

Git Hub url - <https://github.com/SHIVAKUMARGOWDAH222/student-fee-management-microservice.git>

Prerequisite

- IntelliJ IDE
- Docker Desktop
- Java 17

Steps

Clone the project in local and then import all the microservices as modules inside base container called "school platform project".

Navigate to Project Folder where Docker-compose file is present and Run Docker-compose up command. This will start the Kafka server and zookeeper that is being used for event driven architecture.

Navigate to the pom.xml file located in folder "schoolplatform" and run "mvn clean install" command to build the project

Start Individual Microservices. Eureka Server needs to be started first.

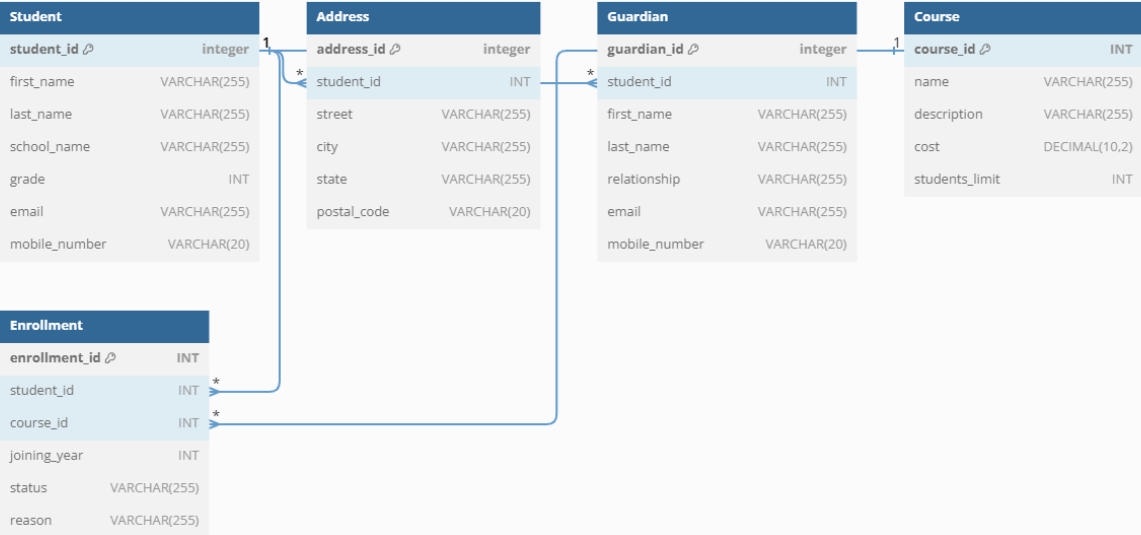
There are 5 microservices

- API Gateway
- Eureka Server
- Fee Process
- Receipt
- Student Management

DATA BASE TABLE DESIGN QUERIES FOR MICRO SERVICES

Student Management

```
• CREATE TABLE Student (  
    student_id INT AUTO_INCREMENT PRIMARY KEY,  
    first_name VARCHAR(255) NOT NULL,  
    last_name VARCHAR(255) NOT NULL,  
    school_name VARCHAR(255) NOT NULL,  
    grade INT,  
    email VARCHAR(255),  
    mobile_number VARCHAR(20)  
);  
  
CREATE TABLE Address (  
    address_id INT AUTO_INCREMENT PRIMARY KEY,  
    student_id INT NOT NULL,  
    street VARCHAR(255) NOT NULL,  
    city VARCHAR(255) NOT NULL,  
    state VARCHAR(255) NOT NULL,  
    postal_code VARCHAR(20),  
    CONSTRAINT fk_address_student FOREIGN KEY (student_id) REFERENCES  
Student(student_id)  
);  
  
CREATE TABLE Guardian (  
    guardian_id INT AUTO_INCREMENT PRIMARY KEY,  
    student_id INT NOT NULL,  
    first_name VARCHAR(255) NOT NULL,  
    last_name VARCHAR(255) NOT NULL,  
    relationship VARCHAR(255),  
    email VARCHAR(255),  
    mobile_number VARCHAR(20),  
    CONSTRAINT fk_guardian_student FOREIGN KEY (student_id) REFERENCES  
Student(student_id)  
);  
  
CREATE TABLE Course (  
    course_id INT AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(255) NOT NULL,  
    description VARCHAR(255),  
    cost DECIMAL(10,2) NOT NULL,  
    students_limit INT NOT NULL,  
    CONSTRAINT CHECK_cost CHECK (cost > 0)  
);  
  
CREATE TABLE Enrollment (  
    enrollment_id INT AUTO_INCREMENT PRIMARY KEY,  
    student_id INT NOT NULL,  
    course_id INT NOT NULL,  
    joining_year INT NOT NULL,  
    status VARCHAR(255) NOT NULL,  
    reason VARCHAR(255) NOT NULL,  
    CONSTRAINT fk_enrollment_student FOREIGN KEY (student_id) REFERENCES  
Student(student_id),  
    CONSTRAINT fk_enrollment_course FOREIGN KEY (course_id) REFERENCES  
Course(course_id)  
);
```




