COMP 7033 CW2 REPORT

Ahmet Efe AKKAYA

19277883

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. GUI and API Design

1.1. Diagrams of Graphical User Interface

1.1.1. Old and New Design

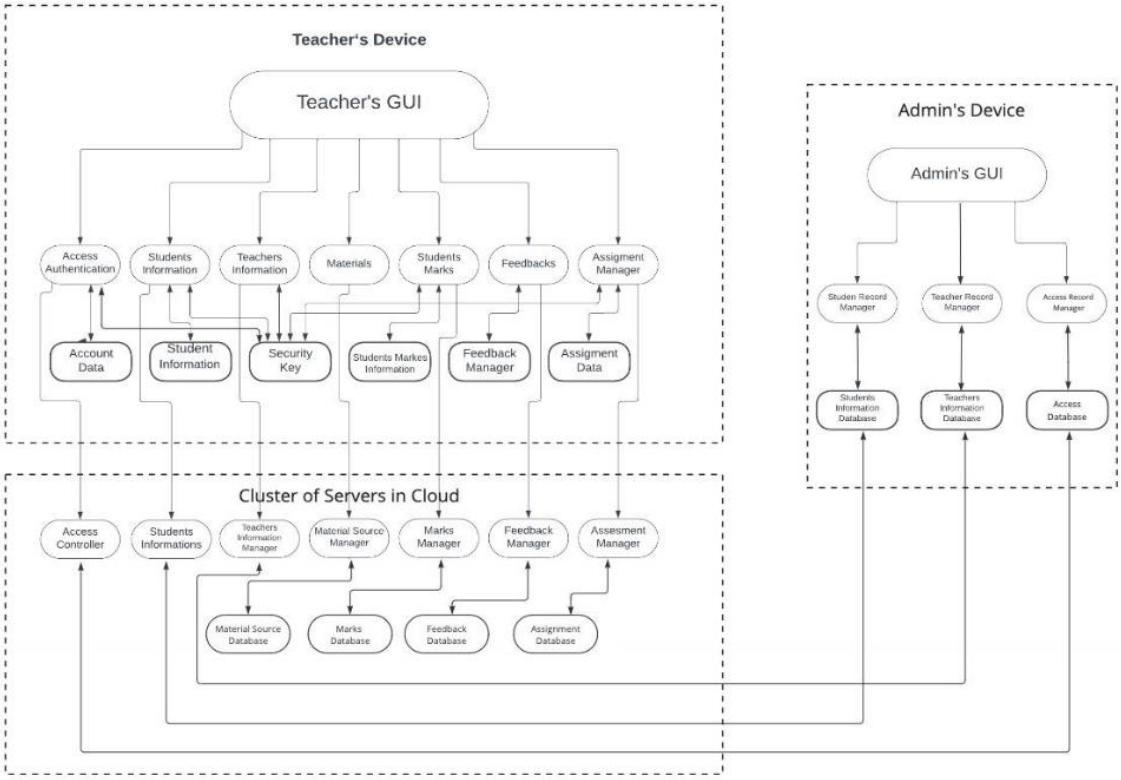


Figure 1.1 Old Design

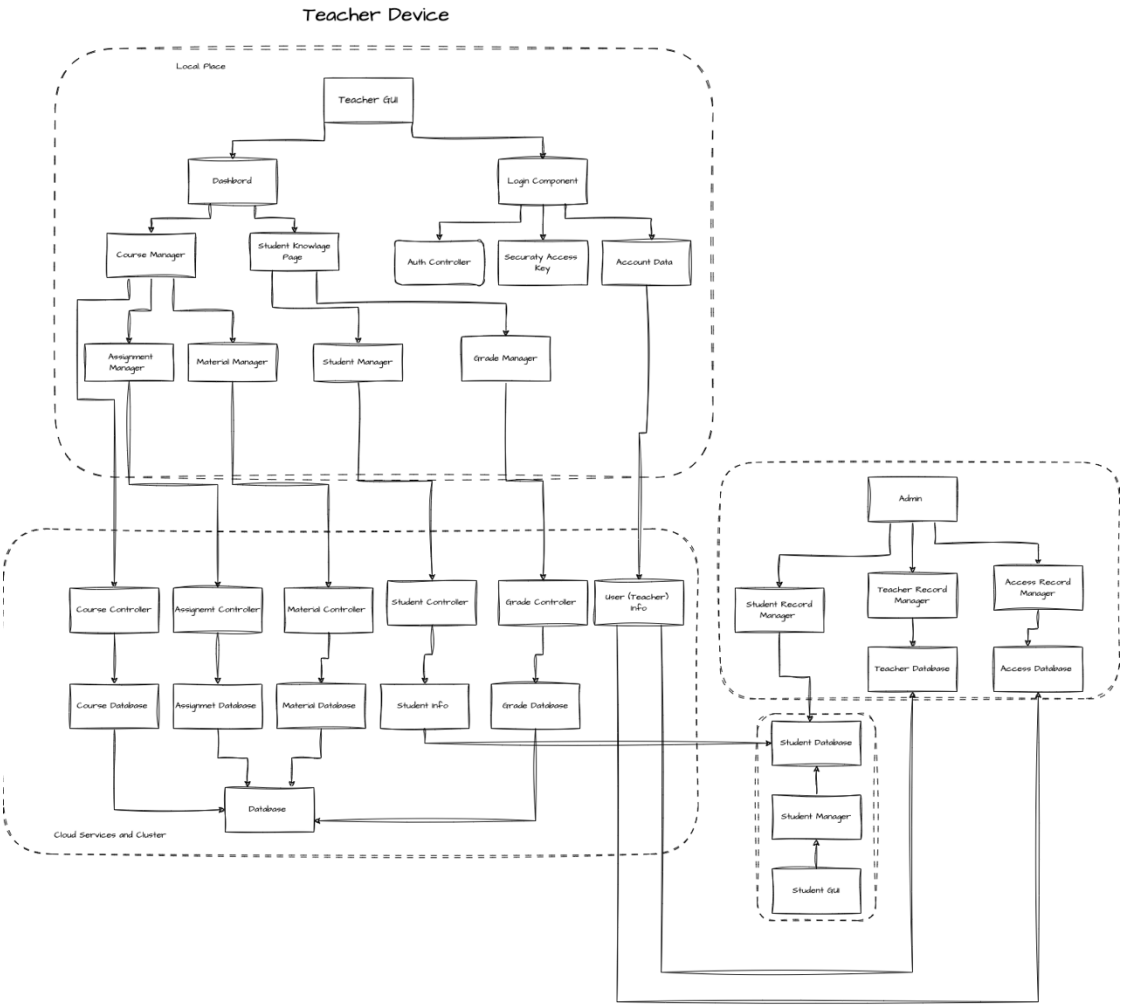


Figure 1.2 New Design

Looking at the old design and the new design, it can be seen that the system has been changed and improved. In the new design (figure 1.2), an additional component such as "Student Manager" has been introduced for more effective management of student information, which allows users to access student data more easily. In addition, new security and account management components such as "Security Access Key" and "Account Data" have been added to improve system security. The new design is more clearly illustrated by the expanded Admin GUI and separate databases for student, teacher and access information, making the whole system more understandable. Compared to the previous design (figure 1.1), the new design is more modular, with each module functioning on its own and in conjunction with other modules, which optimises the workload on the system and makes it easier to maintain and update, as well as to use and understand the system. Components such as "Feedbacks" and "Teachers Information", which were present in the old design, have been restructured in the new design. In addition, the "Assessment Data" feature is handled under a different structure in the new design, which makes the data flow and information

management in the system more effective. In the new design, care has been taken to create the modules in the system in a more organised and advanced way. In the new design, some operations are organised to follow each other. For example, if assignment or material is to be added, a course must be opened for them first, or they can be added to an existing course. This actually makes the system more ball and understandable.

The improvements brought by the new design, especially in the areas of security and access management, better meet the application requirements. The new design is presented as more comprehensive, improved usage and content.

