Tampere University

Sheikh Saimul Haque Nazeef Bin Enam

OPTIMIZING THE EFFICIENCY OF THE DATA ANALYTICS FRAMEWORK USING MICROSERVICE ARCHITECTURE

Faculty of Information Technology and Communication Sciences
Master of Science Thesis
May 2020

ABSTRACT

Sheikh Saimul Haque Nazeef Bin Enam: Optimizing the Efficiency of the Data Analytics Framework using Microservice Architecture
Master of Science Thesis
Tampere University
Master's Degree Programme in Information Technology
May 2020
Examiners: Associate Professor Kari Systä, Tampere University and Johanna Kalliomäki, Chief Operating Officer, Cloubi Ltd.

This thesis describes the backend of the new data analytics framework that has been designed and developed for the new reporting feature of Cloubi. Cloubi is a web application used for creating and distributing learning materials. The reporting feature is used by the students and teachers to check the performance of the students. The new data analytics framework was developed using microservice architecture and aims to be faster in terms of fetching the students' data compared to the previous data analytics framework developed using monolithic architecture.

The design of the previous data analytics was kept in mind while designing the new data analytics framework. The implementation included the creation of a microservice consisting of an application that is used for getting the events from Cloubi via Kafka and filling up the database used by the microservice. The application can then be used to make queries retrieving the students' data required for the reporting feature.

After the microservice was fully functional and was interacting with Cloubi, a set of integration tests were implemented to check whether the individual modules were working as supposed to.

At the end of the thesis, a comparison was done between the previous and the new data analytics framework to prove that the new data analytics framework was better and served the goal of the thesis in optimizing the performance of the reporting feature using the new framework compared to the previous one.

Keywords: backend, data analytics framework, architecture, microservice, monolith, report

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

PREFACE

Firstly, I would like to thank Cloubi Ltd for giving me an opportunity to work on my thesis. While working on the thesis I had the scope to learn from experienced people, their guidelines and daily follow up of my thesis work inspired me to complete the work on due time. I want to thank my company supervisor Johanna Kalliomäki for her constant support and encouragement in the thesis work. I am also thankful to Jarno Saarinen, Antti Hietasaari, and Sami Kuivasaari from the company for their valuable feedback on the development work done for the thesis.

Next, I would like to express my deepest gratitude towards my thesis supervisor Kari Systä from Tampere University, for his continuous support and day-to-day feedback on the thesis writing which helped a lot in shaping up the thesis.

Finally, I would like to thank my family for their love and support from overseas.

Tampere, 03 May 2020

Sheikh Saimul Haque Nazeef Bin Enam

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

LMS	Learning Management Systems
CMS	Content Management Systems
REST	Representational State Transfer
API	Application Programming Interface
CRUD	Create, Read, Update and Delete
JSON	JavaScript Object Notation
XML	eXtensible Markup Language
NoSQL	Non-Structured Query Language
HTTP	HyperText Transfer Protocol
URI	Unique Resource Identifier
CLI	Command-Line Interface
IP	Internet Protocol
JAR	Java Archive
SE	Standard Error
AWS	Amazon Web Services
C1	Cloubi 1
C2	Cloubi 2
M1	Material 1
M2	Material 2
G1	Student Group 1
G2	Student Group 2
G3	Student Group 3

1. INTRODUCTION

This thesis describes the backend of the new data analytics framework developed for Cloubi Ltd. The main product of Cloubi Ltd. is *Cloubi* [1] which is an e-learning platform where education publishers can publish their content. Usually, the content is created by editors and it is then used by teachers and students. Content usually consists of a material, e.g. an e-book. Each material has pages and each page can have static or interactive content. Each interactive content can have attributes like score, progress, and time taken, etc. A student interacts with those interactive contents like tasks (exercises in the material) and their progress is stored or updated into the databases that Cloubi uses. A teacher can check the progress, scores, and answers for a particular group of students who have access to the material. A student can check his or her progress. This feature is called *reporting*. The scope of this thesis is limited to the reporting feature which is a small part of Cloubi.

The previous data analytics framework used to serve the purpose of this report generation. A report is used to get the overall performance of a student or a group of students in each material. The previous data analytics framework used for generating the reports was based on a monolithic architecture and was comparatively slow to load since it used to get data from the databases that Cloubi uses. Due to these performance issues, the reports used to be static and would not show the realtime progress of the students as they completed the tasks. Therefore, this resulted in a need for developing a newer version of the reporting feature.

This thesis focuses on a proof of concept which includes the design and implementation of the backend of the new data analytics framework using microservice architecture which would be much more efficient and at the same time more advanced than the previous one. The main goal of the new data analytics framework is to create better ways to utilize the data that is being collected from the students when they are interacting with Cloubi and is vital for Cloubi Ltd. in the sense that it results in better usability for the teachers and students. This means reports that update the students' data fast enough, in other words, dynamic reporting. The new reporting will also allow publishers to customize the report view in each material in whichever way they want to provide better visualization of the user data.

The limitations of this thesis were initially no one was sure of what kind of data was needed for the new reporting, no user tests were possible at the time of the development

of the backend of the new data analytics framework. During the time of development of the backend, the frontend development was not even started, no mockups were prepared and so it was assumed that it needed similar data that the previous reporting was using. This scope of this thesis is limited to the development of the backend of the new data analytics framework. The previous data analytics framework was developed by Cloubi and is described to give a brief context of how the previous reporting feature was implemented and is mainly used for comparison with the new data analytics framework that has been solely developed by the writer while working for Cloubi.

The research questions that would be answered in this thesis are:

- What is the optimal architecture for the development of the new data analytics framework?
- What is the performance of the new data analytics framework compared to the previous one?

In addition to answering these questions, the thesis is structured accordingly. The first chapter gives a brief introduction to the thesis. The second chapter describes different web service architectural styles and some of the concepts and technologies with respect to the thesis work. The third chapter gives a description of what Cloubi is on an architectural level and also describes the previous data analytics framework. Then the fourth chapter describes the reason for selecting the microservice architecture, followed by the design and implementation of the new data analytics framework along with a set of integration tests. At the end of the fourth chapter, the implementation is compared with related work. The fifth chapter evaluates the performance of the previous and the new data analytics framework. Finally, to conclude there will be a brief summary of the thesis in the conclusion chapter.

2. THEORETICAL BACKGROUND

This chapter gives a theoretical understanding of some of the web service architectural styles such as REST and GraphQL. This chapter also gives a brief understanding of concepts, and technologies such as Microservices, MongoDB, Apache Kafka, Spring Boot, and Docker.

2.1 REST

REST stands for Representational State Transfer and is an architectural style used for developing loosely coupled applications over HTTP (HyperText Transfer Protocol) and is used for creating web services. It defines a set of rules that need to be followed for developing a RESTful API. An API is the application programming interface that acts as a communicator between two components of a software. The rules for REST [2][3] are stated as follows:

1. Client-Server

   This rule suggests that the REST Application must follow a client-server architecture. This means that the client and the server are separated and can be developed independently. The client does not know anything about the business logic or the data layer. Similarly, the server does not know anything about the frontend user interface. The client only knows the resource's Unique Resource Identifiers (URIs) for making CRUD (Create, Read, Update, and Delete) operations on the server-side.

2. Stateless

   This rule suggests that the server does not store any session data and the communication between the client and the server is stateless. This means that all the information required for understanding a request is present within the request.

3. Cache

   This rule states that responses should be cacheable if possible. This means that it should be possible to include the data on whether a response can be cacheable or not. This means that for subsequent requests the client can retrieve response from the cache and thereby reducing the server load.

4. Uniform interface

   This rule is self-explanatory by its name, uniform interface acts as a differentiator between REST APIs and Non-REST APIs. It consists of four key elements [3]:

   - Identification of resources - usually done by URIs.
   - Manipulation of resources through representations - the client can change or delete the resources in the server through the representation that the client has.
   - Self-descriptive messages for each request - each request contains messages on how to handle the request.
   - Hypermedia as the engine of application state - this means that the response consists of links to other resources that the clients can access.

5. Layered

   This rule states that the architecture allows the composition of several layers where each layer does not have any information on the other layers apart from its intermediate layer. Therefore, this reduces the complexity that can be introduced in a single layer.

6. Code on demand

   This rule is optional. This rule states that along with data, the server can provide the client with the executable code.

Therefore, the main idea of REST is that everything is a resource that's identified by a URI. In its simplest form, a request is made via the resource's URI and then this fetches a response or does something in the server according to the type of the request.

2.2 GraphQL

GraphQL is a new and modern architectural style for creating web services and takes a different and flexible approach than REST. Basically, it is a query language for API that makes communication between the client and the server easier [4]. The main difference is that it does not deal with dedicated resources rather everything is considered as a graph in which everything is assumed to be connected. The request can be tailored to fetch more than one resource from different entities with a single query from the server.

Some features of GraphQL [5] are:

1. Defines a definite shape

   GraphQL query response has a similar format to that of the query. This makes it easier for a developer to shape data according to the need of the application.

2. Hierarchical

   GraphQL has a hierarchical nature. This means that the data is usually maintained as having a hierarchical relationship similar to that of the graph-data structures.

3. Strongly typed

   Each level of the query describes a particular type and each type consists of a set of fields. This allows GraphQL to provide descriptive error messages even before executing the query.

4. Use of existing code

   GraphQL API can make use of the existing data and code. Each field of the particular type of data is provided with a function and GraphQL calls the functions maintaining optimum performance.

5. Introspective

   This is an optional feature. It allows a developer to navigate to the particular type of data in the server without even executing the query. It allows developers to add new fields to existing queries since they can see the actual data setup [6].

6. Version free

   GraphQL discourages versioning because the client is the one that determines what fields or values need to be returned in the response.

Despite the features or benefits, this architecture has a major disadvantage, it uses a single endpoint for every operation without following the HTTP specification for caching [7]. Caching is often important since it helps to reduce network overload.

2.3 Microservices

The concept *microservices* emerged from the idea that dealt with the long going issue in software development in which there was the trend of developing monolithic applications. These monolithic applications were larger in size and had all the functionalities tied together in a single application [8]. Any minor code change would have caused the entire

application to be re-built and re-deployed, thereby increasing the complexity of the entire application. Microservices consist of smaller independent applications that traditionally have one functionality or one purpose [9]. These applications can then be piled up to form a large application with lots of different functionality.

The term small in microservice refers to as small as possible. A smaller codebase allows for easy debugging in cases where there might be problems and so those bugs can be solved independently regardless of the other service. These smaller services can be developed by a smaller team and so increasing the benefits of putting smaller teams in the development of each service. If a service is not required it can be easily removed without hampering the other services or the whole application [9].

Each microservice is autonomous which means that it can be deployed to single or multiple servers. There should be a limited amount of sharing between the microservices to avoid too many dependencies amongst themselves. This allows the application to be scaled up and down as required. Also, each microservice can have its own technology and database. This results in getting the best performance out of the whole application allowing developers to choose a technology stack that is best suited for that particular job. For example, an application can be made with two microservices, where service 1 can be Java application using MongoDB, service 2 can be a Golang application using GraphDB.

Microservices tend to be *resilient*, which means that if a service crashes the other can carry on working without hampering the execution of the other services if there are not many dependencies. The problem could be isolated then solved individually whereas for monolith applications if a problem occurs the whole application would stop working. Also since each microservice has a smaller codebase it can be deployed and built easily within a very short time [10]. A good thing about testing microservices is that each service can be tested individually to find bugs. The size of each microservice is small and so there are few features that need to be tested [11].

Although microservice architecture has many advantages, it also has several disadvantages. The communication between the services becomes complicated as more and more services are produced within an application resulting in a lot of dependency within the services. Each microservice can have its own resources and so managing the data transactions and processes can be a bit difficult depending on the size of the data. Since each service can use different technologies, therefore each service has its own logs making it difficult to keep track of the logs.

2.4 MongoDB

MongoDB is an open-source NoSQL database that is flexible and is easily scalable. Here *NoSQL* refers to non-structured query language which means that there are no tables or schemas for storing the data, like relational databases. Its architecture mainly consists of *collections* and *documents*. The data is stored as key-value pairs within a document in JSON-like format [12]. A document is similar to rows of a table in an SQL database but does not necessarily follow the strict data type or column of the tables. Even though multiple documents might have some common attributes, the data types of those attributes can be of different type and so giving it a much more flexible structure. Several documents together form a *collection*. A collection is similar to a table. There can be several collections within a single database. There is no need for schema within these collections.

The data in MongoDB is stored as BSON which is the binary version of JSON. MongoDB supports horizontal scaling with the help of *sharding* [13]. With the increase in the amount of data, a single server might not be enough to hold the data, what sharding does is, it adds more servers (MongoDB instances) in order to meet the increasing data, which is then distributed automatically across the multiple servers to balance the load.

If someone wants to look for a particular data then the data is only searched in the particular shard that holds the data, this reduces the number of operations and the amount of data that each shard holds. For example, if a database of 500 GB is divided across 4 shards then each shard would hold 125 GB of data [14].

MongoDB replicates the data across multiple servers and so making it immune to hardware failures, hence it can perform instant recovery if required. It also has features like *data aggregation* which combines data from multiple documents and performs a particular operation to return a single result. All the operations in MongoDB occur in real-time with no downtime. Any of the fields within a document can be indexed. Indexing allows efficient querying and skips the necessity to go through each and every document.