FEES PROCESS

```
CREATE TABLE Fee (  
    fee_id INT AUTO_INCREMENT PRIMARY KEY,  
    totalFees DECIMAL(10,2) NOT NULL  
);  
  
CREATE TABLE Admissions (  
    admission_id INT AUTO_INCREMENT PRIMARY KEY,  
    enrollment_id INT NOT NULL,  
    fee_id INT NOT NULL,  
    student_id INT NOT NULL,  
    course_id INT NOT NULL,  
    joining_year INT NOT NULL,  
    cost DECIMAL(10,2) NOT NULL,  
    CONSTRAINT fk_admissions_fee FOREIGN KEY (fee_id) REFERENCES Fee(fee_id)  
);  
  
CREATE TABLE Payment (  
    payment_id INT AUTO_INCREMENT PRIMARY KEY,  
    fee_id INT NOT NULL,  
    transaction_date_time DATETIME NOT NULL,  
    amount DECIMAL(10,2) NOT NULL,  
    transaction_status VARCHAR(255) NOT NULL,  
    reference_number INT NOT NULL,  
    CONSTRAINT fk_payment_fee FOREIGN KEY (fee_id) REFERENCES Fee(fee_id)  
);
```

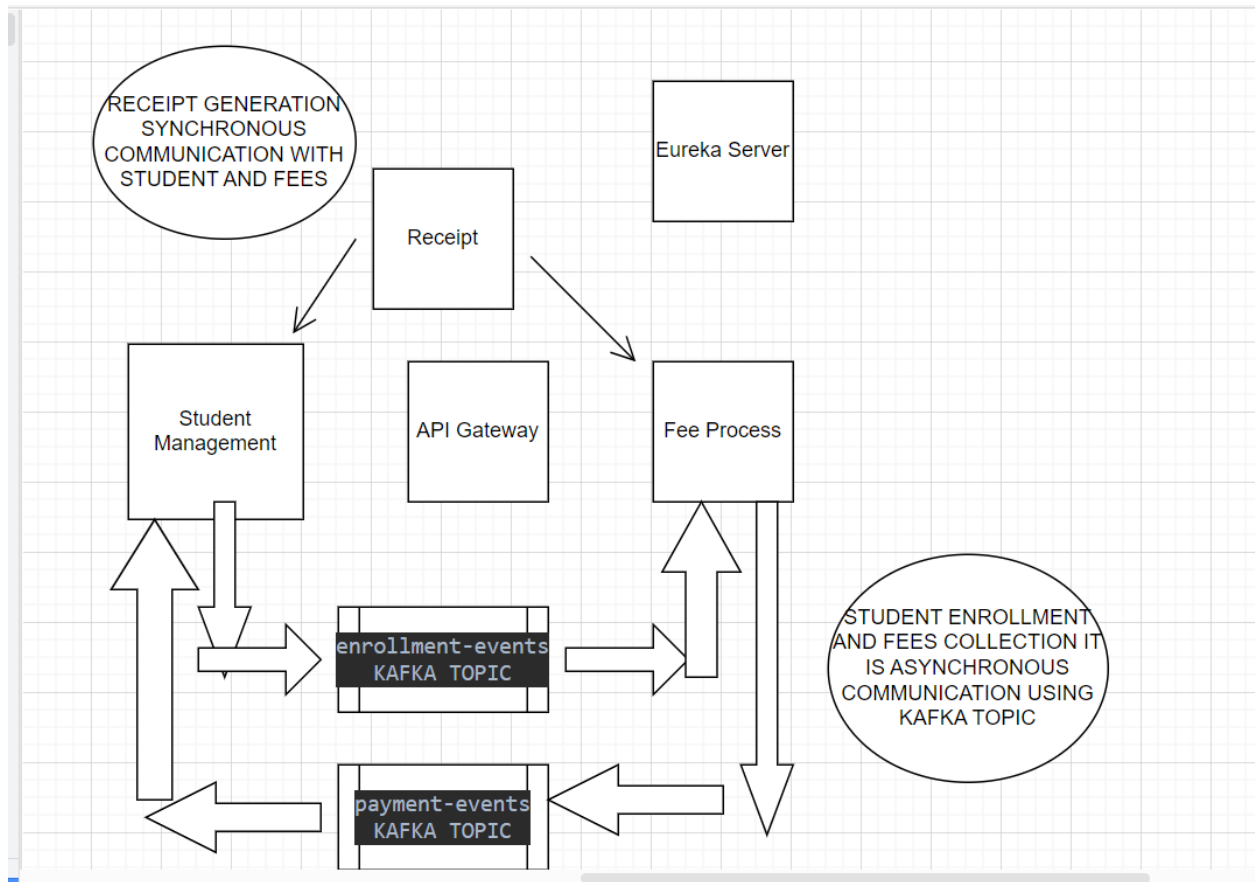
Fee	
fee_id 	INT
totalFees	DECIMAL(10,2)