1.1.2. Specifications of Microservices

Service Name	Course Manager
Description:	Manages course creation, update, and deletion.
Provided Services: CreateCourse	Function: Add a new course to the system. Parameters: Course Name, Description, TeacherID
Provided Services: UpdateCourse	Function: Update details of an existing course. Parameters: Course ID, New Details
Provided Services: DeleteCourse	Function: Remove a course from the system. Parameters: Course ID
Requested Services: Database Access	Operation: Insert/Update/Delete course information Parameters: Course Data

Table 1.1 Course Manager Service

Service Name	Assignment Manager
Description:	Handles the management of assignments within courses.
Provided Services: CreateAssignment	Function: Add a new assignment to a specific course.

	Parameters: Course ID, Assignment Details
Provided Services: UpdateAssignment	Function: Modify details of an existing assignment. Parameters: Assignment ID, New Details
Provided Services: DeleteAssignment	Function: Remove an assignment from a course. Parameters: Assignment ID
Requested Services: Database Access	Operation: Insert/Update/Delete assignment information Parameters: Assignment Data

Table 1.2 Assignment Manager Service

Service Name:	Grade Manager
Description:	Manages the recording and updating of student grades.
Provided Services: RecordGrade	Function: Record a new grade for a student's Parameters: Student ID, Grade
Provided Services: UpdateGrade	Function: Update an existing grade. Parameters: Grade ID, New Grade
Requested Services: Database Access	Operation: Insert/Update grade information Parameters: Grade Data

Table 1.3 Grade Manager Service

Service Name:	Material Manager
Description:	Manages educational materials associated with courses.
Provided Services: UploadMaterial	Function: Add new educational material to a course. Parameters: Course ID, Material Details
Provided Services: UdateMaterial	Function: Update details of existing material. Parameters: Material ID, New Details