2.5 Apache Kafka

Apache Kafka is a distributed message queuing system that receives streams of data or messages from different source systems (producers) and delivers them to different target destination systems (consumers) [15]. It helps to avoid the coupling between the source and the destination systems [16]. Kafka is being widely used by companies such as LinkedIn, Netflix, Uber, and Airbnb. It can be widely used for different transactions of data.

Figure 1 illustrates the concept of Apache Kafka [17], how the data produced by the producers is assigned to a topic in a Kafka Broker, and later consumed by the consumers.

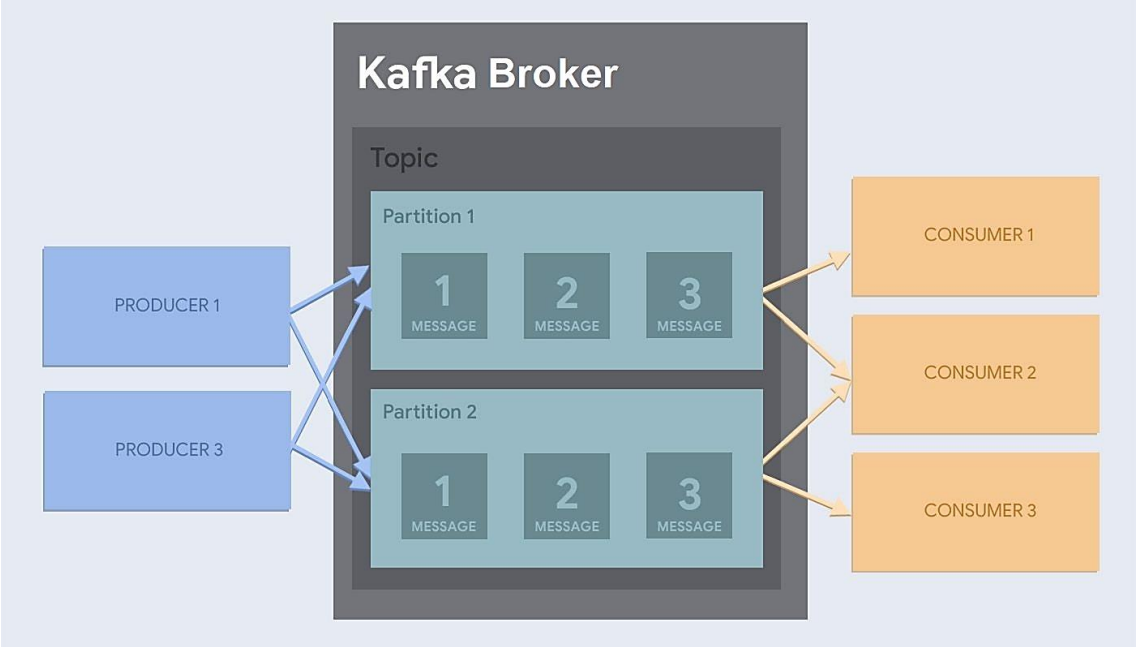

Figure 1: Concept of Apache Kafka

Kafka has concepts like *Kafka Clusters*, *Topics*, *Partitions*, *Offsets*, *Brokers*, *Producers*, *Consumers,* and *Zookeeper* [18] [19]. A *topic* is a stream of data that is similar to database tables. Each topic has a unique name. The topics can be split into *partitions* and the partitions are ordered. Partitions allow similar topics to be split across multiple *brokers* - each partition can be mounted on different machines to allow multiple consumers to read the topic in parallel. The number of partitions in each topic can be defined while creating the topic. An incremental id otherwise known as an *offset* is assigned to each message within a partition. Every offset has a specific meaning for a given partition. The data cannot be changed once it has been written to a partition. Data is kept only for a short period of one week, after that it gets removed.

Kafka cluster is made up of one or more *brokers*. A broker is a server for Kafka that hosts the topics. A broker can be identified by its id. Each broker includes partitions of certain topics. Connection to a single broker allows access to the entire cluster. Whenever a topic is created Kafka automatically distributes the topic partitions across all the brokers. For example, three brokers and a topic named Topic A is created with three partitions a, the partitions are automatically assigned to the brokers as following: Broker_1: Topic A Partition 0, Broker_2: Topic A Partition 1 and Broker_3: Topic A Partition 2. The topics should have a replication factor greater than 1 so that if a broker is down, the other active

brokers are able to serve the data. There is a concept of a *leader* within a partition, a single broker can become the leader for a specific partition at one time and it is only the leader which gets the partition data and serve the data. The remaining brokers sync data amongst themselves. Hence every partition only has a single leader and multiple replicas.

Producers write data into a particular topic. Producers know automatically which broker and which partition to write the data. If a broker fails then the producer recovers automatically. Producers can send the message along with a *key*. A key with the message ensures that every message goes to the same partition, but if the message is sent without a key then the message goes to the partitions in a round-robin manner as explained by the example above.

Consumers read data from a particular topic. Consumers have prior knowledge of which broker to read from. Consumers know how to recover in the event of broker failures. Inside each partition, the data is read in order. A consumer usually reads data as a group of consumers. Each consumer reads data from the exclusive partitions within a group.

Kafka needs *Zookeeper* to work [18]. Zookeeper manages all the brokers and maintains a broker list. Zookeeper assists in choosing the partition leader. Zookeeper notifies Kafka if there are changes like new topic creation, broker failure, deleted topics, active broker, etc. Zookeeper has a leader responsible for the writes and the remaining servers are the readers.

Benefits of Apache Kafka over other distributed messaging systems

Kafka's queue is persistent, as opposed to most messaging systems. It ensures that data sent to Kafka will be stored until a certain amount of time has passed or a size limit is reached. The message remains in the queue until one of the two things happens even after it is consumed. In Kafka, messages can be repeatedly replayed or consumed, an important feature that is useful in different scenarios [20].

Other messaging systems such as RabbitMQ stores messages until the application receiving the message connects to it and receives it. Once it gets the acknowledgment from the receiver that the message has been received or processed, it deletes the message from the queue. Therefore there is no option for replay or keep track once it is gone from the queue.

2.6 Spring Boot

Spring Boot is a lightweight, open-source framework for Java, used for creating micro-services and is maintained by a company called Pivotal. It provides Java developers to get started with an auto-configurable Spring production-ready application [21]. With it, developers can quickly get started without wasting time planning and configuring their Spring application. Spring Boot is designed on top of the Spring framework. One had to manually configure all the stuff in the Spring core framework, therefore several configuration files, such as XML descriptors were required [22].

Spring Boot, however, provides features like auto-configuration allowing developers to configure an application based on the dependencies list [23]. For example, when MySQL is listed as a dependency in the application, it will automatically configure the Spring application with the MySQL Connector. Spring Boot is *standalone*, which means that it can be simply run with a single command since it has a web server like Tomcat embedded in it. Spring Boot has support for annotations.

Spring Boot also imposes its opinions for which it is known to be *opinionated* [21]. This means that Spring Boot determines which configurations are used as default. It also determines which packages are needed to be installed for the dependencies mentioned in the application. For example, if "JPA" for Spring Boot is added, then the in-memory database, the hibernate entity manager, and a simple data source would be configured automatically. A spring boot application is easily deployable into containers and is ideal for the usage of microservices.

2.7 Docker

Docker is a containerization technology that bundles an application along with all of its dependencies together in the form of a *container* to ensure that the application runs seamlessly in any environment [24]. A container is a standardized unit of software that can be created and run easily in a particular environment. It is the runnable instance of an *image*, which is a read-only template of instructions for creating the container and includes information about the code, runtime, libraries, system tools, etc required by the application [25].

For example, a company develops a Java Application. A developer sets up an environment with an Apache Tomcat server installed in it. After the application is developed, it is tested by the tester. The tester sets up the environment again from scratch in order to test the application. When the application has been tested, it will be deployed to the production server. The production team also needs to set up the environment again.

Therefore, the environment has been set up three times in this procedure. This might result in some problems like loss of time and there might also be a mismatch in the server versions that have been set up. The problem can be solved using Docker, the developer creates a Docker image of the server and publishes it in *Docker Hub*, which is a service offered by Docker consisting of container images from different vendors and also allows developers to find and share own created container images. The same image can then be used by the tester and the system admin to deploy the application to production [24].

Another important feature provided by Docker is *integration*. The concept of integration is such that several instances of different tools, all running in the same container or running in different containers can communicate with each other by just running a few commands inside *Docker Compose*, which is a YAML file used for configuring multi-containerized applications [26]. The applications can also be scaled up by creating several containers inside the Docker Compose. The containers that are defined inside the Docker Compose can be easily started and run using a single command from a Docker enabled command-line interface (CLI).

3. CLOUBI AND THE PREVIOUS DATA ANALYTICS FRAMEWORK

This chapter describes Cloubi and its architecture and also gives a brief idea of the previous data analytics framework.

3.1 What is Cloubi?

Cloubi is a web application that is used for creating and distributing learning materials. It is being widely used by education publishers across Finland and Europe. *Learning material* in this context is a website that contains some interactive elements, questions, tasks, etc. Cloubi is sometimes compared to *LMS* or *CMS*. An LMS is a software that is used for providing courses or training programs allowing a teacher or a course instructor the privilege to create a course, enroll students, and grade the assignments. LMS, like Moodle [27] or Fronter [28], is a system for learning. It usually allows teachers and students to collaborate and use learning materials, like those produced with Cloubi. Even though Cloubi has some features in common with learning management systems, like reporting, grading, and commenting, Cloubi is not meant to replace an LMS. CMS is a system for collecting, organizing, and using content. Cloubi has its own CMS (the Library). Cloubi can use content from other content management systems as well. Cloubi has two versions, Cloubi 1 (C1) and Cloubi 2 (C2). Both of them are different products and do not share the same code base. C1 is still used by a few clients of Cloubi Ltd. whereas C2 is being used by most of the clients. Any relevant information can be fetched from C1 via C2.

The usage flow of Cloubi goes according to the following steps:

1. One or more user accounts are created for editors. Editors are the ones who create the learning material.
2. An editor uses Cloubi to create learning material. Multiple editors can edit the same material.
3. When a material is ready, it is published.
4. Students and teachers can then use the published materials from Cloubi.

3.2 Architecture of Cloubi

Cloubi can have one or more learning materials. A learning material in Cloubi consists of three parts as illustrated by the architecture of Cloubi in Figure 2:

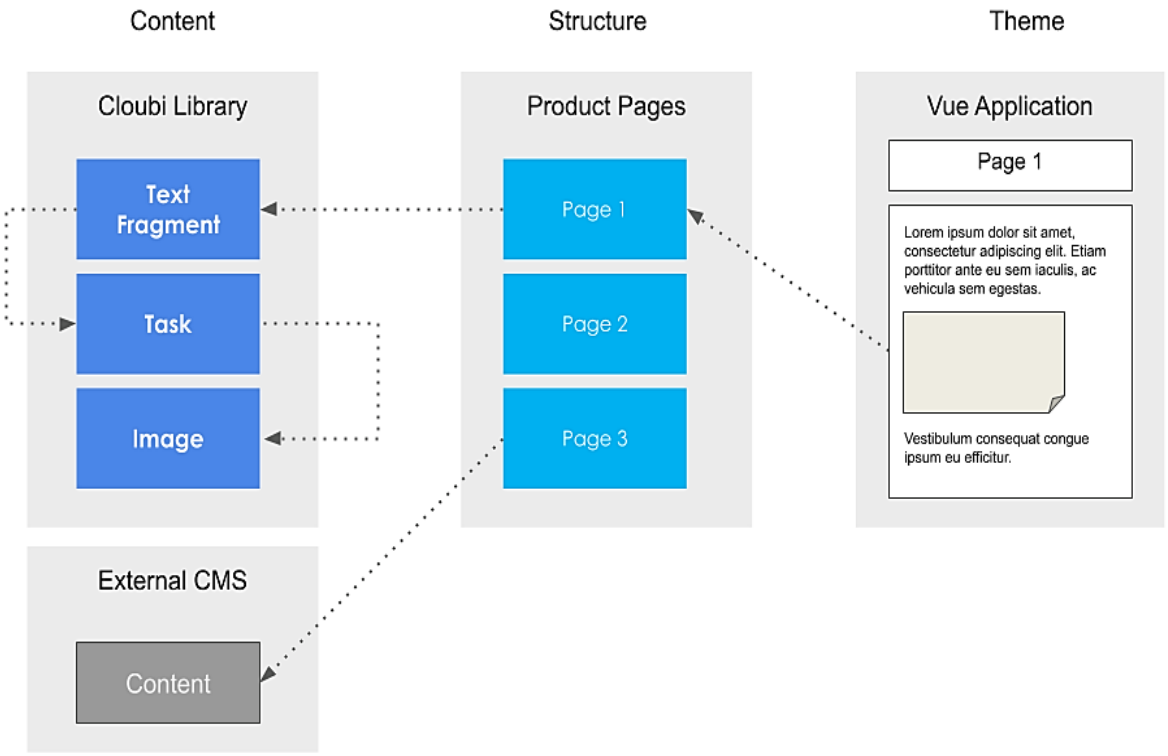

Figure 2: Architecture of Cloubi

1. **Structure**

   Structure is the central part of a learning material. It defines the pages of the material and how those pages are arranged into a hierarchy. This hierarchy is not fixed and it may be arbitrarily deep. Some materials might be divided into chapters and then into pages, while other material might be first divided into parts, parts into chapters, chapters into sections, and finally, sections into pages. Indeed, even a page can contain pages and subchapters.

