Distributed Exams System

Distributed Systems Team Project Final Report

A report submitted in part fulfilment of the Distributed Systems module of Computer Science.

Student Name	*Student ID*
Colm Tang	16760705
Yassr Shaar	14328571
Shubham Narandekar	20200132
Rohan Shivakumar	15924441
Dongryul Jeong	15205766

School of Computer Science and Informatics

University College Dublin

05 January 2021

Abstract

This team project is aimed at developing a distributed exam office system where by the communication between systems is achieved through a RESTful approach and the results of classrooms are stored in a Database. Applications of this project would be in an educational institutes such as a school or university where there would typically be faculty staff such as Teachers, which are assigned to teach classrooms containing many students and a single Exam office which would handle class results and would store them in a Database.

Table of Contents

1 Introduction 4

2 Project's Specifications and Requirements 4

3 Methodology & System Architecture 5

4 Implementation Details 7

4.1 Implemented functionality 7

4.2 Setup and application use 11

4.3 Testing 12

5 Conclusion 13

6 References 14

1 Introduction

In a university setting, the communication between students, lecturers and the school exams office can be a very complicated affair. As such a distributed system would be best suited to tackle this. This project aims to do this by means of passing messages or requests with information between the systems as part of a RESTful, fault tolerant and scalable approach to the problem.

The application Distributed Exams System (DES) takes advantage of Spring boot to connect to a MongoDB database which will in turn store the results given back from the exam office onto the database through CRUD Operations.

These results that are stored on the database would be an overall average of results from each class based on the module and assessment complete. This average would be available at the request of any teacher to determine the class's overall performance.

2 Project's Specifications and Requirements

The Distributed Exams System project is written in Java while using Maven for build automation and dependency management. This is coupled with dependencies to spring boot for the REST configurations and JUnit for the unit tests.

DES requires a database to store the average grades for students within a classroom, for this we used a MongoDB database. The database would store the columns Classroom and average Result along with Student records.

In order to use Docker with our Maven project, certain configurations are required. A Docker-compile.yml file contained in the base repository would point towards all of the maven projects that are to be dockerised while also mentioning which projects are dependent on others and if any server hosts and ports are to be set.

Following this there are Dockerfiles in each subproject that signify which image is the project based on (in our case it would be the openjdk:8-jre-alpine) along with the copy target and destination and finally the commands to be executed such as matching request ports.

Certain plugins within the POM files were derived from practical labs such as org.codehaus.mojo and org.springframework.boot.

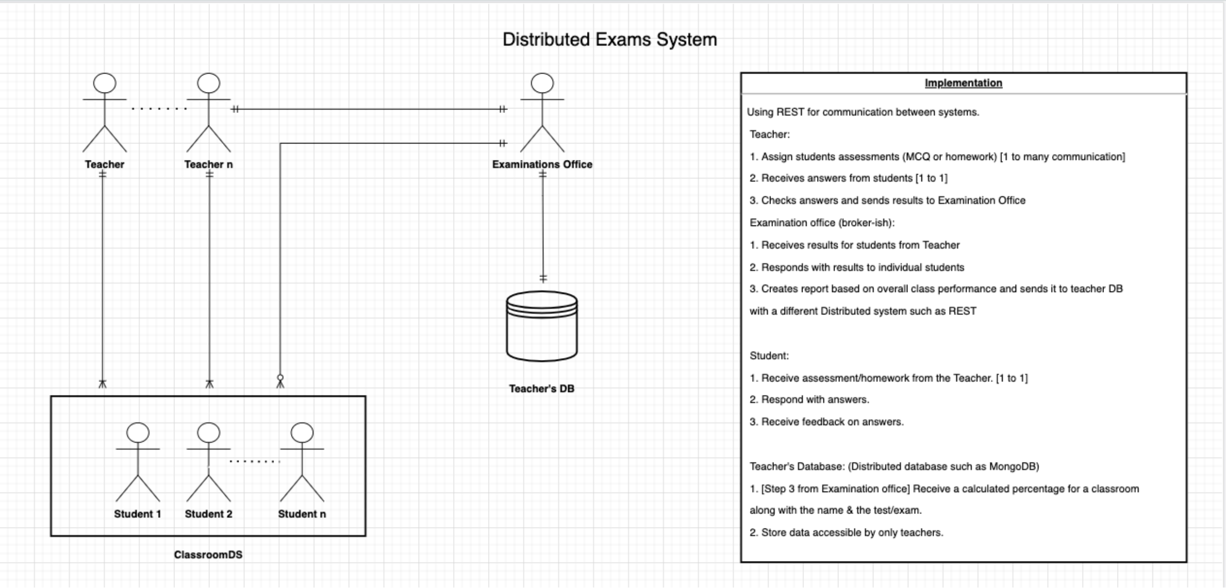
3 Methodology & System Architecture

DES is designed with the REST architecture to make HTTP calls between the Teachers, Classroom and the Exam Office. As REST is a lightweight web service these calls are made easily and can carry over a lot of data.

When deciding between different distributed methods to use on this project, REST was chosen as the primary architecture for the following reasons:

1. REST is the most popular and easy to use technology as it uses native web technologies such as HTTP. This made applications of this project more likely in a real-world scenario.
2. REST is very scalable as it allows for the adding of additional server nodes when needed behind a load balancer.
3. The nature of RESTful APIs allows them to be accessed from virtually any platform.

