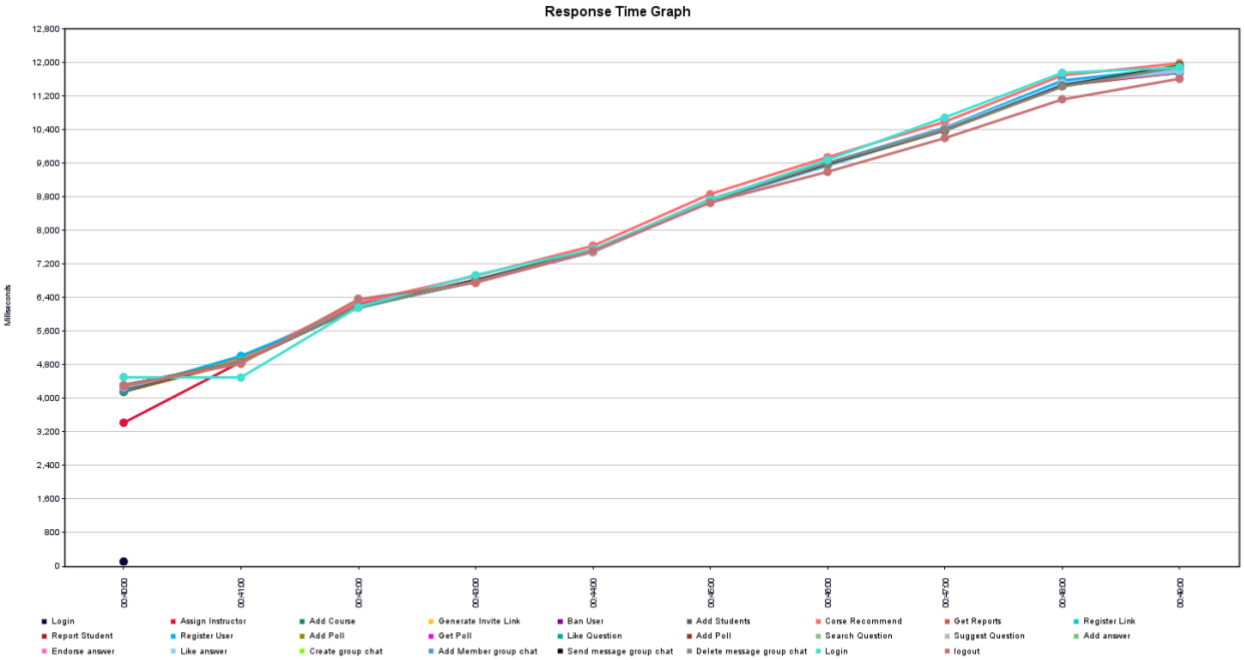


Table 2.3 Response Time Graph