2. **Content**

   Some pages in structure have content and some do not. For example, pages that allow students to navigate to subpages do not have any content. Or, to be more exact, the content of these navigational pages comes from the structure itself.

The pages can have text, images, and tasks which are known as *content.* These content come from some sort of content repository.

There are few content repositories available. The most prominent one is Cloubi's own library. The library can contain pages, texts, images, video files, audio files, tasks, and almost any other type of file. When something, like a task, is picked from the library and placed onto a page, that thing still remains in the library. The page only has a reference to the task. This way, the same content can be used in multiple pages and even in multiple learning materials.

3. **Theme**

   The *theme* is the styling applied to the web application. When the website loads, it has access to the structure and content. With these, the site can show whatever navigation it wants and display any page from the structure. If there are any additional styling, these will be applied to the pages.

3.3 Previous Data Analytics Framework

The previous data analytics framework was used for the reporting feature in Cloubi. Reports allowed teachers or students to check the students' performance within a material. The previous data analytics framework was part of a monolithic architecture. Therefore if any changes to the data analytics framework were required it would need to be done and implemented in the main code base of Cloubi. The previous data analytics framework is used for the reporting feature in Cloubi. A report is a separate page that is specific to the material and is usually found inside the material. The previous reporting required three attributes for it to work:

1. Student groups
2. Tasks
3. Student answers

The workflow of the previous reporting feature is illustrated in figure 3. The reporting feature needed to access three endpoints for it to function properly. The reporting feature works by getting the groups of the students or a single student via the group provider interface from C2's MongoDB. Usually, the group has information like the group id and the list of student ids, etc.

The task provider interface is then used to fetch the tasks that reside within a page or list of pages within a material. A task can be of two types, it can be a task from Cloubi 2 or Cloubi 1. Cloubi 2 tasks are fetched from the C2 library. Cloubi 1 tasks are fetched from Cloubi 1 itself. Depending on the type of the task, if it is a C2 task then the task details like the maximum achievable score from the task is fetched via the C2 library from C2's MySQL database. Similarly, if it is a C1 task then the fetching occurs via C1 from C1's MySQL database. Task has information like material id, page id, the title of the task, maximum score that can be achieved from the task, etc.

Finally, the students' answer data are fetched with the help of the student provider interface. The students' answer data includes the score achieved in a particular task, the progress of the student in that task as a percentage, time taken for completing the task, number of attempts and the time the task was started, etc. This data fetching also depends on whether the task is from Cloubi 2 or Cloubi 1. If it is from Cloubi 2 then the data is fetched via the C2 library from the C2 MongoDB. Likewise, if it is a C1 task then the data is fetched via C1 from C1's MySQL database.

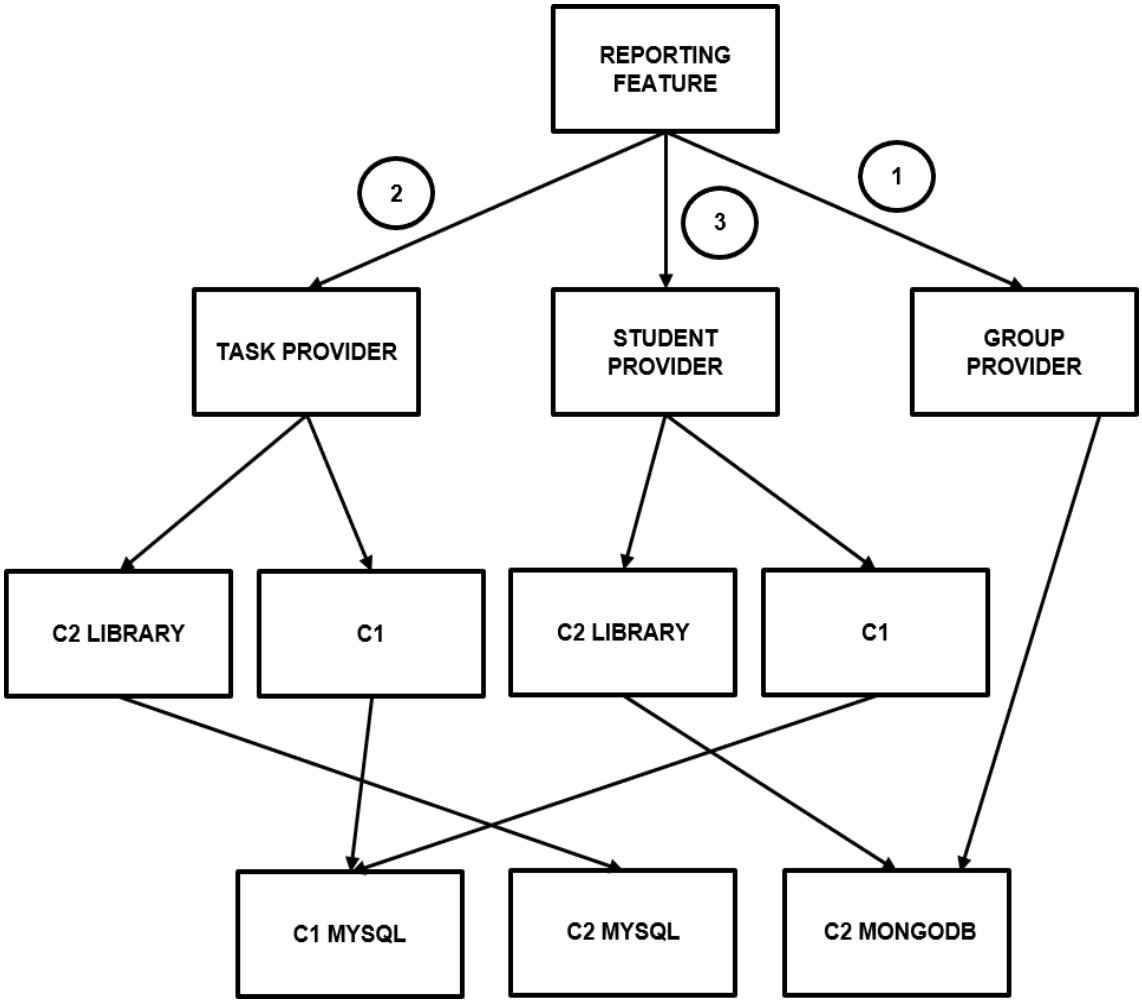

Figure 3: Concept of the previous data analytics framework

This approach results in a slow response, for example, if there are 1000 students and 1000 tasks this results in 1000x1000=1000,000 data about the progress of the students in the tasks which took time since so many queries had to be done in order for the data to be fetched from the databases used by Cloubi.

The implementation of the API was done with the providers mentioned in figure 3. The three endpoints of the API are:

1. **/o/reporting-api/student-groups**

   This endpoint is used to get information about the group of students. It works by making a GET request and fetches the information about the user, whether it is a teacher or a student, whenever the user id is passed as a query along with the URL. The portion ‘teacher’ returns true for a teacher or false for a student. If the user is a teacher then along with the teacher’s information in the ‘self’ portion, there is also the information about the group of students in the ‘groups’ portion. If the user is a student then this endpoint returns the information about that student in the ‘self’ portion.

2. **/o/reporting-api/query-tasks**

   This endpoint is used for getting the information about the tasks within a page found inside a material. The endpoint works by making a POST request and consists of a request body. The body of the request consists of a list of query parameters, which are material id, page id and if the tasks of the child pages were needed then it required passing true with the ‘childpages’ attribute or false if it was not required.

   The response would return the count of the total number of tasks in the ‘total’ portion and detailed information of the tasks in the ‘tasks’ portion.

3. **/o/reporting-api/query-student-tasks**

   This endpoint is used to get information about students’ answers to the tasks within a page found inside a material. It works by making a POST request and it also consists of a request body. The request body of this endpoint is similar to that of the ‘query-tasks’ with an additional ‘options’ portion that allows to sort, limit, and if the response is to be limited then which page should be given preference for the limited response.

   The response would return the count of the total number of answers to each task in the ‘total’ portion. It also includes the tasks’ information along with the id of the

students and detailed information about the answers provided by the students in the ‘tasks’ portion.

The details of the API can be found in Appendix A.

4. DESIGN AND IMPLEMENTATION

This chapter describes the design and implementation of the new data analytics framework used for the reporting feature of Cloubi along with the reasons for choosing the microservice architecture. This chapter also includes the integration tests carried out to check whether the microservice developed for the new data analytics framework was functioning correctly. At the end of the chapter, a comparison is made with related work.

4.1 Architectural Background

There were two reasons for choosing the microservice architecture for developing the new data analytics framework. One of the reasons was that the data requirements for the new reporting were not known in advance. The databases that Cloubi uses were not optimized for the changing data needs and it was not possible to modify the existing Cloubi databases to satisfy the needs since all the possible data requirements were not clear. Therefore using microservices having their own databases that could be optimized according to the data need was the optimum solution to this problem. For a new data need, a new microservice having its own database could be created.

Another reason for choosing the architecture was that the database used by the microservice gets all the data from Cloubi via a messaging system, so eventually it has all the data from Cloubi and so a microservice can be scaled up or duplicated as required for handling increasing network load.

4.2 Design

The new data analytics framework is designed following the microservice architecture as mentioned earlier. If any changes were required in the framework, it would not affect the main codebase of Cloubi. Figure 4 illustrates the design of the new reporting feature that has been implemented for the new data analytics framework of Cloubi for use by the teachers to see their students' overall progress throughout a material or it can also be used by the student themselves to check their own progress.

In the diagram, there are three separate services, which are Kafka, the application, and the MongoDB database. These three services together form the microservice that is used for the new data analytics framework. The application is made using Spring Boot. The MongoDB database is solely used for the purpose of this microservice that is to

serve the new reporting API created using the application and is expected to have an impact on the API's performance.

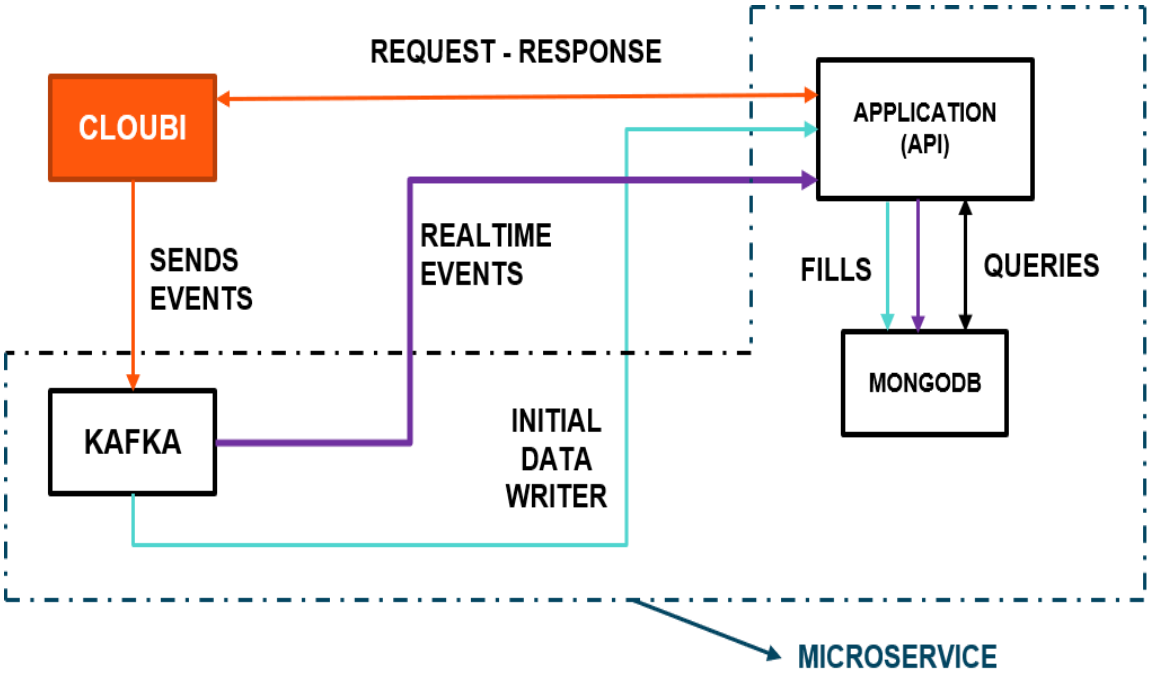

Figure 4: Concept of the new data analytics framework

The design illustrates that initially, Cloubi sends events to the application via Kafka. Upon receiving the data of the events, the database used by the application gets filled up with the data. Once there is data in the MongoDB used by the application, the API created can then respond as the request is made. Usually, the request is made by the frontend of the reporting page of a material in Cloubi. Details about the API is explained in the topic 4.4.