MongoDB was chosen as the distributed cloud solution for the project's database needs as it is fast and easy to integrate into a Maven environment but also because it allows for efficient scaling as needed by the project.



Implementation

Using REST for communication between systems.

Teacher:

1. Assign students assessments (MCQ or homework) [1 to many communication]
2. Receives answers from students [1 to 1]
3. Checks answers and sends results to Examination Office

Examination office (broker-ish):

1. Receives results for students from Teacher
2. Responds with results to individual students
3. Creates report based on overall class performance and sends it to teacher DB with a different Distributed system such as REST

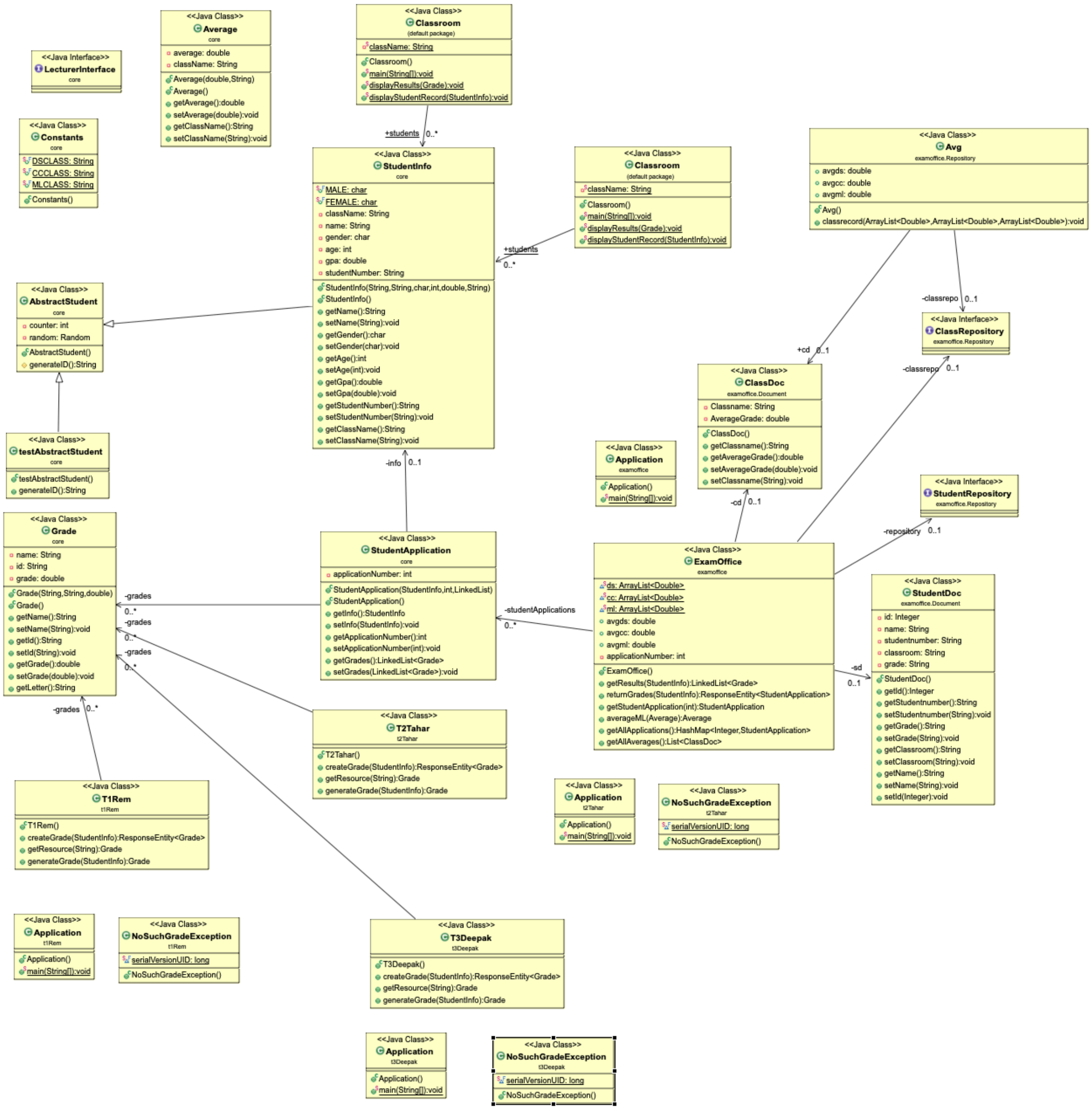
Student:

1. Receive assessment/homework from the Teacher. [1 to 1]
2. Respond with answers.
3. Receive feedback on answers.

Teacher's Database: (Distributed database such as MongoDB)

1. [Step 3 from Examination office] Receive a calculated percentage for a classroom along with the name & the test/exam.
2. Store data accessible by only teachers.

UML Diagram of classes relationship within DES project



4 Implementation Details

The implementation of DES makes use of a client server relationship whereas the Classrooms act like clients and the ExamOffice acts like the server which provides the communication channel to the Teachers such as T1Rem T2Tahar etc…

The Classrooms contain the list of students undergoing the class. A request is sent from classroom to examoffice, asking for a response from the Teachers regarding the grades of each student.