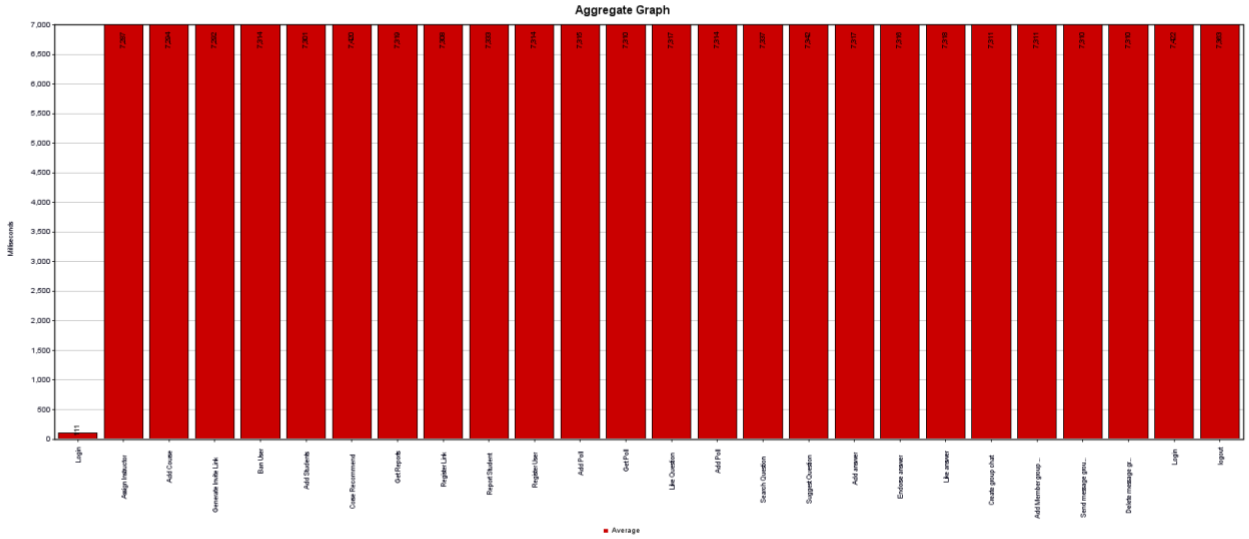
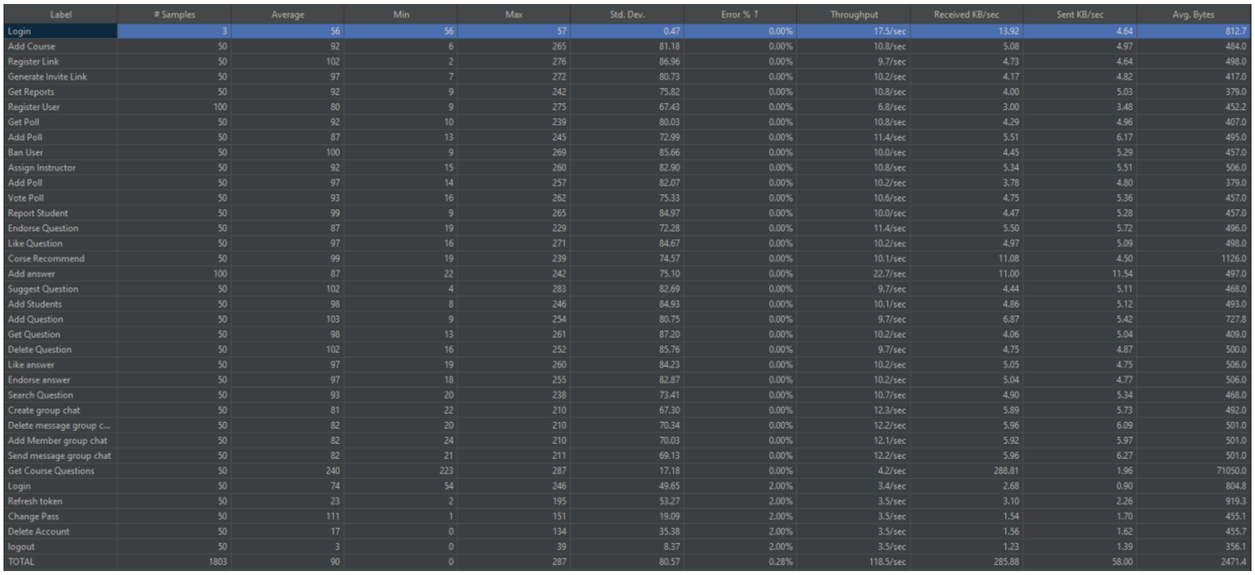