Provided Services: DeleteMaterial	Function: Remove material from a course. Parameters: Material ID
Requested Services: Database Access	Operation: Insert/Update/Delete material information Parameters: Material Data

Table 1.4 Material Manager Service

Service Name:	Login Service
Description:	Handles user authentication and session management for the system.
Provided Services: AuthenticateUser	Function: Authenticate the credentials provided by the user. Parameters: Username, Password
Provided Services: ValidateSession	Function: Validate the existing user session token. Parameters: Session Token
Provided Services: LogoutUser	Function: Terminate the user's session. Parameters: Session Token
Requested Services: Database Access	Operation: Query user credentials Parameters: Username
Requested Services: Security Service	Operation: Encrypt/Decrypt data Parameters: Data to encrypt/decrypt

Table 1.5 Login Service

1.2.Details of the API

1.2.1. Using Information for API Designs

Course Manager: Allows users to manage courses. The functions of adding, updating and deleting courses are supported by buttons and form fields that are specified and easily accessible in the user interface. When users want to add a new course, they can quickly add it by filling out the relevant form. Updating course details and removing existing courses from the system can be done in a similar way. Assignment and Material sections automatically appear at the bottom of the added courses. This makes it easy for the users of the system to add the necessary content.

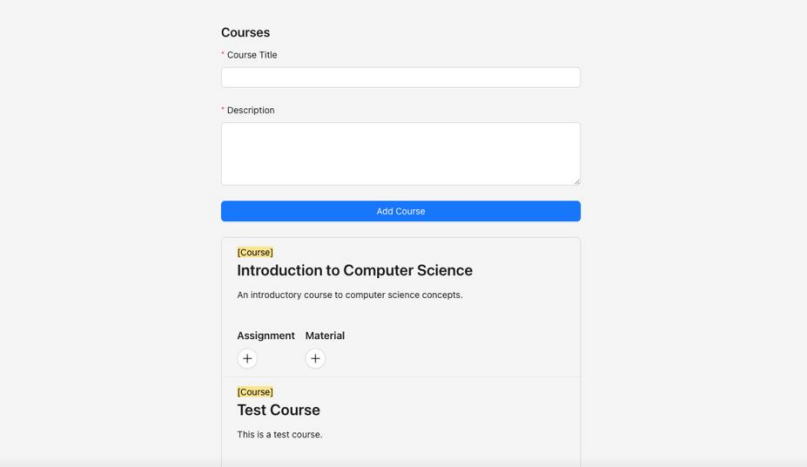


Figure 1.3 Course Manager UI

Assigment Manager: Provides assignment management functions. Users can add assignments, edit or delete existing assignments for a specific course. For these operations, the GUI provides drop-down menus and text boxes for the user to enter or edit assignment information. Each assignment action is designed to provide clear feedback to the user, so they can easily keep track of which assignment they are working on and what actions have been taken.

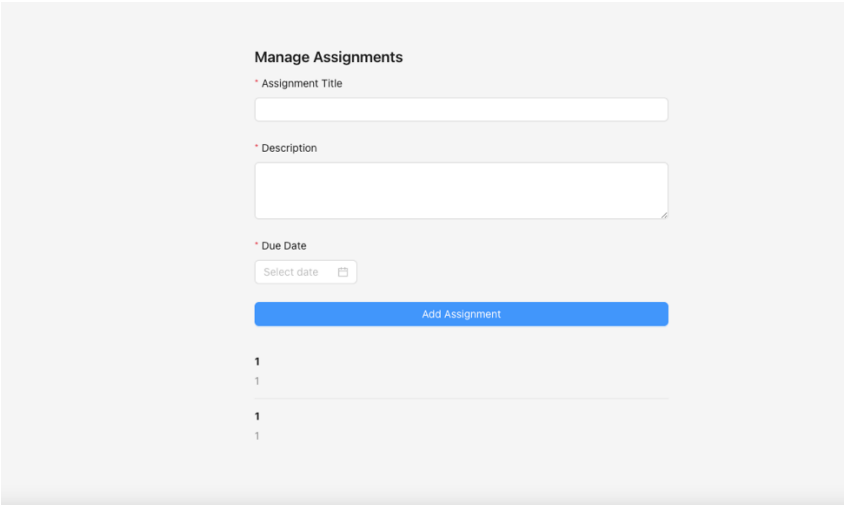


Figure 1.4Assignment Manager UI

Grade Manager: Manages functions such as entering and updating grades. This component is used to evaluate and record students' performance. Users can easily save or update grades by selecting a student from the student list and entering the relevant grade information. These operations allow teachers to enter grades quickly and efficiently.