Kafka acts as a message-passing system connecting the application with Cloubi since Cloubi and the application are two separate disconnected systems. One of the reasons for the usage of Kafka is that it is being widely used in the industry and has many prebuilt libraries, it had plenty of tutorials and guidelines that were easy to follow. The main reason behind the usage of Kafka is that it can retain the data of the events for some time. Therefore, even if the application is down for any reason, there will not be any loss of the data. Whenever the application becomes up and running again it would sync the up-to-date data of events that have already occurred in Cloubi with the help of Kafka. Kafka can send the events to the application in two ways, one way is whenever an event occurs in real-time or at the very beginning as the services are started. Initially, the database of the application does not have any data, this data is filled up using the initial data writer. The details on Kafka integration into the microservice is explained in the next topic.

Whatever happens in Cloubi, from the creation of a material to the creation of a page and task and a student answering the tasks are sent as *events* to the microservice. Therefore events are all the occurrences happening in Cloubi. Events are the only way to let the microservice know that something has occurred in Cloubi and based on those events, the database used by the microservice is filled up with all the relevant data needed for the new reporting API to operate. The events that Cloubi sends to Kafka are structured in key-value pairs. The key consists of the event name and the value consist of a JSON object encoded in a string. The JSON object holds the various attributes of the event. The events along with their name and purpose are listed below:

1. **material**

   This event occurs whenever a new material is created in Cloubi or to inform a microservice that this material exists in Cloubi. The value of this event is described in the table below:

material	String	Unique identifier for the material.
title	String	Title of the material.
modified	long	Timestamp when the material was last modified.
rootPage	String	Unique identifier of the root page (or front page) of the material.
seqId (optional)	String	Unique identifier of the sequence that the event belongs to. If this event does not belong to any sequence, this attribute is missing. Sequences are started by the seqStart event and end with the seqEnd event.
time	long	Timestamp of the event as milliseconds since January 1, 1970, UTC. This value is the same format as returned by java.lang.System.currentTimeMillis() method in Java. This is the time when the event originally happened, not when the event is sent (the current time).

Table 1: Value of the event - material

2. page

This event occurs whenever a new page is created within a material in Cloubi or to inform a microservice that this page exists within a material in Cloubi. The value of this event is described in the table below:

material	String	Unique identifier for the material that this page belongs to.
page	String	Unique identifier for the page. The identifier is only unique within the material.
title	String	Title of the page.
modified	long	Timestamp when the page was last modified.
parent	String	Unique identifier of the page's parent page. If this page is a root page, then this attribute is missing.
children	Array of Strings	List of the page's child pages. Each item in the list is the child page's unique identifier. The pages are in the correct order in the list. If the page has no child pages then this list is empty.
breadcrump	Array of Strings	The page's *breadcrump*. The breadcrump is a list of page IDs, starting from the root page and ending with this page. It shows the page's all parent pages, not just the direct parent page.
tasks	Array of Strings	List of the tasks on this page. The list contains the unique identifiers of the tasks. The list is in no particular order. If the page has no tasks, then this list is empty.
seqId (optional)	String	Unique identifier of the sequence the event belongs to.
time	long	Timestamp of the event as milliseconds since January 1, 1970, UTC.

Table 2: Value of the event - page

3. **task**

   This event occurs whenever a new task is created within a page in Cloubi or to inform a microservice that this task exists within a page in Cloubi. The value of this event is described in the table below:

task	String	Unique identifier for the task.
title	String	Title of the task.
maxScore	int	The maximum score a student can get from the task. It can be zero.
tags	Array of Strings	List of keywords or tags the task has.
seqId (optional)	String	Unique identifier of the sequence the event belongs to.
time	long	Timestamp of the event as milliseconds since January 1, 1970, UTC.

Table 3: Value of the event - task

4. **answer**

   This event occurs whenever a student has answered a task or updated an answer. This also includes any interactions with a task that a student can perform and would result in having a different score or progress within the task, for example, starting a task and not finishing it. So this event does not actually require giving an answer to a question. The value of this event is described in the table below:

task	String	Unique identifier for the task that this answer is for.
user	String	Unique identifier for the user who gave the answer
score	int	User's current score from the task, as a result of answering the task. The value is between 0 to the maximum score that can be achieved from the task.
progress	int	User’s current progress in the task, as a result of answering the task. The value is between 0 to 100, in terms of percentage.

attempts	int	Number of times the user has attempted the task at this point. When the user first starts with the task, the value of the first attempt is 1. After each time the user restarts the task, this value increases by one.
seconds	int	Number of seconds the user has spent in the latest attempt.
secondsTotal	int	Total number of seconds the user has spent in the task, including all the attempts.
seqId (optional)	String	Unique identifier of the sequence that the event belongs to.
time	long	Timestamp of the event as milliseconds since January 1, 1970, UTC.

Table 4: Value of the event - answer

5. allMaterials

This event occurs whenever all the events are sent and it contains a list of all the current materials in Cloubi. Listeners can use this list to remove any old materials they might have. The value of this event is described in the table below:

materials	Array of Strings	List of the unique identifiers of all the available materials.
time	long	Timestamp of the event as milliseconds since January 1, 1970, UTC.

Table 5: Value of the event - allMaterials

6. materialDeleted

This event occurs whenever a material is deleted from Cloubi. The material along with all of its pages gets deleted. The value of this event is described in the table below:

material	String	Unique identifier for the material which was deleted.
time	long	Timestamp of this event as milliseconds since January 1, 1970, UTC.

Table 6: Value of the event - materialDeleted

7. **pageDeleted**

   This event occurs whenever a page within a material gets deleted from Cloubi. This also means that all of the deleted page's child pages are deleted. The value of this event is described in the table below:

page	String	Unique identifier for the page that was deleted.
material	String	Unique identifier for the material which contains the deleted page.
time	long	Timestamp of the event as milliseconds since January 1, 1970, UTC.

Table 7: Value of the event - pageDeleted

8. **seqStart**

   This event occurs whenever there is a new sequence of events that describes all content and data stored in Cloubi. Such sequence will always start with this event and end with the seqEnd event. The value of this event is described in the table below:

seqId	String	Unique random identifier for this sequence. All subsequent events that belong to this same sequence must have this same identifier.
time	long	Timestamp of this event as milliseconds since January 1, 1970, UTC.

Table 8: Value of the event – seqStart

9. **seqEnd**

   This event occurs to notify the end of a sequence of events started by the seqStart event. The value of this event is described in the table below:

seqId	String	Unique random identifier for this sequence.
time	long	Timestamp of this event as milliseconds since January 1, 1970, UTC.

Table 9: Value of the event - seqEnd

An example event looks like the following:

```
event_name: material, value: {"material":"5e562c5c8231b74159152714","modi-
fied":"1585937963769","time":"1588497859839","title":"Phys-
ics","rootPage":"5e562df88231b7415915271e","seqId":"e36ad577-7b33-be8b-4f55-
fb94253b7a7a-1588497859795"}
```

4.3 Kafka Integration

Cloubi contains the producer which publishes the events as they occur in Cloubi. In order for the microservice to receive the events, there was a need to implement the consumer within the microservice that would receive the events from Cloubi as it occurs. This was done with the help of two consumer classes named FullDataConsumer and RealTimeConsumer. Cloubi sends the events into two topics. The consumer classes in the application subscribes to both the topics. FullDataConsumer class handles the topic “FULL_DATA” and RealTimeConsumer class handles the topic “REALTIME”. FullData-Consumer serves the purpose of the initial data writer as illustrated by figure 4. Consumer class is the way of consuming the messages produced by Kafka. In Spring Boot a consumer class is implemented with the help of the “KafkaListener” annotation [29]. The purpose of the topics are described below:

1. **FULL_DATA**

   This topic is responsible for handling initial data of the events or whenever there is a need to send the full data of the events from Cloubi to the microservice. The events that are handled within this topic are seqStart, material, page, task, answer, seqEnd, and allMaterials. The microservice after receiving these events checks whether the seqId that it receives from the seqStart event is the same as the seqId in the events material, page, task, answer, and seqEnd. The purpose of this checking is to ensure that all these events are from the same batch and if they are not then the events will be discarded and the microservice will not do anything with them.

   When the microservice receives the material event, it fills the MongoDB used by the microservice with the information of the material. Similarly, when the microservice receives the page event, it fills in the database with the information about the page. The same happens with the events' task and answer. The event allMaterials is used to check whether the database used by the microservice has all the current materials that are present in Cloubi. If there are any old materials in the database it is deleted to make sure that the database used by the microservice is up to date with Cloubi.

2. **REALTIME**

   This topic is responsible for handling data of the events from Cloubi to the microservice as they occur in real-time. There are some events that can occur in both the topics, which include material, page, task, and answer. In this topic, there is no checking of whether all these events belong to the same batch or not.

   The microservice responds to these events as they occur and fills in the database with relevant information. In addition to these events, this topic also deals with the events materialDeleted and pageDeleted. Every time a material or a page gets deleted from Cloubi, the database used by the microservice also needs to delete the information of the material or the page that has been deleted.

4.4 MongoDB Repositories

The reason for choosing MongoDB over relational databases is that the structure of the data was too complicated to fit in any relational database and therefore MongoDB was used since it is the most widely used NoSQL database management system. The MongoDB repositories were designed in such a way so that the documents in each collection are organized in such a way that would result in efficient queries for the application. Three collections were used for keeping the information of the materials, pages, tasks, and answers of the students. As mentioned earlier, these collections were filled with the data from Cloubi with the help of the events.

The three collections and their structure is described below:

1. **Page Collection**

   Figure 5 illustrates the attributes of a document in the page collection. The _id is the unique identifier of a document in a collection in MongoDB, here page id is used as the unique identifier because every page residing in any material in Cloubi has unique IDs specific to the material. Then there is the material id and the list of task attributes.

   Each task attributes have the task id, the maximum score that can be achieved from the task, and a list of tags associated with the task. For example, if it was a math task, it can have tags like geometry, algebra, or calculus depending on the type of the task.

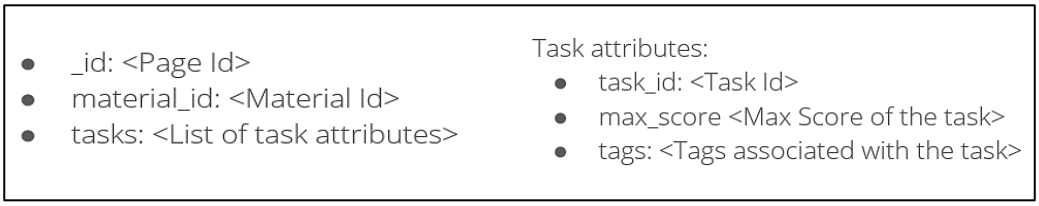

Figure 5: Attributes of a document in page collection

2. Task Collection

Figure 6 illustrates the attributes of a document in the task collection. The task id is used as the unique identifier of the document in this collection. The document also consists of a list of pages that has the task. The same task can exist on multiple pages in a Cloubi material.

Figure 6: Attributes of a document in task collection

3. StudentMaterial Collection

Figure 7 illustrates the attributes of a document in the studentMaterial collection. Here an auto-generated object id is used as the unique identifier of the document. The document also consists of the student id, material id, and a list of task attributes.

The tasks attributes consist of the task id, progress of the student in the task, the score achieved by the student in the task, the number of attempts made by the student in the task, the time taken in the latest attempt, the total time taken in all the attempts and the time of latest update in any tasks completed by the student.

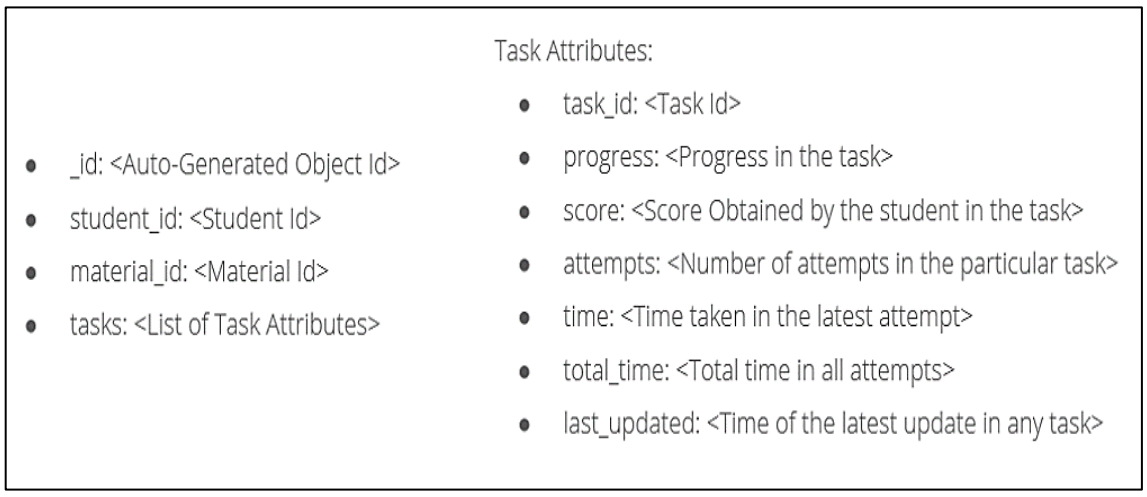

Figure 7: Attributes of a document in studentMaterial collection

4.5 API Implementation

The new API of the reporting feature was implemented to keep it as simple and as logical as possible. Unlike the previous reporting feature's API, this API is kept as minimalistic as possible. The previous reporting required three endpoints for the reporting feature to work but in this new version, it has been brought down to only one making it easy to access and respond accordingly. The API has been implemented with the help of a Controller Class in a Spring Boot Application that would fetch information from the MongoDB Repositories as necessary when requested with relevant information. A controller is a way of handling HTTP requests in the approach used by Spring for creating RESTful web services. In Spring Boot, a controller class is implemented with the help of the "RestController" annotation [29]. However, the API does not fully represent REST Architecture but it follows some of the concepts of REST and has some similarities to GraphQL. Therefore, it can be called a REST-like API.