Once the examoffice receives a response from the Teachers, the MongoDB database is updated with the results for each student record. This is then sent to the classroom.

When the response is received from examoffice by classroom, a print of the student record with their results and grade is displayed.

Once all students in the classroom have received their grading, an average for the class is calculated and is posted to examoffice where it is stored in the Database.

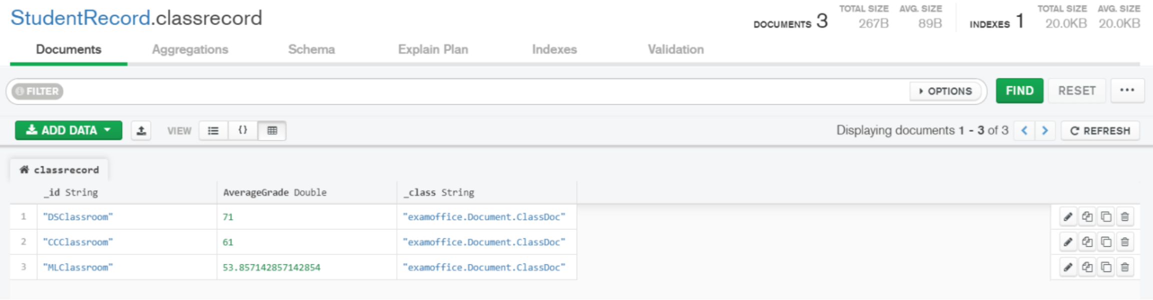
The Get methods can be tested through the use of Postman requests:

http://localhost:8080/applications/average
http://localhost:8080/applications/{application-number}

4.1 Implemented functionality

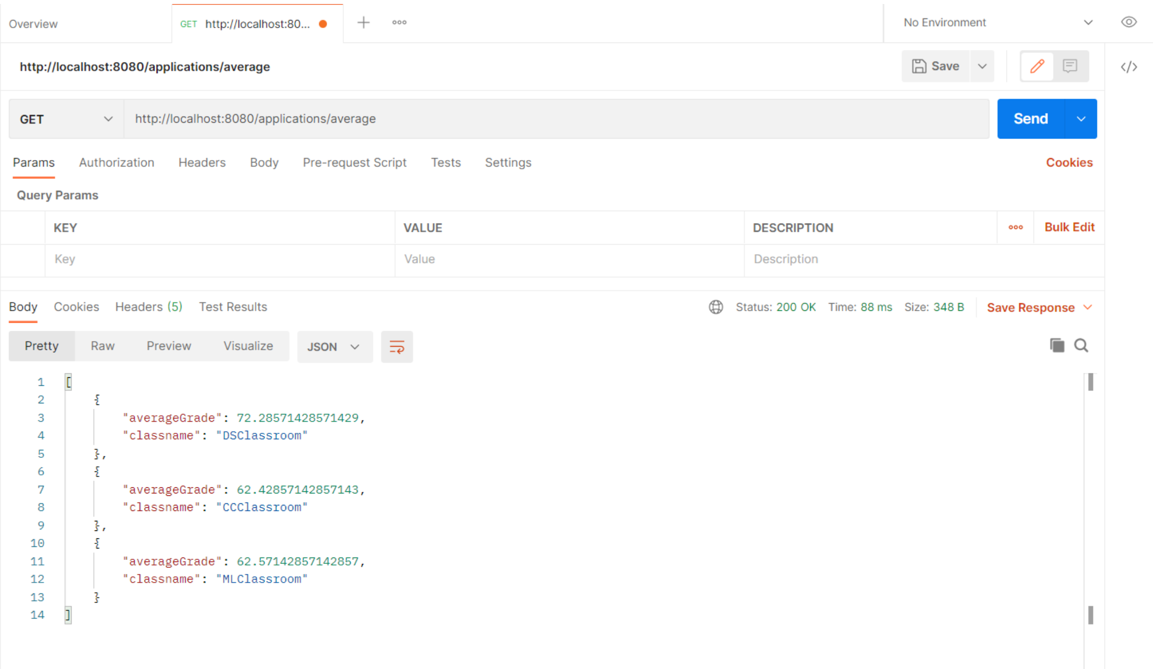
The functionality of Distributed Exams System is to pass communication appropriately between Teachers, classrooms and the Exam office then finally to process student results to find an average for a given classroom and store this in a Database.

The following screenshot displays the expected outcome of the DES project which is the classroom average results for all students stored within the MongoDB Database.



	_id String	AverageGrade Double	_class String
1	"DSClassroom"	71	"examoffice.Document.ClassDoc"
2	"CCClassroom"	61	"examoffice.Document.ClassDoc"
3	"MLClassroom"	53.857142857142854	"examoffice.Document.ClassDoc"

The following is the same as the above screenshot however displayed through a get request on Postman.