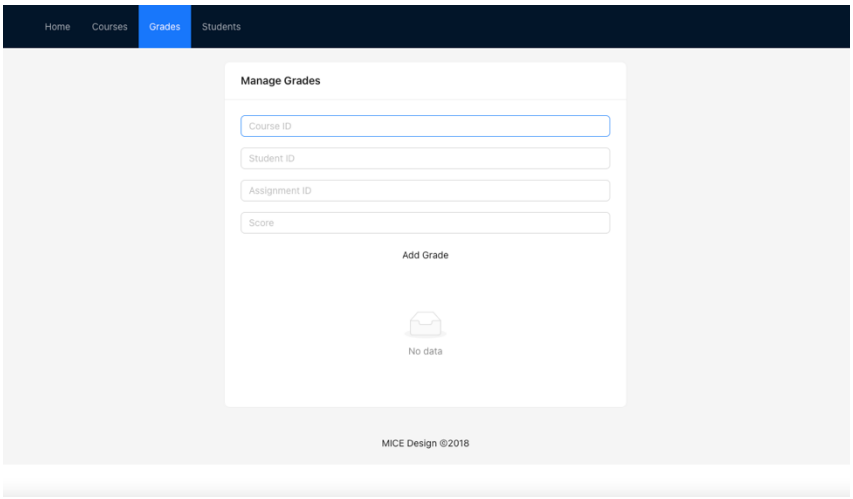


Figure 1.5 Grade Manager UI

Material Manager: Allows the management of course materials. Teachers can add course-specific course materials, update or delete existing materials. Material can be placed as URL in this prototype. When people click on the material, they can directly switch to the URL. These processes are facilitated by file upload tools and editing spreadsheets, so that users can effectively manage their course materials on the system.

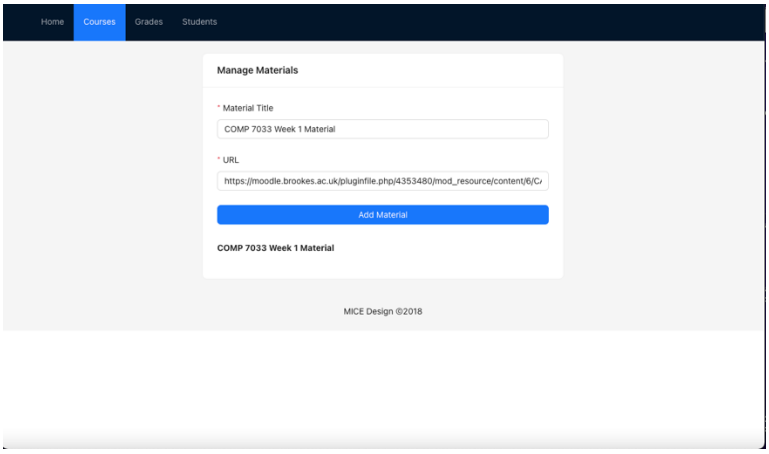


Figure 1.6 Material Manager UI

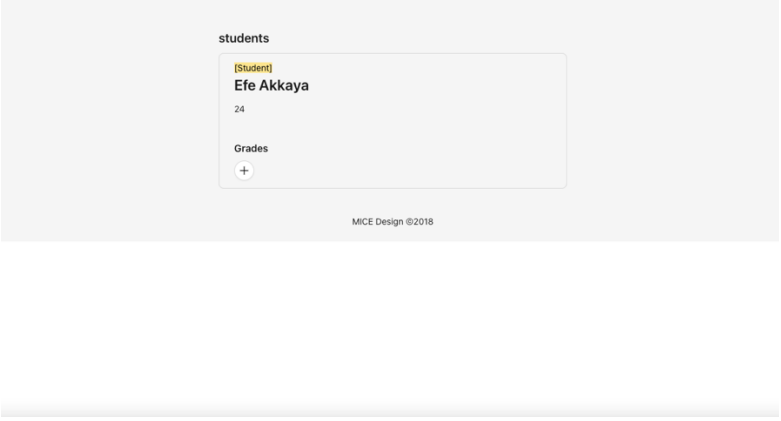


Figure 1.7 Student Info. UI

1.2.2. API Design Format

The API design in the project is realised with RESTful architecture. RESTful API provides access to resources using the HTTP protocol in general. In this approach, resources are expressed with URLs and various operations are performed on these resources with HTTP methods (GET, POST, PUT, DELETE). HTTP Operators and Resource URIs: RESTful APIs perform operations on a specific resource using HTTP operators (GET, POST, PUT, DELETE). These operations usually include CRUD operations. Operant Parameters define the content and processing of API requests. They are usually carried as URL parameters, query strings or body data.

Description:	Create Course
HTTP Method:	POST
Source URI:	/courses
Submitted Data (JSON):	{"courseName": "Introduction to REST", "description": "A basic course on RESTful API design" }
Operation:	Adds a new course to the system.

Table 1.6 Create Course

Description:	Update Course
HTTP Method:	PUT
Source URI:	/courses/{courseId}
Submitted Data (JSON):	{"courseName": "Advanced REST", "description": "An advanced course on RESTful API design" }
Operation:	Updates the details of an existing course.