The endpoint of the new API is "/o/analytics-framework/report-table". Initially, a request is made to the API using a POST request consisting of a request body containing a list of page ids and a list of student ids for which the response is generated. The page ids and the student ids are checked to see if they exist in the database and if the page ids and the student ids are present in the database then only a suitable response would be generated. The response has two portions, one is "pages" and the other is "students".

For n number of page ids, there is n number of responses in the "pages" portion. For n number of page ids and m number of student ids, there will be (m × n) number of responses in the "students" portion. For generating the response for the "pages" portion, the total number of tasks on a particular page and the sum of the maximum scores that can be achieved from the tasks on the page is calculated from the information of the tasks obtained for that page from the MongoDB repositories.

Similarly for generating the response for the "students" portion, the score obtained by a student in the tasks on that page is calculated, also the average progress of the student in the tasks on that page is calculated. Other calculated attributes include the sum of the time of the latest attempt made by the student in the tasks on the page, the sum of the total time spent in all the attempts made by the student in the tasks on the page, the total number of attempts in the tasks on the page, the average number of attempts in the tasks on the page, the time of latest update made by the student in any of the tasks on the page, the number of tasks started and completed by the student in the tasks on that page, the average progress of the tasks started and completed by the student in the tasks on that page, the score and the maximum score of the tasks started and completed

by the student in the tasks on that page. All of these attributes are calculated from the information of the student's answers obtained from the MongoDB repositories.

The details of the API is provided in Appendix B.

4.6 Dockerization

After the application was ready it was necessary to containerize the application. This is done with a Dockerfile that is used for building the Docker image of the application. Figure 8 illustrates the content of the Dockerfile used for the containerization of application.

```
FROM maven:3-jdk-8

WORKDIR /data

COPY src ./src
COPY pom.xml .

RUN mvn clean package

FROM openjdk:8-jre-alpine

EXPOSE 9000
ENTRYPOINT ["/usr/bin/java", "-jar", "/usr/share/spring-boot/analytics-report-table.jar"]

COPY --from=0 /data/target/report-table-0.0.1-SNAPSHOT.jar /usr/share/spring-boot/analytics-report-table.jar
```

Figure 8: Content of the Dockerfile used for containerization of application

Since it is a Spring Boot Application, it required Maven for the packaging of the application into a JAR (Java ARchive) file, which enables Java runtimes to deploy the application efficiently. The application also required Java 8 for it to run. The docker image has two build stages. The first build stage starts by pulling the official image of Maven using "FROM maven:3-jdk-8" command. The image was downloaded from the Docker Hub. Then by using "WORKDIR /data" command, the working directory of the image is specified to be "/data". Next, the src directory of the host is copied to the "src" folder in the image using "COPY src ./src" command. Also, the "pom.xml" which consists of all the dependencies required by the Spring Boot Application to function properly is copied from the host to the image's present location using "COPY pom.xml ." command. Then by using the "RUN mvn clean package" command, the existing target folder containing the jar file inside the image is deleted and rebuilt.

The second stage begins by downloading JDK 8 from Docker Hub's official image using the "FROM openjdk:8-jre-alpine" command. At runtime, the port 9000 is exposed by the container using the "EXPOSE 9000" command. This is the port that the application listens to, so it needs to be exposed for the application to be accessible from outside the

container. The application is run inside the container using the command “ENTRYPOINT ["/usr/bin/java", "-jar", "/usr/share/spring-boot/analytics-report-table.jar"]”. Finally, the jar file of the application is copied from the host to the image using the “COPY --from=0 /data/target/report-table-0.0.1-SNAPSHOT.jar /usr/share/spring-boot/analytics-report-table.jar” command copying the build artifact from the first stage to the second stage as indicated by “--from=0” [30].

This is the container that has been created for running the application. But for the micro-service to operate it requires other containers which are illustrated in figure 9.

```yaml
version: '3'

services:
  report-table-microservice:
    build: ./
    image: analytics-report-table
    ports:
      - '9000:9000'
    environment:
      - SPRING_DATA_MONGODB_HOST=mongodb
      - SPRING_KAFKA_CONSUMER_BOOTSTRAP-SERVERS=kafka:9093
    depends_on:
      - "mongodb"
      - "kafka"

  mongodb:
    image: mongo
    ports:
      - '27017'

  zookeeper:
    image: 'bitnami/zookeeper:3'
    ports:
      - '2181:2181'
    environment:
      - ALLOW_ANONYMOUS_LOGIN=yes

  kafka:
    image: 'bitnami/kafka:2'
    ports:
      - '9092:9092'
      - '9093'
    environment:
      - ALLOW_PLAINTEXT_LISTENER=yes
      - KAFKA_CFG_ZOOKEEPER_CONNECT=zookeeper:2181
      - KAFKA_CFG_ADVERTISED_LISTENERS=INSIDE://kafka:9093,OUTSIDE://10.0.2.2:9092
      - KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP=INSIDE:PLAINTEXT,OUTSIDE:PLAINTEXT
      - KAFKA_CFG_LISTENERS=INSIDE://0.0.0.0:9093,OUTSIDE://0.0.0.0:9092
      - KAFKA_CFG_INTER_BROKER_LISTENER_NAME=INSIDE
    depends_on:
      - zookeeper
```

Figure 9: Content of the Docker-Compose used for creating the microservice

The version of the docker-compose is defined as 3.0. There are three services that are required for the microservice to function which are the ‘report-table-microservice’, ‘mongodb’, and ‘kafka’. But since Apache Kafka does not work without Zookeeper it is included as a service named ‘zookeeper’ in this docker-compose file which results in the total number of services to be four. Initially, the ‘report-table-microservice’ service is built from the code and named as the ‘analytics-report-table’. The port 9000 of the container is mapped to the port 9000 of the host. The environment consists of the configuration required for the service to run. The ‘report-table-microservice’ requires ‘kafka’ and ‘mongodb’ services which are provided as the service dependencies.

Similarly, the image for MongoDB is pulled from Docker Hub and port 27017 is exposed for access by the other containers. Next, the image of Zookeeper is pulled from Docker Hub, the port of the host 2182 of the container is mapped to the port 2182 of the host. The environment consisting of the configuration for the ‘zookeeper’ service is set.

Finally, the image for Apache Kafka is pulled from the Docker Hub. The port 9092 of the container is mapped to 9092 of the host, also the port 9093 is exposed for access by other containers. The environment consisting of the configuration for running the service is provided and the only dependency required for the service ‘kafka’ to function is set to ‘zookeeper’ service. All the ports mapped here are the ports that the services listen to, so they need to be mapped in order for the services to be accessible from outside their containers. This is how the microservice that is being used for the new data analytics framework gets created.

One problem was mapping the Kafka listeners configuration to inside and outside the containers, this was solved by using the IP (Internet Protocol) address and the port numbers for communication outside the container.

Once the microservice was ready, a gateway was implemented inside Cloubi which is a service component that introduces the reporting microservice to Cloubi and handles the checking of the access rights.

4.7 Integration Tests

After the microservice was functioning it was necessary to perform *integration tests* to check that the microservice was generating the correct response whenever a request was made after receiving certain events. Integration testing is a method for checking how the individual modules perform when they are integrated together. For the case of the reporting microservice one module is for consuming the events from Cloubi and filling up the database, the other module is to extract the data from the database and return the

calculated attributes whenever a request is made. The microservice is tested using Cloubi's own microservice tester named 'ms-tester' that carries out the tests provided in a JSON file. The test cases were written while the frontend of the new reporting feature was not even started. A total of nineteen tests are carried out, which are:

1. **Empty page list in request**

   In this test case, the page list in the request is kept empty and checked whether the microservice returns an empty response in the "pages" and the "students" portion of the response. If it sends an empty response, then it passes the test.

2. **Empty student list in request**

   In this test case, the student list in the request is kept empty and checked whether the microservice returns an empty list in the "pages" and the "students" portion of the response. If it sends an empty response, then it passes the test.

3. **Empty page list and student list in request**

   In this test case, both the page list and the student list are kept empty in the request and checked whether the microservice returns an empty list in the "pages" and the "students" portion of the response. If it sends an empty response, then it passes the test.

4. **Single task in single page with single user**

   In this test case, the request is made with a single student and a single page. The page consists of a single task and answer to the task by the student. It is then checked whether the microservice returns the page's number of tasks and the maximum score that can be obtained from the page along with the student's answer to the task on the page. If the response is correct, then it passes the test.

5. **Single task in single page with multiple users**

   In this test case, the request is made with multiple students and a single page. The page consists of a single task and multiple answers from multiple students. It is then checked whether the microservice returns the page's number of tasks

and the maximum score that can be obtained from the page along with each student's answers to the task on the page. If the response is correct, then it passes the test.

6. **Single task in page hierarchy with single user**

   In this test case, the request is made with a single student and multiple pages. The multiple pages maintain a hierarchy with the child page consisting of a single task and a single answer by the student. In such a hierarchical structure the same task is also allocated to the parent page. It is then checked whether the micro-service returns all the pages' number of tasks and the maximum score that can be obtained from each page along with the student's answer to the task. If the response is correct, then it passes the test.

7. **Single task in page hierarchy with multiple users**

   In this test case, the request is made with multiple students and multiple pages. The multiple pages maintain a hierarchy with the child page consisting of a single task and multiple answers to the task by each student. In such a hierarchical structure the same task is allocated to the parent page. It is then checked whether the microservice returns all the pages' number of tasks and the maximum score that can be obtained from each page along with each student's answers to the task. If the response is correct, then it passes the test.

8. **Multiple tasks in single page with single answer by single user**

   In this test case, the request is made with a single student and a single page. The page consists of multiple tasks but a single answer to one of the tasks by the student. It is then checked whether the microservice returns the page's number of tasks and the maximum score that can be obtained from the page along with the student's answer to one of the tasks on the page. If the response is correct, then it passes the test.

9. **Multiple tasks in single page with single answer by multiple users**

   In this test case, the request is made with multiple students and a single page. The page consists of multiple tasks but a single answer to one of the tasks by

each student. It is then checked whether the microservice returns the page's number of tasks and the maximum score that can be obtained from the page along with each student's answers to one of the tasks on the page. If the response is correct, then it passes the test.

10. Multiple tasks in single page with multiple answers by single user

In this test case, the request is made with a single student and a single page. The page consists of multiple tasks and answers to those tasks by the student. It is then checked whether the microservice returns the page's number of tasks and the maximum score that can be obtained from the page along with the student's answer to all of the tasks on the page. If the response is correct, then it passes the test.

11. Multiple tasks in single page with multiple answers by multiple users

In this test case, the request is made with multiple students and a single page. The page consists of multiple tasks and answers to those tasks by each student. It is then checked whether the microservice returns the page's number of tasks and the maximum score that can be obtained from the page along with each student's answers to all of the tasks on the page. If the response is correct, then it passes the test.

12. Multiple tasks in page hierarchy with single answer by single user

In this test case, the request is made with a single student and multiple pages. The multiple pages maintain a hierarchy with the child page containing multiple tasks and a single answer to one of the tasks by the student. In such a hierarchical structure the same tasks are allocated to the parent page. It is then checked whether the microservice returns all the pages' number of tasks and the maximum score that can be obtained from each page along with the student's answer to one of the tasks. If the response is correct, then it passes the test.

13. Multiple tasks in page hierarchy with single answer by multiple users

In this test case, the request is made with multiple students and multiple pages. The multiple pages maintain a hierarchy with the child page containing multiple

tasks and multiple answers to one of the tasks by each student. In such a hierarchical structure the same tasks are allocated to the parent page. It is then checked whether the microservice returns all the pages' number of tasks and the maximum score that can be obtained from each page along with each student's answers to one of the tasks. If the response is correct, then it passes the test.

14. Multiple tasks in page hierarchy with multiple answers by single user

In this test case, the request is made with a single student and multiple pages. The multiple pages maintain a hierarchy with the child page containing multiple tasks and answers to the tasks by the student. In such a hierarchical structure the same tasks are allocated to the parent page. It is then checked whether the microservice returns all the pages' number of tasks and the maximum score that can be obtained from each page along with the student's answers to all of the tasks. If the response is correct, then it passes the test.

15. Multiple tasks in page hierarchy with multiple answers by multiple users

In this test case, the request is made with multiple students and multiple pages. The multiple pages maintain a hierarchy with the child page containing multiple tasks and answers to the tasks by each student. In such a hierarchical structure the same tasks are allocated to the parent page. It is then checked whether the microservice returns all the pages' number of tasks and the maximum score that can be obtained from each page along with each student's answers to all of the tasks. If the response is correct, then it passes the test.