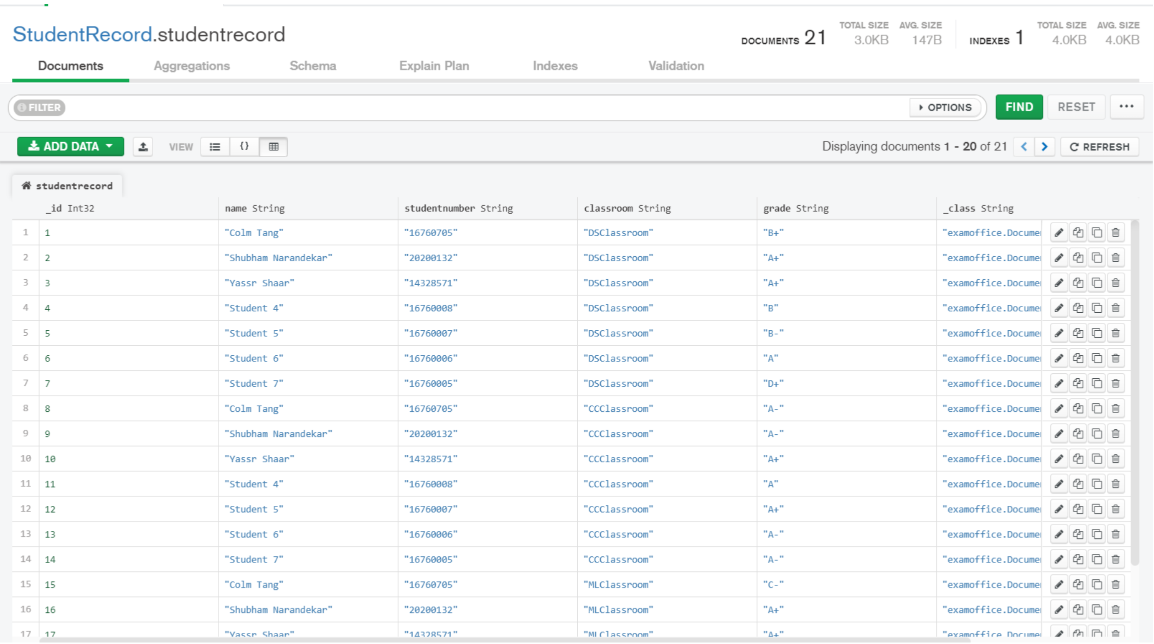


GET http://localhost:8080/applications/average

Status: 200 OK Time: 88 ms Size: 348 B

```
[
    {
        "averageGrade": 72.28571428571429,
        "classname": "DSClassroom"
    },
    {
        "averageGrade": 62.42857142857143,
        "classname": "CCClassroom"
    },
    {
        "averageGrade": 62.57142857142857,
        "classname": "MLClassroom"
    }
]
```

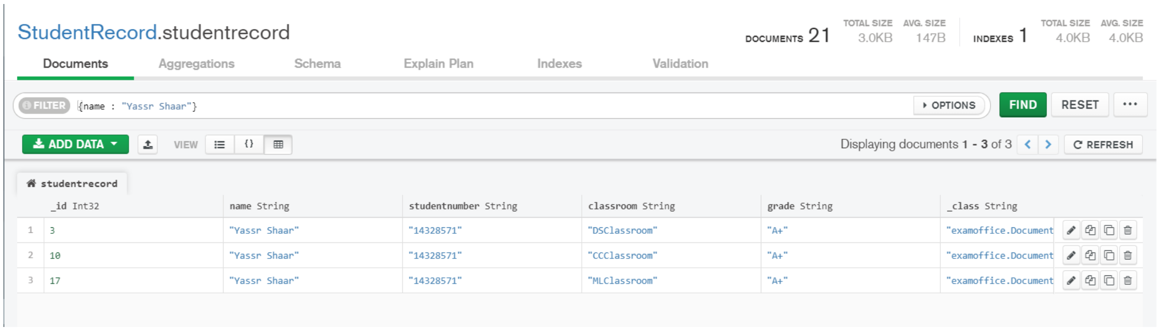
The MongoDB database shows the following results for the student records. It contains all of the students from the classrooms, with their respective student numbers and grades.



Displaying documents 1 - 20 of 21

	_id Int32	name String	studentnumber String	classroom String	grade String	_class String
1	1	"Colm Tang"	"16760705"	"DSClassroom"	"B+"	"examoffice.Docume
2	2	"Shubham Narandekar"	"20200132"	"DSClassroom"	"A+"	"examoffice.Docume
3	3	"Yassr Shaar"	"14328571"	"DSClassroom"	"A+"	"examoffice.Docume
4	4	"Student 4"	"16760008"	"DSClassroom"	"B"	"examoffice.Docume
5	5	"Student 5"	"16760007"	"DSClassroom"	"B-"	"examoffice.Docume
6	6	"Student 6"	"16760006"	"DSClassroom"	"A"	"examoffice.Docume
7	7	"Student 7"	"16760005"	"DSClassroom"	"D+"	"examoffice.Docume
8	8	"Colm Tang"	"16760705"	"CCClassroom"	"A-"	"examoffice.Docume
9	9	"Shubham Narandekar"	"20200132"	"CCClassroom"	"A-"	"examoffice.Docume
10	10	"Yassr Shaar"	"14328571"	"CCClassroom"	"A+"	"examoffice.Docume
11	11	"Student 4"	"16760008"	"CCClassroom"	"A"	"examoffice.Docume
12	12	"Student 5"	"16760007"	"CCClassroom"	"A+"	"examoffice.Docume
13	13	"Student 6"	"16760006"	"CCClassroom"	"A-"	"examoffice.Docume
14	14	"Student 7"	"16760005"	"CCClassroom"	"A-"	"examoffice.Docume
15	15	"Colm Tang"	"16760705"	"MLClassroom"	"C-"	"examoffice.Docume
16	16	"Shubham Narandekar"	"20200132"	"MLClassroom"	"A+"	"examoffice.Docume
17	17	"Yassr Shaar"	"14328571"	"MLClassroom"	"A+"	"examoffice.Docume

This list can be filtered to a specific student name to see their results for all classrooms/Modules.