Table 1.7 Update Course

Description:	Delete Course
HTTP Method:	DELETE
Source URI:	/courses/{courseId}
Operation:	Removes a specific course from the system.

Table 1.8 Delete Course

1.3.Link GUI to API

In the project, a structure has been established in which each GUI component communicates with the relevant microservices. For example, adding a course via CourseComponent sends an HTTP POST request to the relevant Course Controller using the RESTful API in the background. This request contains the course information received from the user and the microservice processes this data and saves it as a new course in the database.

The details of the request sent to the API for each operation from the UI are specified in the API specifications. Table 1.9 contains an example.

HTTP Metod:	PUT
Endpoint:	/courses/{courseId}
Payload:	{"courseName": "New Course Name", "description": "Updated Description"}

Table 1.9 Description Example

Successful Response:	Course updated successfully"
Error Response:	Failed to update course

Table 1.10 Response Example

1.4.Login

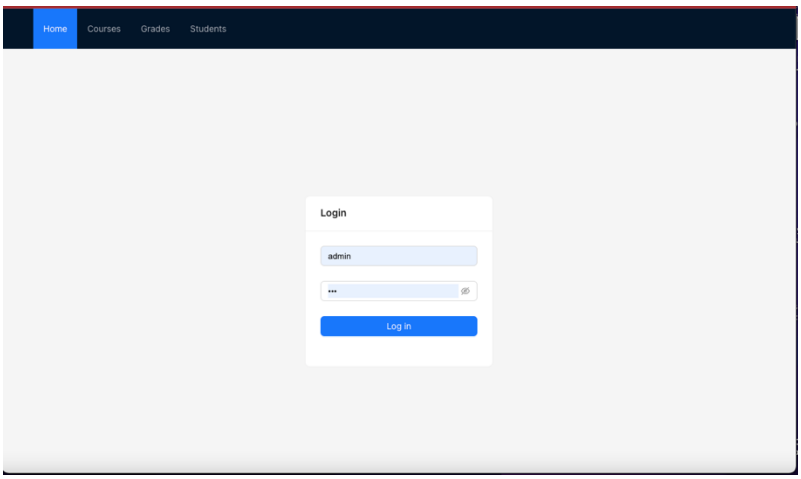


Table 1.11 Login Page

```
exports.loginUser = (req, res) => {
  const { username, password } = req.body;
  // Basit kullanıcı doğrulama
  if (username === 'admin' && password === '123') {
    res.json({ message: 'Login successful', token: 'fake-jwt-token' });
  } else {
    res.status(401).json({ message: 'Login failed, username or password incorrect' });
  }
};
```

	Message
Successful:	{"message": "Login successful", "token": "fake-jwt-token" }
Failure:	{ "message": "Login failed, username or password incorrect" }

Table 1.12 Message Table

2. Implementation

2.1.Forend, Backend and API's

In the backend and forend of the system prepared in this project for Cla-Clo Teacher. There are login system, course management, assignment management, grade management, material management, api for student as mock system and mock-user-server which returns an empty message with 200 message for admin as another mock system. Figure shows all forent and backend files of the system.

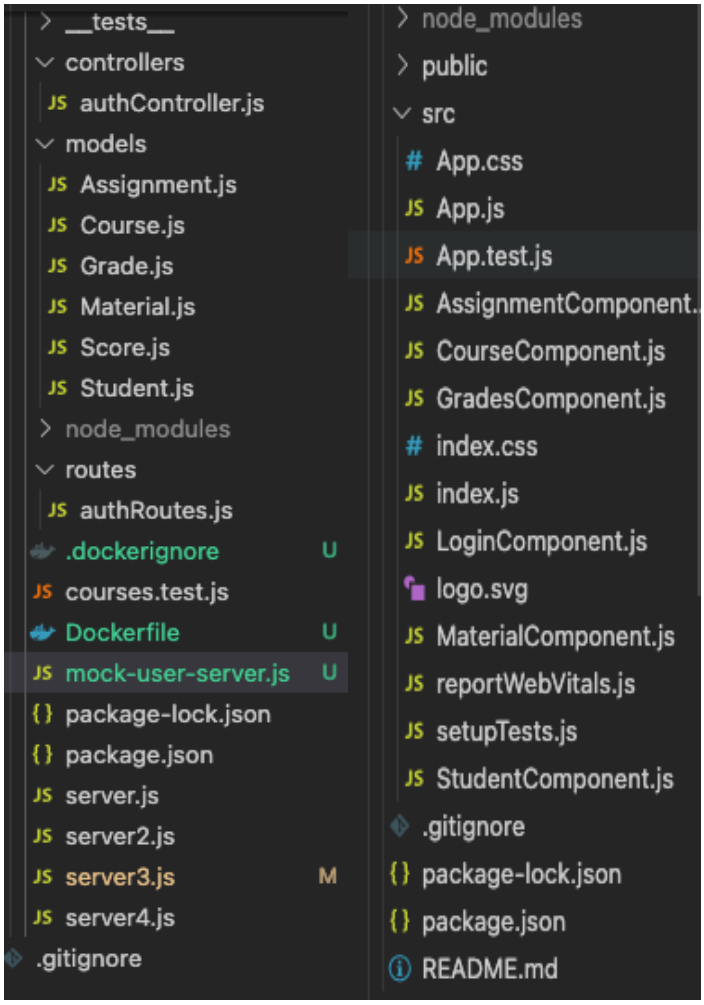


Figure 2.1 Bakend and Forend(API's)