16. Multiple tasks in multiple pages excluding rootPage and single user

In this test case, the request is made with a single student and multiple pages excluding the root page. The multiple pages contain multiple tasks and answers to the tasks by the student. It is then checked whether the microservice returns all the pages' number of tasks and the maximum score that can be obtained from each page along with the student's answers to the tasks on each page. If the response is correct, then it passes the test.

17. **Multiple tasks in multiple pages excluding rootPage and multiple users**

    In this test case, the request is made with multiple students and multiple pages excluding the root page. The multiple pages contain multiple tasks and answers to the tasks by each student. It is then checked whether the microservice returns all the pages' number of tasks and the maximum score that can be obtained from each page along with each student's answers to the tasks on each page. If the response is correct, then it passes the test.

18. **Multiple tasks in multiple pages including rootPage and single user**

    In this test case, the request is made with a single student and multiple pages including the root page. The multiple pages contain multiple tasks and answers to the tasks by the student. It is then checked whether the microservice returns all the pages' number of tasks and the maximum score that can be obtained from each page along with the student's answers to the tasks on each page. If the response is correct, then it passes the test.

19. **Multiple tasks in multiple pages including rootPage and multiple users**

    In this test case, the request is made with multiple students and multiple pages including the root page. The multiple pages contain multiple tasks and answers to the tasks by the student. It is then checked whether the microservice returns all the pages' number of tasks and the maximum score that can be obtained from each page along with each student's answers to the tasks on each page. If the response is correct, then it passes the test.

A sample test case is provided in Appendix-C.

4.8 Comparison with Related Work

There has been a lot of ongoing research on the use of microservice architecture over monolithic architecture. The use of microservice architecture has several benefits compared to that of monolithic architecture [31]. Microservices are independent and are relatively small in size, therefore allowing easy testing and debugging of the individual services. A change in one service does not necessarily affect the other services within the microservice and so the whole system does not need to be down for maintenance or code changes. Microservice based applications are easy to scale by adding other ser-

vices or duplicating them in order to reduce the network load. Each service can be developed with different technologies and interact with a common interface. Sadien et al. developed a mobile location-based *crowdsourcing* (MBLC) platform named QRowdsource using microservice architecture [32]. Crowdsourcing refers to a model in which mass people of indeterminate size would solve a complex problem. QRowdsource was a crowdsourcing application that was developed as the authors' first prototype using the microservice architecture, it allowed users to log in by scanning QR code from several locations of the authors' university campus and solve tasks to get credits. These credits could then be used in a coin dispenser set up on the campus.

The developers of QRowdsource used a load balancer named PM2 instead of Docker containers considering the small size of the deployment, for that reason they had to manually scale the services depending on the load. However, in this thesis, Docker containers have been used for microservice development which provides the advantage of connecting the services to each other easily.

QRowdsource was designed in a way that resulted in merging several services together to a single service for reducing the complexity of the application due to the amount of interconnectivity across the services. It is most suitable that each service within a microservice should serve a single functionality. In this thesis, the service within the microservice developed for the new data analytics framework mainly focuses on a single functionality of getting the students' data from the microservice's own database and returning the calculated data that is required for the reporting feature of Cloubi.

QRowdsource has a separate data service for handling all the data related transactions in the application, more like a monolithic approach. The primary reason for this was to avoid the problems caused by data migrations. Usually, in a microservice way of approach, each service has its own database. The microservice developed in this thesis for the new data analytics framework, has its own database and receives all the data from Kafka and is only used for the purpose of the new reporting API. Therefore, there is no need to worry about data loss since the microservice connects to Kafka to sync all the up-to-date data.

5. PERFORMANCE EVALUATION

This chapter gives a comparison between the performance for both the previous and the new data analytics framework.

5.1 Methodology

The performance of both the previous and the new data analytics framework API has been measured using *Glowroot* [33], which is an open-source application performance management tool for Java Applications used for getting the average API response time via the web browser.

As mentioned earlier, the previous data analytics framework required three endpoints, whereas the new data analytics framework requires only a single endpoint for it to operate. All the endpoints were tested with real Cloubi data from a small server consisting of two materials, one being relatively larger than the other. The larger material consisted of 49 pages and a total of approximately 960 tasks whereas the smaller material consisted of 8 pages and a total of approximately 213 tasks. For an easier understanding of the evaluation results, the larger material is called *M1* and the smaller material is called *M2*. Three student groups were used for this evaluation, the first student group (G1) had 15 students, the second group (G2) had 5 students and the third group (G3) had only 1 student.

Each endpoint was tested with both the larger material (M1) and the smaller material (M2) and the request was made with the three student groups. Each test was conducted several times and the average response time for each case was taken into account. The standard error (SE) for each endpoint was calculated to get the allowable error for each endpoint's response time using the formula mentioned below [34].

$$SE = \frac{Standard\ Deviation}{\sqrt{Number\ of\ Readings}} \quad (1)$$

The total response time and the total standard error for the endpoints for both the previous and the new data analytics framework were then compared.

The three endpoints required for the previous data analytics framework to work are:

- "/o/reporting-api/student-groups"
- "/o/reporting-api/query-tasks"

- “/o/reporting-api/query-student-tasks”

Whereas the only endpoint required by the new data analytics framework is:

- “/o/analytics-framework/report-table”

It is worth mentioning that the report page makes exactly the same number of calls to each endpoint for the previous data analytics framework. Therefore the response time from each endpoint is added to get the total response time for the previous reporting API.

For example, figure 10 illustrates the Glowroot API average response graph of the endpoint “/o/reporting-api/query-tasks” for case 1 (M1, G1) over a period of half an hour. The same request was made 5 times within an interval of approximately 5 mins with the larger material (M1) and a group of 15 students (G1). Out of these 5 readings, an average was then calculated.

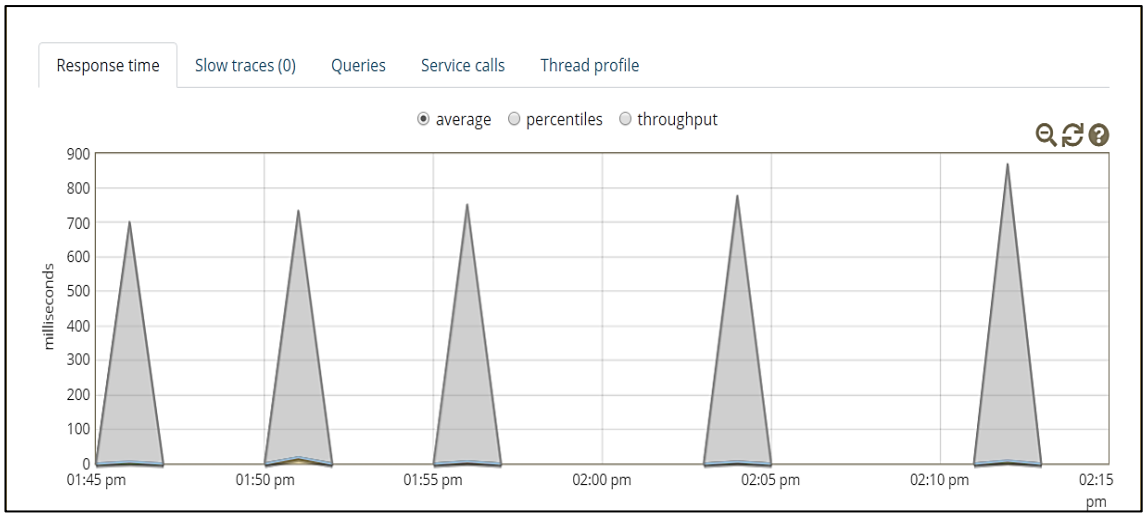

Figure 10: Average response time of the endpoint “/o/reporting-api/query-tasks” for case M1, G1

In some cases, this endpoint stops responding, this might have occurred due to the fact that Glowroot runs on the server, and what may have happened is that the browser used to make the requests cached the responses. In this case, the browser receives a response from the cache but no actual response is sent by the server, and therefore no response is visible in the graph.

This same procedure was followed for all the endpoints of the previous and the new data analytics framework and the comparisons will be elaborated by the following cases:

5.1.1 Case 1 (M1, G1) Comparison

This test was performed using the larger material and a group of 15 students.

Table 10 shows the total response time of the API used in the previous data analytics framework for the case M1, G1 represented in a tabular format based on the data from the API response graph. The standard error for each endpoint is calculated and summed up to find the total standard error of the endpoints. It can be seen that the endpoint used for fetching the student answers takes the most time followed by the endpoint used for querying the tasks present in the material and the endpoint used for fetching the information about the group of students' answers takes a negligible amount of time. The total amount of time taken for the three endpoints required for the previous reporting to work takes a total of 1756.3 milliseconds and the total standard error for the three endpoints is 49.8 milliseconds.

Previous Data Analytics Framework	**1**	**2**	**3**	**4**	**5**	**Avg.**	**SE**
/o/reporting-api/student-groups	5	6	5.5	6	9.1	6.3	±0.6
/o/reporting-api/query-tasks	700	730	750	780	880	768	±27.6
/o/reporting-api/query-student-tasks	980	1050	1000	980	900	982	±21.6
Total Time (ms)						**1756.3**	**±49.8**

Table 10: Total response time of the API used in the previous data analytics framework (M1, G1)

Table 11 shows the total response time of the API used in the new data analytics framework for the case M1, G1 represented in a tabular format based on the data from the API response graph. The standard error for the endpoint is calculated. It can be seen that the API used for fetching the information needed for the new reporting to work takes a total of 359 milliseconds and the total standard error for the endpoint is 34.6 milliseconds.

New Data Analytics Framework	**1**	**2**	**3**	**4**	**5**	**Avg.**	**SE**
/o/analytics-framework/report-table	450	325	325	250	445	359	±34.6
Total Time (ms)						**359**	**±34.6**

Table 11: Total response time of the API used in the new data analytics framework (M1, G1)

From Table 10 and Table 11, it can be seen that the new data analytics framework seems to be performing much better than the previous data analytics framework. The response time of the new data analytics framework is better than the previous data analytics framework by a time of 1397.3 milliseconds for the case M1, G1.

5.1.2 Case 2 (M1, G2) Comparison

This test was performed using the larger material and a group of 5 students.

Table 12 shows the total response time of the API used in the previous data analytics framework for the case M1, G2 represented in a tabular format based on the data from the API response graph. The standard error for each endpoint is calculated and summed up to find the total standard error of the endpoints. It can be seen that the endpoint used for fetching the student answers takes most of the time followed by the endpoint used for querying the tasks present in the material and the endpoint used for fetching the information about the group of students' answers takes a negligible amount of time. The total amount of time taken for the three endpoints required for the previous reporting to work takes a total of 1767.9 milliseconds and the total standard error for the three endpoints is 38.5 milliseconds.

Previous Data Analytics Framework	**1**	**2**	**3**	**4**	**5**	**6**	**Avg.**	**SE**
/o/reporting-api/student-groups	6	8.5	5	9	8	5	6.9	±0.7
/o/reporting-api/query-tasks	795	800	795	840	880	800	818	±12.9
/o/reporting-api/query-student-tasks	980	930	850	920	1050	930	943	±24.9
Total Time (ms)							**1767.9**	**±38.5**

Table 12: Total response time of the API used in the previous data analytics framework (M1, G2)

Table 13 shows the total response time of the API used in the new data analytics framework for the case M1, G2 represented in a tabular format based on the data from the API response graph. The standard error for the endpoint is calculated. It can be seen that the API used for fetching the information needed for the new reporting to work takes a total of 278 milliseconds and the total standard error for the endpoint is 16.1 milliseconds.

New Data Analytics Framework	1	2	3	4	5	6	Avg.	SE
/o/analytics-frame-work/report-table	260	250	280	240	360	280	278	±16.1
Total Time (ms)							**278**	**±16.1**

Table 13: Total response time of the API used in the new data analytics framework (M1, G2)

From Table 12 and Table 13, it can be seen that the new data analytics framework seems to be performing much better than the previous data analytics framework. The response time of the new data analytics framework is better than the previous data analytics framework by a time of 1489.9 milliseconds for the case M1, G2.

5.1.3 Case 3 (M1, G3) Comparison

This test was performed using the larger material and a group consisting of a single student.

Table 14 shows the total response time of the API used in the previous data analytics framework for the case M1, G3 represented in a tabular format based on the data from the API response graph. The standard error for each endpoint is calculated and summed up to find the total standard error of the endpoints. It can be seen that the endpoint used for fetching the student answers takes most of the time followed by the endpoint used for querying the tasks present in the material and the endpoint used for fetching the information about the student's answers takes a negligible amount of time. The total amount of time taken for the three endpoints required for the previous reporting to work takes a total of 671 milliseconds and the total standard error for the three endpoints is 46.4 milliseconds.