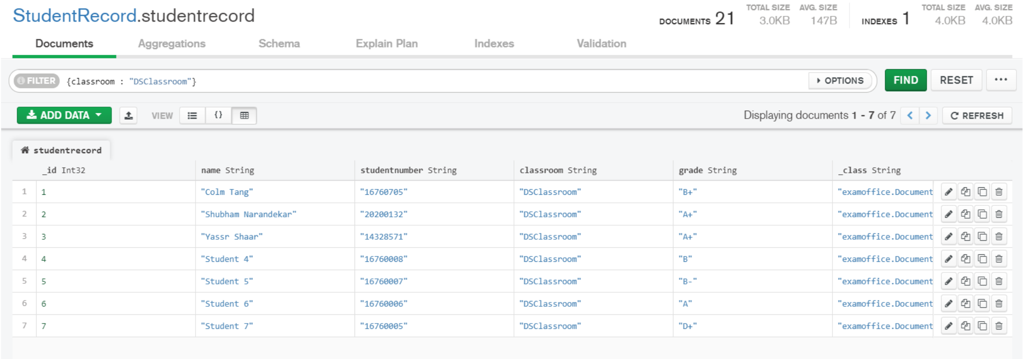


```
{name : "Yassr Shaar"}
```

Displaying documents 1 - 3 of 3

	_id Int32	name String	studentnumber String	classroom String	grade String	_class String
1	3	"Yassr Shaar"	"14328571"	"DSClassroom"	"A+"	"examoffice.Document
2	10	"Yassr Shaar"	"14328571"	"CCClassroom"	"A+"	"examoffice.Document
3	17	"Yassr Shaar"	"14328571"	"MLClassroom"	"A+"	"examoffice.Document

Similarly, a refinement can be done on the classroom to view all of the students results within that classroom.

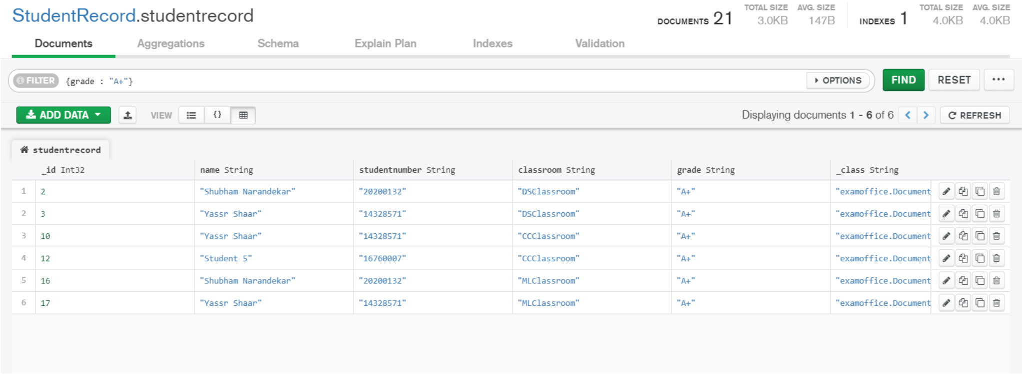


```
{classroom : "DSClassroom"}
```

Displaying documents 1 - 7 of 7

	_id Int32	name String	studentnumber String	classroom String	grade String	_class String
1	1	"Colm Tang"	"16760705"	"DSClassroom"	"B+"	"examoffice.Document
2	2	"Shubham Narandekar"	"20200132"	"DSClassroom"	"A+"	"examoffice.Document
3	3	"Yassr Shaar"	"14328571"	"DSClassroom"	"A+"	"examoffice.Document
4	4	"Student 4"	"16760008"	"DSClassroom"	"B"	"examoffice.Document
5	5	"Student 5"	"16760007"	"DSClassroom"	"B-"	"examoffice.Document
6	6	"Student 6"	"16760006"	"DSClassroom"	"A"	"examoffice.Document
7	7	"Student 7"	"16760005"	"DSClassroom"	"D+"	"examoffice.Document

Student record can also be filtered by the grades.