2.2.Computing Techniques

2.2.1. MongoDB Cluster

MongoDB is more suitable for this project than other SQL styles with its scalability and flexible schema structures among NoSQL databases. MongoDB Cluster was preferred for the management of large volumes of data and fast access. MongoDB cluster can automatically manage data distribution and load balancing, thus ensuring high availability and fault tolerance in the application's database.

“const mongoose = require('mongoose');
mongoose.connect('mongodb+srv://<username>:<password>@<cluster-url>/<database>?retryWrites=true&w=majority', {
  useNewUrlParser: true,
  useUnifiedTopology: true
});”

The example shows how the system is connected to the cluster. The figure shows an example of the data held by the teacher system in the cluster.

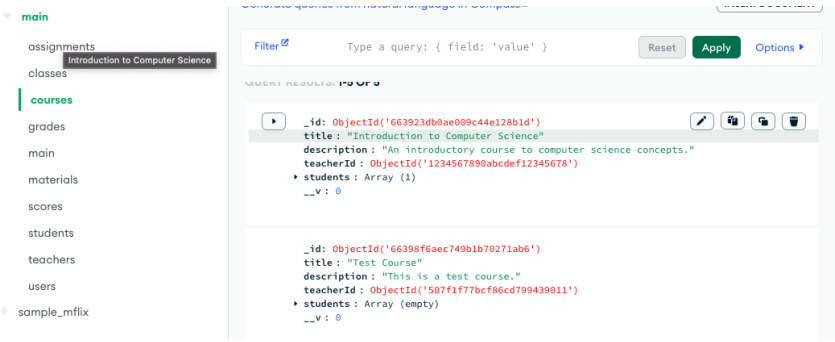


Figure 2.2 Cluster

2.2.2. Docker

Docker is used in this project to put application dependencies and environments in a container. By using Docker, it is ensured that the application will work the same way in different environments (development, test, production). In this project, it was used to upload services to the google cloud platform.

```
“FROM node:20
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 8000
```

Thanks to this Dockerfile, Node.js applications are containerised. The dependencies of the application are installed and services are started.

2.2.3. Google Cloud Run

In this project, Google cloud run was used instead of Kubernetes. Google cloud run has advantages such as automatic scaling and load balancing with simple configuration. Kubernetes is more suitable for more complex structures, but the requirements of our project could be met with the features offered by Cloud Run. Thanks to Google cloud run, servers containerised with docker could be moved into the Google cloud platform and run.