Previous Data Analytics Framework	1	2	3	4	5	Avg.	SE
/o/reporting-api/student-groups	9	12.5	5.3	8.5	6	8.3	±1.1
/o/reporting-api/query-tasks	360	380	260	270	250	304	±24.4
/o/reporting-api/query-student-tasks	330	450	345	350	320	359	±20.9
Total Time (ms)						**671**	**±46.4**

Table 14: Total response time of the API used in the previous data analytics framework (M1, G3)

Table 15 shows the total response time of the API used in the new data analytics framework for the case M1, G3 represented in a tabular format based on the data from the API response graph. The standard error for the endpoint is calculated. It can be seen that the API used for fetching the information needed for the new reporting to work takes a total of 239 milliseconds and the total standard error for the endpoint is 5.6 milliseconds.

New Data Analytics Framework	1	2	3	4	5	6	Avg.	SE
/o/analytics-framework/report-table	225	240	235	245	225	265	239	±5.6
Total Time (ms)							**239**	**±5.6**

Table 15: Total response time of the API used in the new data analytics framework (M1, G3)

From Table 14 and Table 15, it can be seen that the new data analytics framework seems to be performing better than the previous data analytics framework. The response time of the new data analytics framework is better than the previous data analytics framework by a time of 432 milliseconds for the case M1, G3.

5.1.4 Case 4 (M2, G1) Comparison

This test was performed using the smaller material and a group of 15 students.

Table 16 shows the total response time of the API used in the previous data analytics framework for the case M2, G1 represented in a tabular format based on the data from the API response graph. The standard error for each endpoint is calculated and summed up to find the total standard error of the endpoints. It can be seen that the endpoint used for fetching the student answers takes most of the time. After a certain time, the endpoint used for querying the tasks present in the material stops responding due to browser caching, and the endpoint used for fetching the information about the group of students' answers takes a negligible amount of time. The total amount of time taken for the three endpoints required for the previous reporting to work takes a total of 340.7 milliseconds and the total standard error for the three endpoints is 74 milliseconds.

Previous Data Analytics Framework	1	2	3	4	5	6	Avg.	SE
/o/reporting-api/student-groups	6.4	6.6	6.2	5.9	5.9	6.3	6.2	±0.1
/o/reporting-api/query-tasks	80	265	-	-	-	-	172.5	±65.4

/o/reporting-api/query-student-tasks	167	161	190	181	128	145	162	±8.5
Total Time (ms)							**340.7**	**±74**

Table 16: Total response time of the API used in the previous data analytics framework (M2, G1)

Table 17 shows the total response time of the API used in the new data analytics framework for the case M2, G1 represented in a tabular format based on the data from the API response graph. The standard error for the endpoint is calculated. It can be seen that the endpoint used for fetching the information needed for the new reporting to work takes a total of 95.8 milliseconds and the total standard error for the endpoint is 3 milliseconds.

New Data Analytics Framework	**1**	**2**	**3**	**4**	**5**	**6**	**Avg.**	**SE**
/o/analytics-framework/report-table	92	90	105	107	91	90	95.8	±3
Total Time (ms)							**95.8**	**±3**

Table 17: Total response time of the API used in the new data analytics framework (M2, G1)

From Table 16 and Table 17, it can be seen that the new data analytics framework seems to be performing better than the previous data analytics framework. The response time of the new data analytics framework is better than the previous data analytics framework by a time of 244.9 milliseconds for the case M2, G1.

5.1.5 Case 5 (M2, G2) Comparison

This test was performed using the smaller material and a group of 5 students.

Table 18 shows the total response time of the API used in the previous data analytics framework for the case M2, G2 represented in a tabular format based on the data from the API response graph. The standard error for each endpoint is calculated and summed up to find the total standard error of the endpoints. It can be seen that the endpoint used for fetching the student answers takes most of the time. The second endpoint used for querying the tasks present in the material does not have any response. The third endpoint used for fetching the information about the group of students' answers takes a negligible amount of time. Despite that, the total amount of time taken for the endpoints required for the previous reporting to work takes a total of 232.6 milliseconds. The total standard error for the three endpoints is 55 milliseconds.

Previous Data Analytics Framework	1	2	3	4	5	Avg.	SE
/o/reporting-api/student-groups	7.8	6	7	6.2	6	6.6	±0.3
/o/reporting-api/query-tasks	-	-	-	-	-	-	-
/o/reporting-api/query-student-tasks	455	205	220	102	145	226	±54.7
Total Time (ms)						**232.6**	**±55**

Table 18: Total response time of the API used in the previous data analytics framework (M2, G2)

Table 19 shows the total response time of the API used in the new data analytics framework for the case M2, G2 represented in a tabular format based on the data from the API response graph. The standard error for the endpoint is calculated. It can be seen that the endpoint used for fetching the information needed for the new reporting to work takes a total of 75.3 milliseconds and the total standard error for the endpoint is 7.8 milliseconds.

New Data Analytics Framework	1	2	3	4	5	6	Avg.	SE
/o/analytics-framework/report-table	70	55	75	90	54	108	75.3	±7.8
Total Time (ms)							**75.3**	**±7.8**

Table 19: Total response time of the API used in the new data analytics framework (M2, G2)

From Table 18 and Table 19, it can be seen that the new data analytics framework seems to be performing better than the previous data analytics framework. The response time of the new data analytics framework is better than the previous data analytics framework by a time of 157.3 milliseconds for the case M2, G2.

5.1.6 Case 6 (M2, G3) Comparison

This test was performed using the smaller material and a group of a single student.

Table 20 shows the total response time of the API used in the previous data analytics framework for the case M2, G3 represented in a tabular format based on the data from the API response graph. The standard error for each endpoint is calculated and summed up to find the total standard error of the endpoints. It can be seen that the endpoint used for fetching the student answers takes most of the time. The second endpoint used for

querying the tasks present in the material does not have any response. The third endpoint used for fetching the information about student's answers takes a negligible amount of time. Despite that, the total amount of time taken for the endpoints required for the previous reporting to work takes a total of 75.4 milliseconds. The total standard error for the three endpoints is 6.8 milliseconds.

Previous Data Analytics Framework	1	2	3	4	5	6	Avg.	SE
/o/reporting-api/student-groups	6.4	6	6.6	6.2	6	6.2	6.2	±0.1
/o/reporting-api/query-tasks	-	-	-	-	-	-	-	-
/o/reporting-api/query-student-tasks	105	65	68	59	59	59	69.2	±6.7
Total Time (ms)							**75.4**	**±6.8**

Table 20: Total response time of the API used in the previous data analytics framework (M2, G3)

Table 21 shows the total response time of the API used in the previous data analytics framework for the case M2, G3 represented in a tabular format based on the data from the API response graph. The standard error for the endpoint is calculated. It can be seen that the endpoint used for fetching the information needed for the new reporting to work takes a total of 45.6 milliseconds and the total standard error for the endpoint is 3.7 milliseconds.

New Data Analytics Framework	1	2	3	4	5	6	7	Avg.	SE
/o/analytics-framework/report-table	36	41	38	65	55	39	45	45.6	±3.7
Total Time (ms)								**45.6**	**±3.7**

Table 21: Total response time of the API used in the new data analytics framework (M2, G3)

From Table 20 and Table 21, it can be seen that the new data analytics framework seems to be performing better than the previous data analytics framework. The response time of the new data analytics framework is better than the previous data analytics framework by a time of 29.8 milliseconds for the case M2, G3.

5.2 Graph Representation of Case Comparisons

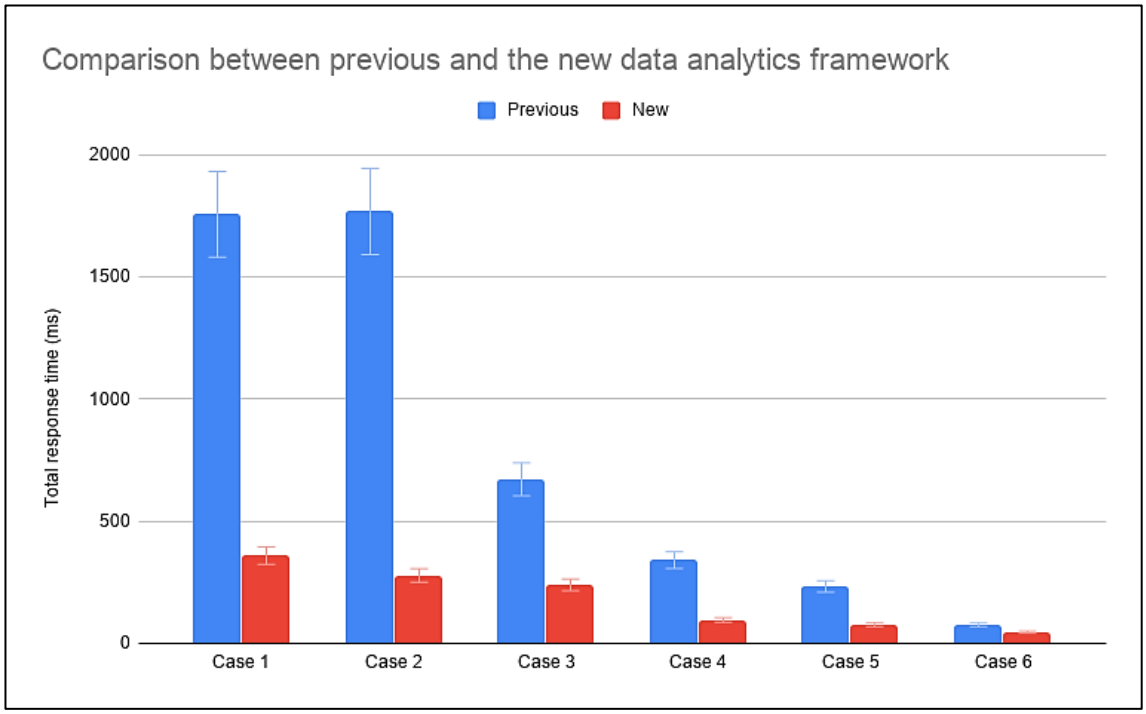

Figure 11: Graph representation of case comparisons

Figure 11 illustrates the graph representation of case comparisons for both the previous and the new data analytics framework consisting of the total response times and the total standard error.

5.3 Reliability of the Evaluation

The reliability of the evaluation results is determined by calculating the standard error. Figure 11 shows the total standard error lines for both the previous and the new data analytics framework for all the cases conducted and it has been calculated by dividing the standard deviation with the square root of the number of readings for each endpoint. Therefore, if the performance tests were conducted again with the same setup the allowable range of error would lie between the bounds represented by the error lines.

5.4 Validity of the Evaluation

Figure 11 shows that the total response time tends to decrease with the decrease in the number of students. However, this turned out to be false for the reduction in the number of students from case 1 (M1, G1) to case 2 (M2, G2) for the previous data analytics framework. The total response time taken for the endpoints of the previous data analytics framework for case 1 was 1756.3 milliseconds whereas the total response time taken for

the endpoints of the previous data analytics framework for case 2 was 1767.9 milliseconds.

It is difficult to determine the exact reason for this increase in response time but the total standard error should be considered in this case. There might be a couple of reasons that might have caused a delay in response. When the performance tests were conducted there might have been other users who were using the server. Another reason could be that the previous data analytics framework uses the databases that Cloubi uses and when the request was made, the databases had other loads from Cloubi that it had to deal with. The test was conducted on an AWS (Amazon Web Services) server that had some limitations due to the type of subscription, for that reason the server could have temporarily limited its capabilities resulting in the delay. Also, the server runs some background tasks periodically which might cause delays.

Nevertheless, the main comparison is between the total response time of the previous and the new data analytics framework which was not affected by this exception. It could be seen clearly from figure 11 that the new data analytics framework is working better than the previous data analytics framework in all the cases described above.

6. CONCLUSION

In conclusion, it can be said that the proof of concept for the design and development of the backend of the new data analytics framework is now complete and functional. This proof of concept was very important since it provided some valuable insights for Cloubi. The most important learning was that the users of the reporting feature did not know what they wanted in the new reporting, this resulted in the development of the backend assuming similar data requirements as the previous data analytics framework which was not true. It is not suggested to develop the backend without any front end user interface design. There was a need to sync the microservice with the latest data from Cloubi. For that reason, the Kafka topics were divided into 2 topics, one for the real-time data fetching and the other for the initial data writer.

Considering the design and implementation in chapter 4 of this thesis, it can be seen that an enormous amount of work had to be done to make the backend ready, connecting each of the modules to make them work and then writing integration tests to check whether the microservice was working as expected. The biggest challenge was to integrate Kafka to work along with the developed application since there were some difficulties with the Kafka configurations for the listeners inside and outside the containers while packing it as a microservice. While this was not much of a big problem but the connection of Cloubi with the microservice was a bit delayed which delayed the development and testing. However, the development of the proof of concept of the backend was done ahead of the deadline.

After the development of the backend was complete, it was necessary to conduct the performance testing of the new reporting API that had been developed with real Cloubi data. Therefore, a small server with data was set up to compare the performance of the new data analytics framework with that of the previous data analytics framework. From the results in chapter 5, it can be seen that the new framework has better performance in all the cases compared to the previous framework. Consequently, choosing the microservice architecture for the new data analytics framework had a positive impact on the performance since the microservice has a dedicated MongoDB database which serves solely for the purpose of the new reporting feature. Finally, the use of Kafka was most suitable for connecting Cloubi and the microservice because in case of disconnections the events would be stored and retrieved to synchronize the database used by the microservice with the latest information from Cloubi.