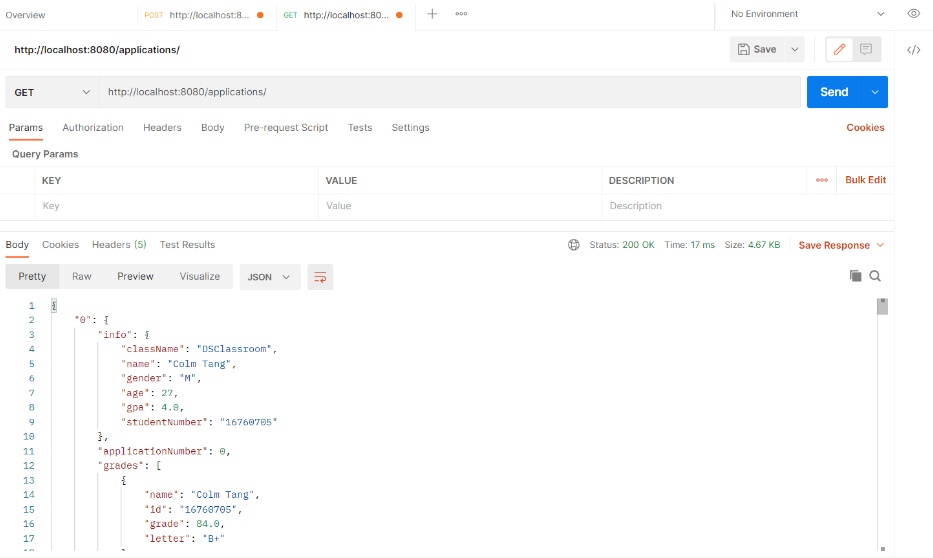


```
{grade : "A+"}
```

Displaying documents 1 - 6 of 6

	_id Int32	name String	studentnumber String	classroom String	grade String	_class String
1	2	"Shubham Narandekar"	"20200132"	"DSClassroom"	"A+"	"examoffice.Document
2	3	"Yassr Shaar"	"14328571"	"DSClassroom"	"A+"	"examoffice.Document
3	10	"Yassr Shaar"	"14328571"	"CCClassroom"	"A+"	"examoffice.Document
4	12	"Student 5"	"16760007"	"CCClassroom"	"A+"	"examoffice.Document
5	16	"Shubham Narandekar"	"20200132"	"MLClassroom"	"A+"	"examoffice.Document
6	17	"Yassr Shaar"	"14328571"	"MLClassroom"	"A+"	"examoffice.Document

Postman is used to test the REST GET methods for returning the student records.



http://localhost:8080/applications/

GET http://localhost:8080/applications/

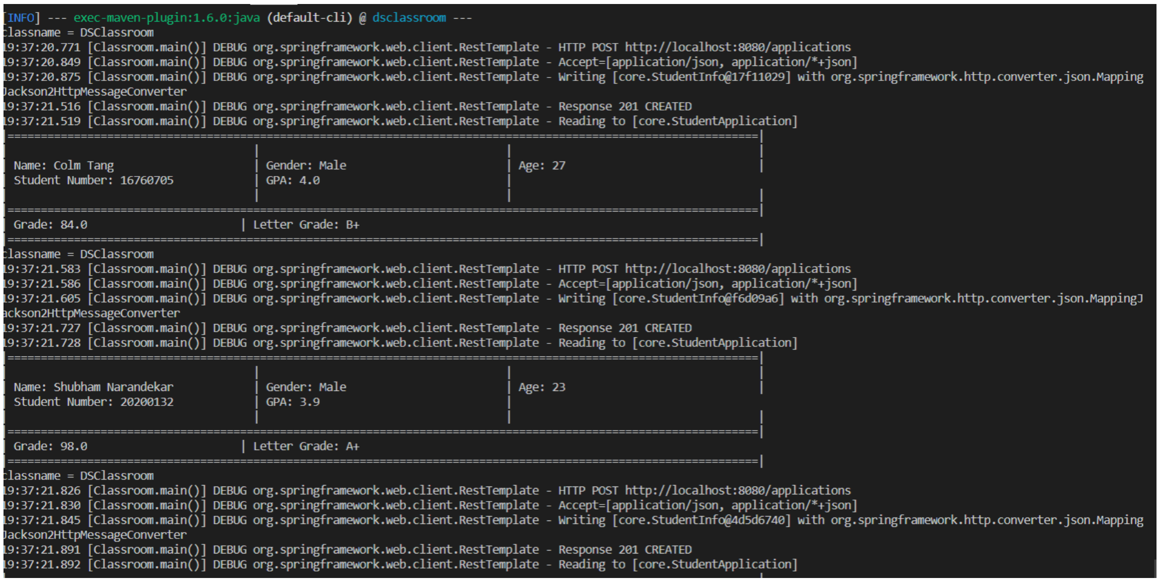
Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Status: 200 OK Time: 17 ms Size: 4.67 KB

```
{
    "0": {
        "info": {
            "className": "DSClassroom",
            "name": "Colm Tang",
            "gender": "M",
            "age": 27,
            "gpa": 4.0,
            "studentNumber": "16760705"
        },
        "applicationNumber": 0,
        "grades": [
            {
                "name": "Colm Tang",
                "id": "16760705",
                "grade": 84.0,
                "letter": "B+"
```