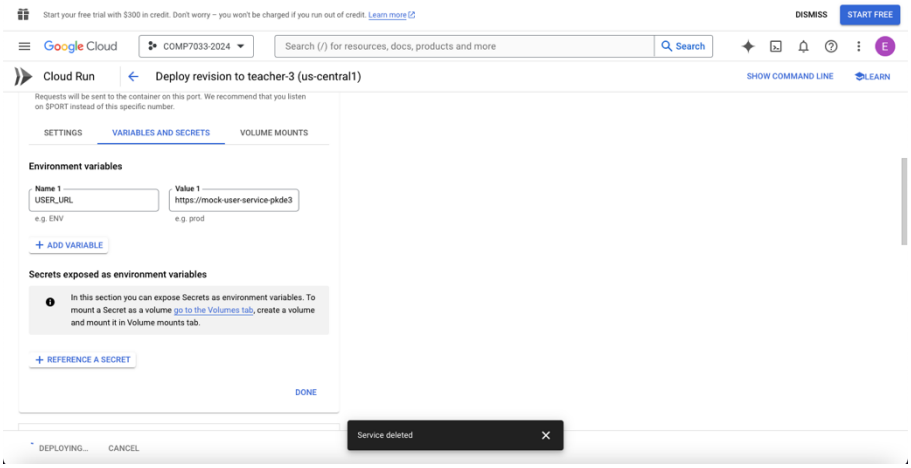


Figure 2.3 Google Cloud Run Deployment Step1

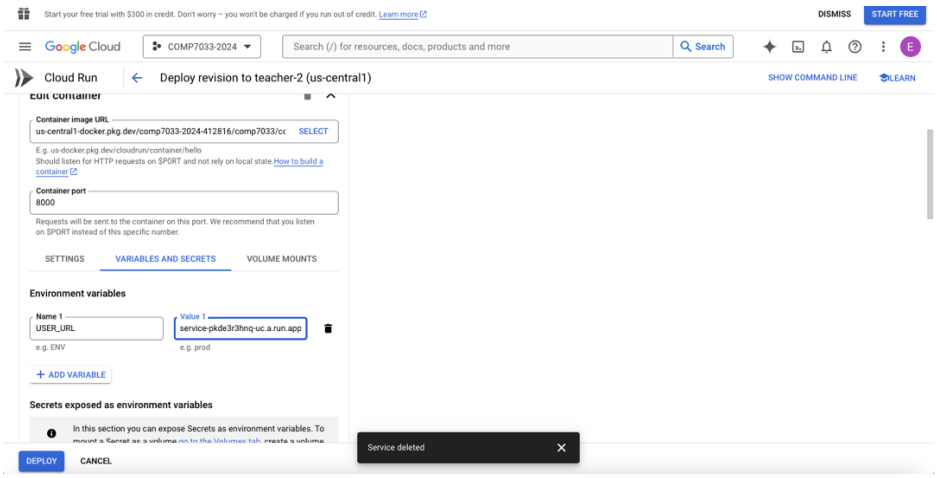


Figure 2.4 Google Cloud Run Deployment Step2

2.2.4. Load Balancer

Google Cloud Run has its own load balancing mechanisms. Therefore, the load balancing needs of the application are met by the cloud platform.

2.3. Integration With Other Subsystems

There should be four different subsystems in the project. Student, Teacher, Admin, ops. The subsystem made from this project was made in such a way that it can integrate and communicate with the others.

However, since the connection with the subsystems of my other project mates could not be established, mock/dummyAPIs were installed for student and admin.

Mock API for Admin:
The API for the admin service returns a simple JSON response when needed. It can receive requests such as GET /users/:userId and returns an empty response without any action.
Mock API for Student:
Used for managing student information, this API is designed to simulate functions such as listing student information, adding new students, etc. It provides a list of students via the GET /students endpoint, and receives new student registrations via the POST method. Since teachers should be able to see the students in the system, I already had a studen API, so I added a mock API and manually entered students. Normally, students should log in to the system with the student subsystem and I should get the student information from the database that goes to the admins via the student.

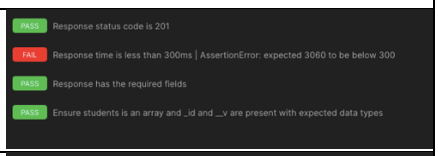
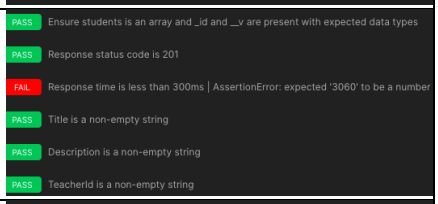
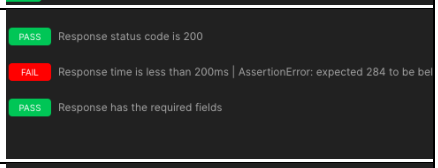
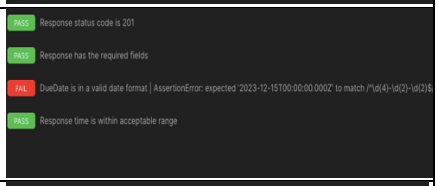
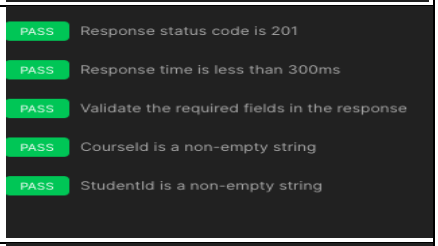
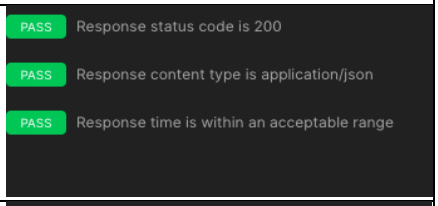
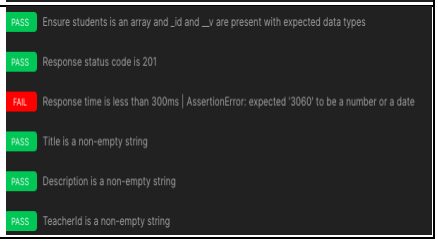
2.4. How does the team collaborate

The communication of the team in general was very disjointed and not good. Since the communication was bad, we could not exchange information. Everyone developed their own subsystem. I could not integrate it because my project was late. I had a lot of difficulty in developing the project because everyone was alone in the project and information could not be exchanged. One person never communicated and I think he did not even do the project. The team could never act like a team.