Table 2.4 Aggregate Graph

Performance Testing

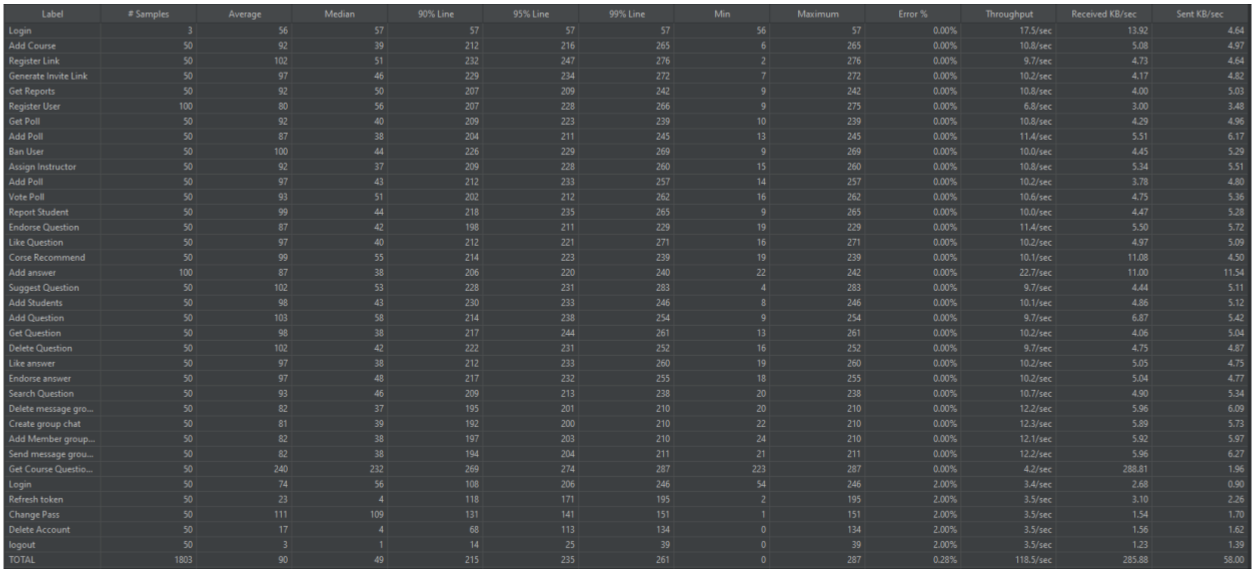
To test the performance of our system we decided to run 50 sequential requests on all routes in the system from one user. These routes are from all our microservices totaling 34 routes. The goal from this is that all these endpoints are always sustaining a request and that each endpoint is tested for 50 times.

Table 3.1 shows a summary report of all the routes executed along with their corresponding average, minimum, maximum response times, error rate and throughput among others. As can be seen there was a 0.78 % error rate, most likely due to previous info in the databases interfering with initial user-app requests. The average throughput was 118.5 requests/sec. The average observed response time was 90 milliseconds. Figures 3.2 and 3.3 show extra information and visualization on the running of the endurance tests. What we can deduce from our observations is that due to the simultaneous load on all endpoints the throughput decreased, however the average response time was remarkably low.



Label	# Samples	Average	Min	Max	Std. Dev.	Error % ↑	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Login	3	56	56	57	0.47	0.00%	17.5/sec	13.92	4.64	812.7
Add Course	50	92	6	265	81.18	0.00%	10.8/sec	5.08	4.97	484.0
Register Link	50	102	2	276	86.96	0.00%	9.7/sec	4.73	4.64	498.0
Generate Invite Link	50	97	7	272	80.73	0.00%	10.2/sec	4.17	4.82	417.0
Get Reports	50	92	9	242	75.82	0.00%	10.8/sec	4.00	5.03	379.0
Register User	100	80	9	275	67.43	0.00%	6.8/sec	3.00	3.48	452.2
Get Poll	50	92	10	239	80.03	0.00%	10.8/sec	4.29	4.96	407.0
Add Poll	50	87	13	245	72.99	0.00%	11.4/sec	5.51	6.17	495.0
Ban User	50	100	9	269	85.66	0.00%	10.0/sec	4.45	5.29	457.0
Assign Instructor	50	92	15	260	82.90	0.00%	10.8/sec	5.34	5.51	506.0
Add Poll	50	97	14	257	82.07	0.00%	10.2/sec	3.78	4.80	379.0
Vote Poll	50	93	16	262	75.33	0.00%	10.6/sec	4.75	5.36	457.0
Report Student	50	99	9	265	84.97	0.00%	10.0/sec	4.47	5.28	457.0
Endorse Question	50	87	19	229	72.28	0.00%	11.4/sec	5.50	5.72	496.0
Like Question	50	97	16	271	84.67	0.00%	10.2/sec	4.97	5.09	498.0
Corse Recommend	50	99	19	239	74.57	0.00%	10.1/sec	11.08	4.50	1126.0
Add answer	100	87	22	242	75.10	0.00%	22.7/sec	11.00	11.54	497.0
Suggest Question	50	102	4	283	82.69	0.00%	9.7/sec	4.44	5.11	468.0
Add Students	50	98	8	246	84.93	0.00%	10.1/sec	4.86	5.12	493.0
Add Question	50	103	9	254	80.75	0.00%	9.7/sec	6.87	5.42	727.8
Get Question	50	98	13	261	87.20	0.00%	10.2/sec	4.06	5.04	409.0
Delete Question	50	102	16	252	85.76	0.00%	9.7/sec	4.75	4.87	500.0
Like answer	50	97	19	260	84.23	0.00%	10.2/sec	5.05	4.75	506.0
Endorse answer	50	97	18	255	82.87	0.00%	10.2/sec	5.04	4.77	506.0
Search Question	50	93	20	238	73.41	0.00%	10.7/sec	4.90	5.34	468.0
Create group chat	50	81	22	210	67.30	0.00%	12.3/sec	5.89	5.73	492.0
Delete message group c...	50	82	20	210	70.34	0.00%	12.2/sec	5.96	6.09	501.0
Add Member group chat	50	82	24	210	70.03	0.00%	12.1/sec	5.92	5.97	501.0
Send message group chat	50	82	21	211	69.13	0.00%	12.2/sec	5.96	6.27	501.0
Get Course Questions	50	240	223	287	17.18	0.00%	4.2/sec	288.81	1.96	71050.0
Login	50	74	54	246	49.65	2.00%	3.4/sec	2.68	0.90	804.8
Refresh token	50	23	2	195	53.27	2.00%	3.5/sec	3.10	2.26	919.3
Change Pass	50	111	1	151	19.09	2.00%	3.5/sec	1.54	1.70	455.1
Delete Account	50	17	0	134	35.38	2.00%	3.5/sec	1.56	1.62	455.7
logout	50	3	0	39	8.37	2.00%	3.5/sec	1.23	1.39	356.1
TOTAL	1803	90	0	287	80.57	0.28%	118.5/sec	285.88	58.00	2471.4