Expected terminal output upon running the project.



```
[INFO] --- exec-maven-plugin:1.6.0:java (default-cli) @ dsclassroom ---
classname = DSClassroom
19:37:20.771 [Classroom.main()] DEBUG org.springframework.web.client.RestTemplate - HTTP POST http://localhost:8080/applications
19:37:20.849 [Classroom.main()] DEBUG org.springframework.web.client.RestTemplate - Accept=[application/json, application/*+json]
19:37:20.875 [Classroom.main()] DEBUG org.springframework.web.client.RestTemplate - Writing [core.StudentInfo@17f11029] with org.springframework.http.converter.json.Mapping
Jackson2HttpMessageConverter
19:37:21.516 [Classroom.main()] DEBUG org.springframework.web.client.RestTemplate - Response 201 CREATED
19:37:21.519 [Classroom.main()] DEBUG org.springframework.web.client.RestTemplate - Reading to [core.StudentApplication]
|=====================================================================================================================|
|                                      |                                       |                                      |
| Name: Colm Tang                      | Gender: Male                          | Age: 27                              |
| Student Number: 16760705             | GPA: 4.0                              |
|                                      |                                       |                                      |
|=====================================================================================================================|
| Grade: 84.0                        | Letter Grade: B+
|=====================================================================================================================|
classname = DSClassroom
19:37:21.583 [Classroom.main()] DEBUG org.springframework.web.client.RestTemplate - HTTP POST http://localhost:8080/applications
19:37:21.586 [Classroom.main()] DEBUG org.springframework.web.client.RestTemplate - Accept=[application/json, application/*+json]
19:37:21.605 [Classroom.main()] DEBUG org.springframework.web.client.RestTemplate - Writing [core.StudentInfo@f6d09a6] with org.springframework.http.converter.json.MappingJ
ackson2HttpMessageConverter
19:37:21.727 [Classroom.main()] DEBUG org.springframework.web.client.RestTemplate - Response 201 CREATED
19:37:21.728 [Classroom.main()] DEBUG org.springframework.web.client.RestTemplate - Reading to [core.StudentApplication]
|=====================================================================================================================|
|                                      |                                       |                                      |
| Name: Shubham Narandekar             | Gender: Male                          | Age: 23                              |
| Student Number: 20200132             | GPA: 3.9                              |
|                                      |                                       |                                      |
|=====================================================================================================================|
| Grade: 98.0                        | Letter Grade: A+
|=====================================================================================================================|
classname = DSClassroom
19:37:21.826 [Classroom.main()] DEBUG org.springframework.web.client.RestTemplate - HTTP POST http://localhost:8080/applications
19:37:21.830 [Classroom.main()] DEBUG org.springframework.web.client.RestTemplate - Accept=[application/json, application/*+json]
19:37:21.845 [Classroom.main()] DEBUG org.springframework.web.client.RestTemplate - Writing [core.StudentInfo@4d5d6740] with org.springframework.http.converter.json.Mapping
Jackson2HttpMessageConverter
19:37:21.891 [Classroom.main()] DEBUG org.springframework.web.client.RestTemplate - Response 201 CREATED
19:37:21.892 [Classroom.main()] DEBUG org.springframework.web.client.RestTemplate - Reading to [core.StudentApplication]
```

4.2 Setup and application use

Starting MongoDB and connecting to localhost

1. Go to C:\Program Files\MongoDB\Server\4.4\bin (Where mongodb is installed).
2. Open cmd.
3. `mongod --dbpath="C:\Program Files\MongoDB\Server\4.4\data"` (Make sure to specify the path of your data folder).
4. Once you have started mongodb open MongoDBCompass which is a desktop application and then connect to your localhost.
5. After connecting you can access your databases.

Run the program

1. `mvn install`
2. `mvn exec:java -pl examoffice`
3. `mvn exec:java -pl t1Rem`
4. `mvn exec:java -pl t2Tahar`
5. `mvn exec:java -pl t3Deepak`
6. `mvn exec:java -pl dsclassroom`
7. `mvn exec:java -pl ccclassroom`
8. `mvn exec:java -pl mlclassroom`

How to check on Postman

To test `/applications` and `/applications/average` finally `/applications/{application-number}` need to fill a request body.

Install mongoDB image

1. Make sure docker is running
2. Enter cmd prompt and enter `docker pull mongo`

4.3 Testing

For this project our group implemented JUnitTesting on each java module. We used standard maven project structuring, creating a testing package that sits alongside the main code of each module.

We used three main types of JUnitTesting Asserts:
AssertNotNull to check that a class existed once called
AssertTrue to check that an object was of the correct class type
AssertEqual to check that a method returns correct output given the known input

By Module

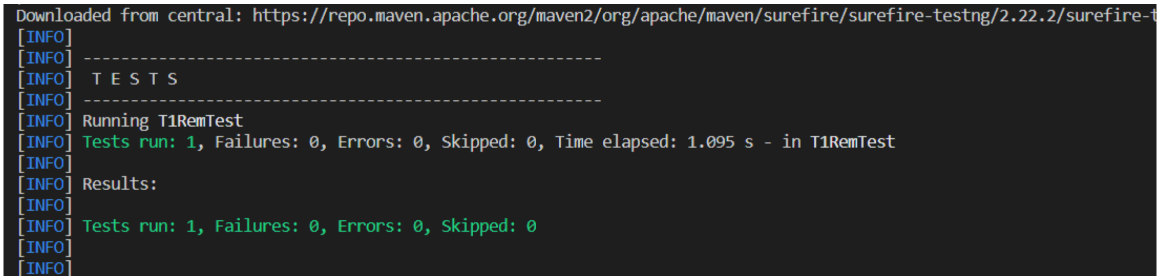
Core
GenerateIdTest: Checks that the ID generation method incrementation function works.
GetLetterTest: Checks that a given mark will return a given letter grade.
TestName: Checks the set and get name functions work.
TestID: Checks the set and get name functions work.
LecturerInterfaceTest: Ensures that the lecturer interface exists and is an interface type.

CCClassroom
TestName: Checks the set and get name functions work.
TestAge: Checks the set and get name functions work.
TestNotNull: Check that studentInfo is not null after setup.