3. Testing and Integration

3.1. Postman Tests

Postman application was used in the project to test the teacher system created in the project. Thanks to this application, both functional and performance tests could be performed on the system. In the table below, almost all functions of the system were tested and the results were noted.

type	Pass/fail	time	Response size	Table
Course:POST	PASS	3.06s	602B	
Course:GET	PASS	8.88s	1.36kb	
Course:PUT	PASS	2.73s	573B	
Course:DELET	PASS	284ms	404B	
Assignment:POST	PASS	200ms	463B	
Assignment:GET	PASS	180ms	652B	
Grade:POST	PASS	172ms	445B	
Grade:GET	PASS	180ms	616B	
Material:POST	PASS	3.06s	602B	

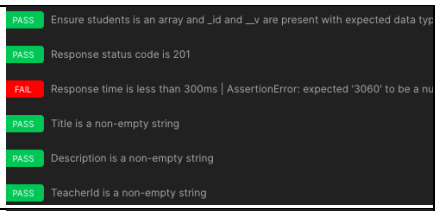
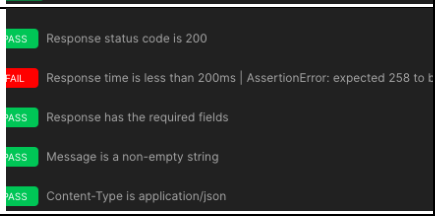
Material:GET	PASS	238ms	530B	
Materal:DELET	PASS	258 MS	406B	

Table 3.1 Testing Table

Mock-api-admin	PASS	145MS	495B	

Table 3.2Mock-api Testing table

As a result of the tests with Postman, it was observed that the parts of the system worked completely. However, when we look at the functionality, it is observed that many functions of the system are below 200ms. This shows us that the functionality of the system should be increased.

3.2.Scenario Test

The system was also tested with the GUI side. The system was logged in from the login page. After going to the course section, first assignment was added and then the material was added. During these processes, results similar to the test table were obtained in terms of performance. All works except the grading part on the student profile. In the GUI part, the part of adding a grade by pressing the button under the learner's information from the student section has not been fully developed. In Section 1.2.1, there are visuals of some of the tests performed.

Test No:	Use Case	Inputs	Expected Outputs	Status and comments
Course Creation	Adding a new course to the system.	Title, Description, TeacherID, Students	adding a course to the system .201 code.	PASS
Course Listing	Listing all courses.		List of courses and code 200.	PASS
Assignment Creation	Adding an assignment to a specific course.	Title, Description, DueDate, CourseID	Adding the assignment to the course content and 201.	PASS
Assignment Listing	Listing all assignments for a specific course.	CourseID	List of assignments and code 200.	PASS
Material Creation	Adding material to a specific course.	Title, URL, CourseID	successful addition of material and 201.	PASS
Material Listing	Listing all materials for a specific course.	CourseID	List of materials and 200.	PASS
Material URL opening	Click to material's URL.		URL need to connect and open a new web page.	PASS
Performance Testing	Testing the system performance.	Multiple concurrent requests	Response time and transaction success.	As the system load increases, the slowdown increases at the same rate. At some point

				it becomes unusable
Scalability Testing	Testing the scalability of the system.	Increasing user load under test	System resource usage and response time.	Although some services respond quickly, the majority respond in more than 200ms.(table 3.2)
Fault Tolerance Testing	Observing system behavior when a microservice fails.	Manually crashing a microservice	System continues to function with other services.	When the systems working under another subsystem in the system are corrupted, the system can continue to work, but when the systems containing other subsystems such as Course manager are corrupted, the system's operation is disrupted.

Table 3 Scenario Testing Table

3.3.Code test

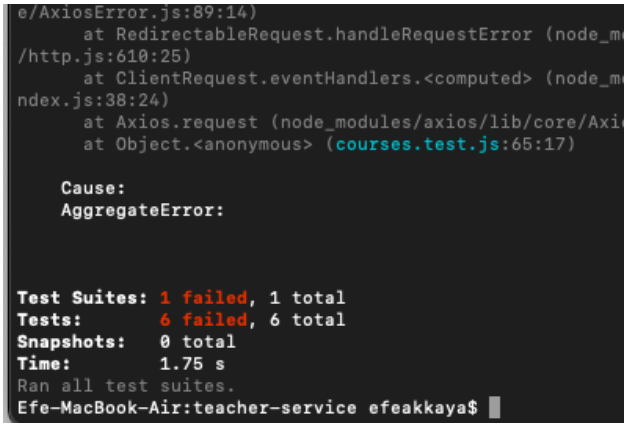


Table 3.4 Code Test

As can be seen, successful results were achieved in the tests performed with code in the system. The failed test in the image is not related to the system, there is an unsolvable problem in the test. In the tests made from Postman, the function has been proven to work.

3.4.Fault Tolerance Test

This test actually happened naturally while the system was being built. During the construction of the system, the modules naturally gave errors. As a result, when certain parts of the system encountered errors, other parts had the opportunity to be observed. When the system was finished building and fully operational, these tests were performed by trying this sever again. As a result of this. It was observed that Assignment and Material manager systems could still work in case of some corruption. But when the course module breaks, the assignment and material modules do not work. Because they cannot be accessed.

https://bitbucket.org/brookesrobotics/comp7033/src/main/