Table 3.1 Summary Report



Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
Login	3	56	57	57	57	57	56	57	0.00%	17.5/sec	13.92	4.64
Add Course	50	92	39	212	216	265	6	265	0.00%	10.8/sec	5.08	4.97
Register Link	50	102	51	232	247	276	2	276	0.00%	9.7/sec	4.73	4.64
Generate Invite Link	50	97	46	229	234	272	7	272	0.00%	10.2/sec	4.17	4.82
Get Reports	50	92	50	207	209	242	9	242	0.00%	10.8/sec	4.00	5.03
Register User	100	80	56	207	228	266	9	275	0.00%	6.8/sec	3.00	3.48
Get Poll	50	92	40	209	223	239	10	239	0.00%	10.8/sec	4.29	4.96
Add Poll	50	87	38	204	211	245	13	245	0.00%	11.4/sec	5.51	6.17
Ban User	50	100	44	226	229	269	9	269	0.00%	10.0/sec	4.45	5.29
Assign Instructor	50	92	37	209	228	260	15	260	0.00%	10.8/sec	5.34	5.51
Add Poll	50	97	43	212	233	257	14	257	0.00%	10.2/sec	3.78	4.80
Vote Poll	50	93	51	202	212	262	16	262	0.00%	10.6/sec	4.75	5.36
Report Student	50	99	44	218	235	265	9	265	0.00%	10.0/sec	4.47	5.28
Endorse Question	50	87	42	198	211	229	19	229	0.00%	11.4/sec	5.50	5.72
Like Question	50	97	40	212	221	271	16	271	0.00%	10.2/sec	4.97	5.09
Corse Recommend	50	99	55	214	223	239	19	239	0.00%	10.1/sec	11.08	4.50
Add answer	100	87	38	206	220	240	22	242	0.00%	22.7/sec	11.00	11.54
Suggest Question	50	102	53	228	231	283	4	283	0.00%	9.7/sec	4.44	5.11
Add Students	50	98	43	230	233	246	8	246	0.00%	10.1/sec	4.86	5.12
Add Question	50	103	58	214	238	254	9	254	0.00%	9.7/sec	6.87	5.42
Get Question	50	98	38	217	244	261	13	261	0.00%	10.2/sec	4.06	5.04
Delete Question	50	102	42	222	231	252	16	252	0.00%	9.7/sec	4.75	4.87
Like answer	50	97	38	212	233	260	19	260	0.00%	10.2/sec	5.05	4.75
Endorse answer	50	97	48	217	232	255	18	255	0.00%	10.2/sec	5.04	4.77
Search Question	50	93	46	209	213	238	20	238	0.00%	10.7/sec	4.90	5.34
Delete message gro...	50	82	37	195	201	210	20	210	0.00%	12.2/sec	5.96	6.09
Create group chat	50	81	39	192	200	210	22	210	0.00%	12.3/sec	5.89	5.73
Add Member group...	50	82	38	197	203	210	24	210	0.00%	12.1/sec	5.92	5.97
Send message grou...	50	82	38	194	204	211	21	211	0.00%	12.2/sec	5.96	6.27
Get Course Questio...	50	240	232	269	274	287	223	287	0.00%	4.2/sec	288.81	1.96
Login	50	74	56	108	206	246	54	246	2.00%	3.4/sec	2.68	0.90
Refresh token	50	23	4	118	171	195	2	195	2.00%	3.5/sec	3.10	2.26
Change Pass	50	111	109	131	141	151	1	151	2.00%	3.5/sec	1.54	1.70
Delete Account	50	17	4	68	113	134	0	134	2.00%	3.5/sec	1.56	1.62
logout	50	3	1	14	25	39	0	39	2.00%	3.5/sec	1.23	1.39
TOTAL	1803	90	49	215	235	261	0	287	0.28%	118.5/sec	285.88	58.00