DSClassroom
TestName: Checks the set and get name functions work.
TestAge: Checks the set and get name functions work.
TestNotNull: Check that studentInfo is not null after setup.

MLClassroom
TestName: Checks the set and get name functions work.
TestAge: Checks the set and get name functions work.
TestNotNull: Check that studentInfo is not null after setup.

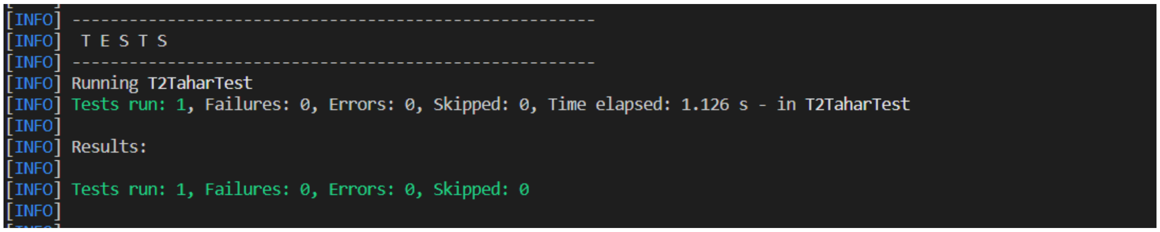
T1Rem
GradeTest: Checks grade is not null and is a type of grade class.
GenerateGradeTest: Checks that a given GPA returns a proper grade.



```
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-testng/2.22.2/surefire-t
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running T1RemTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.095 s - in T1RemTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
```

T2Tahar

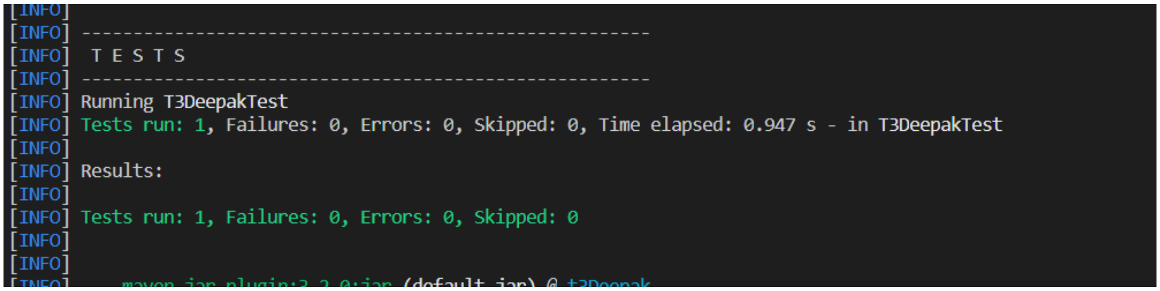
GradeTest: Checks grade is not null and is a type of grade class.
GenerateGradeTest: Checks that a given GPA returns a proper grade.



```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running T2TaharTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.126 s - in T2TaharTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
```

T3Deepak

GradeTest: Checks grade is not null and is a type of grade class.
GenerateGradeTest: Checks that a given GPA returns a proper grade.



```
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running T3DeepakTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.947 s - in T3DeepakTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ t3Deepak
```

5 Conclusion

The aim of this project was to create a distributed system that communicated through REST and to pass a manipulated outcome which would get stored in a MongoDB database. This was achieved throughout the implementation of Distributed Exams System application.

The passing of data between systems and storing of manipulated results to get the average was a tedious task which at first had many falling points. However, once we realised what factors influenced the REST communication and what the requirements needed by MongoDB are for recognising the connection it was a much more straight forward task.

The final hurdle was dockerising the project as it would require taking into consideration the MongoDB database. However despite the use of a MongoDB image and configuration of the ports we were unable to achieve dockerisation of the project.

The use of REST in this project with the combination of MongoDB has taught us valuable real life applications of technology that would be deemed as industry standard. This will go on to benefit us all greatly in our future careers.

6 References

6.1.1.1 *Mastering REST Architecture — REST Architecture Details* (2019). Available at: https://ahmetozlu93.medium.com/mastering-rest-architecture-rest-architecture-details-e47ec659f6bc (Accessed: 4 January 2021).

6.1.1.2 *MongoDB vs MySQL* (2021). Available at: https://www.mongodb.com/compare/mongodb-mysql#:~:text=MySQL%20is%20a%20relational%20database,(SQL)%20for%20database%20access.&text=MongoDB%20is%20a%20NoSQL%20database,data%20as%20JSON%2Dlike%20documents. (Accessed: 4 January 2021).

6.1.1.3 *Spring Boot With MongoDB CRUD Example | Java Techie* (2021) *Youtube.com*. Available at: https://www.youtube.com/watch?v=k5PeywcbVYc&ab_channel=JavaTechie (Accessed: 4 January 2021).

6.1.1.4 *Deploying Spring Boot and MongoDB as Containers Using Docker and Docker Compose* (2021) *Youtube.com*. Available at: https://www.youtube.com/watch?v=FA3VdlBo0nA&ab_channel=TechnoTownTechie (Accessed: 4 January 2021).

6.1.1.5 *Control startup and shutdown order in Compose* (2020). Available at: https://docs.docker.com/compose/startup-order/ (Accessed: 4 January 2021).