REFERENCES

[1] Cloubi, http://www.cloubi.com/ (accessed December 17, 2019).

[2] Kiran R. REST API - REST Constraints, https://www.youtube.com/watch?v=JYNYv8jJQTE (accessed April 18, 2020).

[3] Wu X, Zhu H. Formalization and analysis of the REST architecture from the process algebra perspective. *Future Generation Computer Systems* 2016; 56: 153–168.

[4] GraphQL, https://graphql.org/ (accessed April 19, 2020).

[5] Khachatryan G. What is GraphQL?, https://medium.com/devgorilla/what-is-graphql-f0902a959e4 (accessed April 19, 2020).

[6] Lewis J. What is GraphQL and GraphQL's advantages over REST Architecture?, https://medium.com/@jeffrey.allen.lewis/2018-beginners-guide-graphql-its-advantages-over-rest-architecture-972b0ef1dccb (accessed April 19, 2020).

[7] GraphQL vs REST, https://medium.com/@back4apps/graphql-vs-rest-62a3d6c2021d (accessed April 19, 2020).

[8] Fowler M. Microservices, https://martinfowler.com/articles/microservices.html (accessed February 2, 2020).

[9] Newman S. *Building Microservices*. 1st ed. O'Reilly Media, 2015.

[10] Advantages and Disadvantages of Microservices Architecture, https://cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback/ (accessed February 7, 2020).

[11] Ruponen E. *The Front-End Architectural Design And Implementation of a Modularized Web Portal*, http://urn.fi/URN:NBN:fi:tty-201905311819 (2019).

[12] What is MongoDB?, https://www.mongodb.com/what-is-mongodb (accessed February 1, 2020).

[13] MongoDB - Sharding, https://www.tutorialspoint.com/mongodb/mongodb_sharding.htm (accessed January 12, 2020).

[14] Replication and Sharding in MongoDB Tutorial, https://www.simplilearn.com/replication-and-sharding-mongodb-tutorial-video (accessed January 12, 2020).

[15] Noac'h P le, Costan A, Boug´e L. A Performance Evaluation of Apache Kafka in Support of Big Data Streaming Applications. 2017 IEEE International Conference on Big Data (BIGDATA), 2017.

[16] Apache Kafka, https://kafka.apache.org/intro (accessed January 12, 2020).

[17] Todd. Becoming Familiar with Apache Kafka and Message Queues, https://hackersandslackers.com/apache-kafka/ (accessed April 19, 2020).

[18] Kafka Architecture, http://cloudurable.com/blog/kafka-architecture/index.html (accessed January 12, 2020).

[19] Maarek S. Udemy Course - Apache Kafka Series - Learn Apache Kafka Series for Beginners v2, https://www.udemy.com/course/apache-kafka/ (accessed December 22, 2019).

[20] Johansson L. Which Service: RabbitMQ vs Apache Kafka, https://www.cloudkarafka.com/blog/2020-02-02-which-service-rabbitmq-vs-apache-kafka.html (accessed April 19, 2020).

[21] Mulders M. What is Spring Boot?, https://stackify.com/what-is-spring-boot/ (accessed March 31, 2020).

[22] Raffai Z. Spring Boot: The Most Notable Features You Should Know, https://dzone.com/articles/what-is-spring-boot (accessed April 7, 2020).

[23] Spring Boot, https://spring.io/projects/spring-boot (accessed March 31, 2020).

[24] Chaturvedi V. What is Docker and Docker Container? A Deep Dive into Docker!, https://www.edureka.co/blog/what-is-docker-container (accessed March 31, 2020).

[25] What is a Container?, https://www.docker.com/resources/what-container (accessed March 31, 2020).

[26] Overview of Docker Compose, Overview of Docker Compose (accessed March 31, 2020).

[27] Moodle, https://en.wikipedia.org/wiki/Moodle (accessed December 17, 2019).

[28] Fronter, https://itslearning.com/global/fronter/fronter-home/ (accessed December 17, 2019).

[29] Building a RESTful Web Service, https://spring.io/guides/gs/rest-service/ (accessed March 30, 2020).

[30] Use multi-stage builds, https://docs.docker.com/develop/develop-images/multistage-build/ (accessed April 6, 2020).

[31] Dragoni N, Giallorenzo S, Lluch-Lafuente A, et al. Microservices: yesterday, today, and tomorrow. 2016.

[32] Sadien I, Papangelis K, Fleming C, et al. Lessons Learned from Developing a Microservice Based Mobile Location-Based Crowdsourcing Platform, http://arxiv.org/abs/1909.03596 (2019).

[33] Glowroot - An open source APM tool for Java, https://medium.com/@gaurav.sharan4u/glowroot-an-open-source-apm-tool-for-java-7a570cc29378 (accessed April 20, 2020).

[34] Standard Error, https://en.wikipedia.org/wiki/Standard_error (accessed April 30, 2020).

APPENDIX-A: ENDPOINTS OF THE PREVIOUS REPORTING API

Endpoints of the previous reporting API:

1. GET /o/reporting-api/student-groups

Response:

```
{
   "teacher": true if the user is a teacher, false otherwise,
   "groups": [
         {
               "id": "the group's ID",
               "namespace": "the group's namespace (identifies which in-
                           terface the group data came from)",
               "fullId": "namespace:id",
               "title": "the name of the group",
               "schoolTitle": "the name of the school",
               "schoolId": "the identifier of the school",
               "students": [
                     {
                           "sessionId": "the user session ID of the stu-
                                       dent",
                           "name": "the student's name"
                     },...
               ]
         },...
   ],
   "self": {
         "id": "self_ + the user's session ID",
         "namespace": "(empty string)",
         "fullId": "self_ + the user's session ID",
         "title": "the user's name",
         "schoolTitle": "(empty string)",
         "schoolId": "(empty string)",
         "students": [
               {
                     "sessionId": "the user's session ID (the self group
                                 only ever has one student, who is the
                                 user making the request)",
                     "name": "the user's name"
               }
         ]
   }
}
```

2. POST /o/reporting-api/query-tasks

Request:

```
{
   "query": {
```

```
        "materialId": the long ID of the material whose tasks are being
                      queried,
        "pageId": "the String ID of the page whose tasks are being que-
                  ried",
        "childPages": a boolean indicating whether tasks on child pages
                      of the specified page should be included in the re-
                      sponse
   }
}
```

Response:

```
{
   "total": the number of tasks in the response,
   "tasks": [
         {
               "materialId": the ID of the material containing the task,
               "pageId": "the ID of the page containing the task",
               "taskId": "the library file ID of the task",
               "scoreMax": the maximum possible score for the task,
               "title": "the name of the task",
               "difficultyLevel": the difficulty of the task (from 1 to
                                  5),
               "keywords": [
                     "an array of metadata tags associated with the
                     task",...
               ],
               "gradeable": true if a teacher can grade the task,
               "breadcrump": [
                     "an array of page IDs indicating where the task is in
                     the material structure",...
               ]
         },...
   ]
}
```

3. POST /o/reporting-api/query-tasks

Request:

```
{
   "options": {
         "sort": "either null, byStarted or byStartedDesc, determines the
                 order of results (arbitrary order if null)".
         "limit": limits the number of returned results if present and
                  greater than 0,
         "page": which page of results to return when the result count is
                      limited
   },
   "query": {
         "materialId": the long ID of the material whose answers are being
                      queried,
         "pageId": "the String ID of the page whose answers are being
                   queried",
         "sessionId": the user session ID of the user whose answers are
                     being queried,
```

```
        "childPages": a boolean indicating whether tasks on child pages
                      of the specified page should be included in the re-
                      sponse
   }
}
```

Response:

```
{
   "total": the number of task answers in the response,
   "tasks": [
         {
               "materialId": the ID of the material containing the task,
               "pageId": "the ID of the page containing the task",
               "taskId": "the library file ID of the task",
               "scoreMax": the maximum possible score for the task,
               "title": "the name of the task",
               "difficultyLevel": the difficulty of the task (from 1 to
                                  5),
               "keywords": [
                     "an array of metadata tags associated with the
                     task",...
               ],
               "gradeable": true if a teacher can grade the task,
               "breadcrump": [
                     "an array of page IDs indicating where the task is in
                     the material structure",...
               ],
               "sessionId": the session ID of the student this answer be-
                            longs to,
               "attempts": [
                     {
                           "score": the score the student received in this
                                    attempt,
                           "progress": how far the student got in the task
                                       in this attempt on a scale from 0.0
                                       to 1.0,
                           "started": the timestamp when this attempt was
                                      started,
                           "time": how much time the student spent on this
                                   attempt (in seconds),
                           "teacherComment": "the comment the teacher left
                                             for the student when grading
                                             the task, if any",
                           "teacherCommentUnread": true if the student has
                                                   not yet seen the teach-
                                                   er's comment
                           "teacherCommentTimestamp": the time when the
                                                      teacher commented on
                                                      the answer, if any,
                           "teacherCommentorName": "the name of the
                                                   teacher who left the
                                                   comment, if any"
                     },...
               ]
         },...
   ]
}
```

APPENDIX-B: ENDPOINT OF THE NEW REPORTING API

Endpoint of the new reporting API:

POST /o/analytics-framework/report-table

Request:

```
{
      pages: list of page ids,
      students: list of student ids
}
```

Response:

```
{
      pages: [
                  {
                        maxScore: maximum score achievable from tasks in the
                                  page,
                        numTasks: number of tasks in the page
                  },...
            ]
      students: [
                  [
                    {
                        score: sum of scores the student achieved from the
                               tasks in the page,
                        progress: average of progress the student achieved
                                  from the tasks in the page,
                        time: sum of the number of seconds spent in the
                              latest attempt in the tasks in the page,
                        totalTime: sum of the number of seconds spent in all
                                   attempts in the tasks in the page,
                        attemptsTotal: total number of attempts in the tasks
                                       in the page,
                        attemptsAvg: average number of attempts in the tasks
                                     in the page,
                        lastUpdated: time of latest update in any of the task
                                     in the page,
                        tasksStarted: number of tasks started in the page,
                        progressOfStarted: average progress of started tasks
                                           in the page,
                        scoreOfStarted: sum of scores of started tasks in the
                                        page,
                        maxScoreOfStarted: sum of maximum scores of started
                                           tasks in the page,
                        tasksCompleted: number of tasks completed in the
                                        page,
                        progressOfCompleted: average progress of completed
                                             tasks in the page,
```

```
                        scoreOfCompleted: sum of scores of completed tasks in
                                          the page,
                        maxScoreOfCompleted: sum of maximum scores of com
                                             pleted tasks in the page
                    },...
                ],...
        ]
}
```

APPENDIX-C: SAMPLE TEST CASE FROM THE INTEGRATION TESTS

JSON Format of a test case from the integration tests:

```
{
	"name": "Multiple tasks in single page with multiple answers by single
		user",
	"events": [
			{
				"type": "material",
				"data": {
					"seqId": "1",
					"material": "${material}",
					"title": "Example material",
					"modified": 200,
					"rootPage": "${page}",
					"time": 100
				}
			},
			{
				"type": "page",
				"data": {
					"seqId": "1",
					"material": "${material}",
					"page": "${page}",
					"title": "Example page",
					"time": 101,
					"modified": 200,
					"parent": null,
					"children": [],
					"breadcrump": [
						"${page}"
					],
					"tasks": [
						"${task}",
						"${task2}"
					]
				}
			},
			{
				"type": "task",
				"data": {
					"seqId": "1",
					"task": "${task}",
					"title": "Example task",
					"maxScore": 3,
					"time": 101,
					"tags": []
				}
			},
			{
				"type": "task",
				"data": {
```

```
                        "seqId": "1",
                        "task": "${task2}",
                        "title": "Example task 2",
                        "maxScore": 3,
                        "time": 101,
                        "tags": []
                }
        },
        {
                "type": "answer",
                "data": {
                        "task": "${task}",
                        "user": "${user}",
                        "score": 3,
                        "progress": 100,
                        "attempts": 1,
                        "time": 1546300800000,
                        "seconds": 40,
                        "secondsTotal": 450
                }
        },
        {
                "type": "answer",
                "data": {
                        "task": "${task2}",
                        "user": "${user}",
                        "score": 3,
                        "progress": 100,
                        "attempts": 1,
                        "time": 1546300800000,
                        "seconds": 30,
                        "secondsTotal": 430
                }
        }
],
"request": {
        "pages": ["${page}"],
        "students": ["${user}"]
},
"response": {
        "pages": [
                {
                        "maxScore": 6,
                        "numTasks": 2
                }
        ],
        "students": [
        [
                {
                        "score": 6,
                        "progress": 100.0,
                        "time": 70,
                        "totalTime": 880,
                        "attemptsTotal": 2,
                        "attemptsAvg": 1,
                        "lastUpdated": "2019-01-
                                        01T00:00:00.000Z",
                        "tasksStarted": 2,
                        "progressOfStarted": 100.0,
```

```
                                        "scoreOfStarted": 6,
                                        "maxScoreOfStarted": 6,
                                        "tasksCompleted": 2,
                                        "progressOfCompleted": 100.0,
                                        "scoreOfCompleted": 6,
                                        "maxScoreOfCompleted": 6
                                    }
                                ]
                            ]
                    }
            }
```