Table 3.2 Aggregate Report

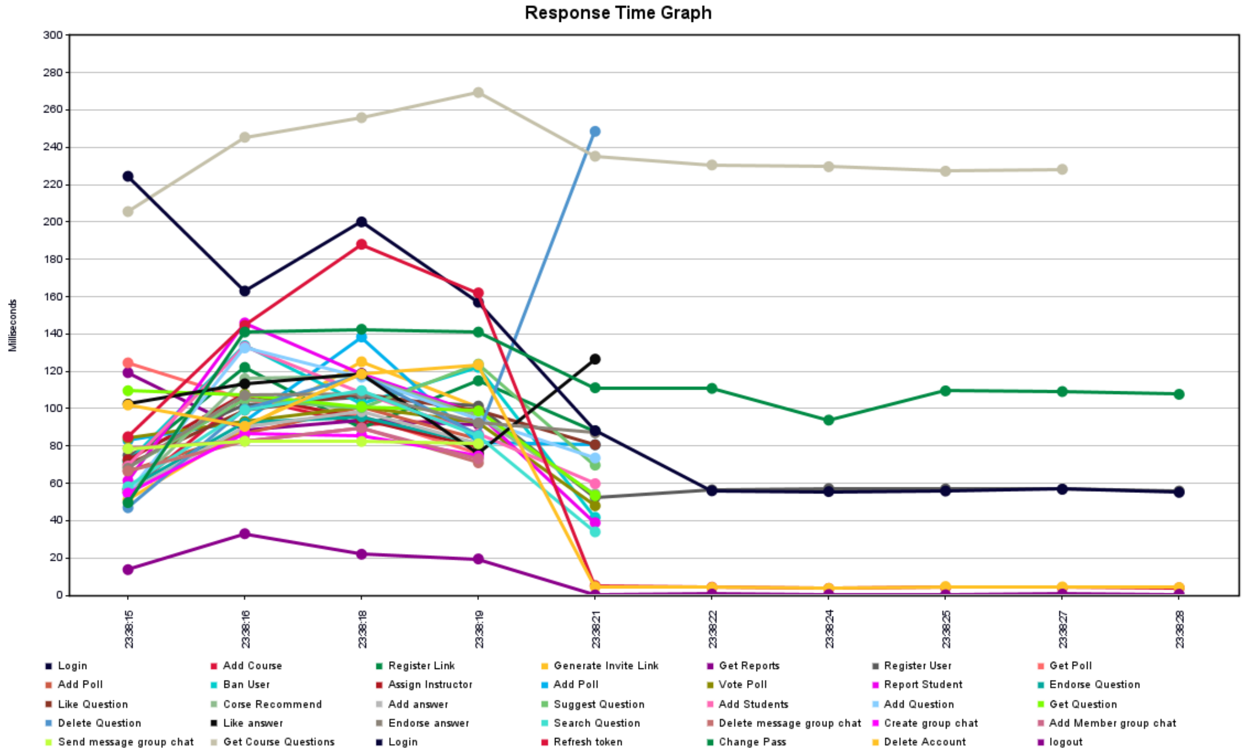


Figure 3.3 Response Time Graph

Packaging and Dockerization

Each microservice was dockerized individually and can be run independently. The Apache Kafka message queue was also dockerized. These containers can be run separately. The server was also dockerized.