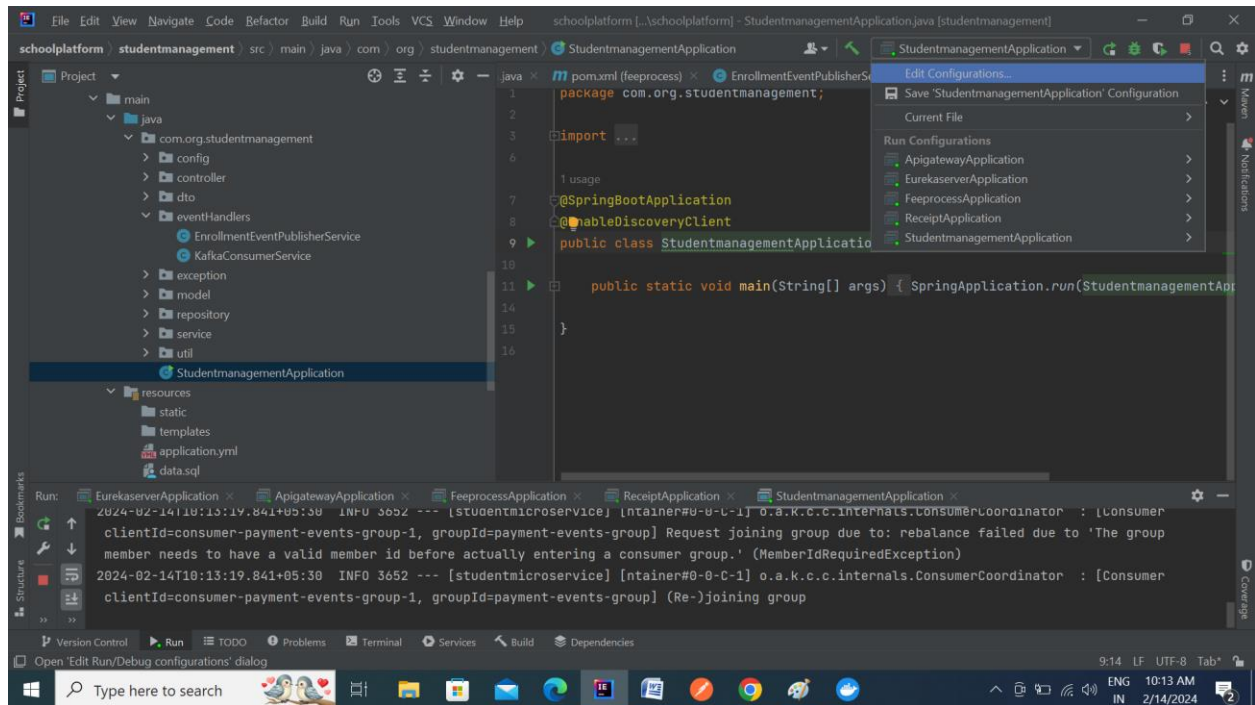
Admissions	
admission_id 	INT
enrollment_id	INT
fee_id	INT
student_id	INT
course_id	INT
joining_year	INT
cost	DECIMAL(10,2)

Payment	
payment_id 	INT
fee_id	INT
transaction_date_time	DATETIME
amount	DECIMAL(10,2)
transaction_status	VARCHAR(255)
reference_number	INT

Architecture



Once all the services are up



Student Service Swagger URL - <http://localhost:8082/swagger-ui/index.html>

Eureka server Url - <http://localhost:8761/>

1st API CALL to Register the Student

student-controller

POST

/api/v1/students/

Parameters

No parameters

Request body required

application/json

```
{
  "firstName": "shivamani",
  "lastName": "Kumar",
  "schoolName": "RRHS",
  "grade": 1,
  "email": "shiva@gmail.com",
  "mobileNumber": "1234567890",
  "addresses": [
    {
      "street": "1",

```

Request URL

http://localhost:8082/api/v1/students/

Server response

Code


Details

201

Undocumented

Response body

```
{
  "id": 2,
  "firstName": "Shivaraj",
  "lastName": "string",
  "schoolName": "RRHS",
  "grade": 1,
  "email": "shiva@gmail.com",
  "mobileNumber": "1234567890",
  "addresses": [
    {
      "id": 2,
      "street": "1",
      "city": "dubai",
      "state": "UAE",
      "postalCode": "1234"
    }
  ],
  "guardians": [
    {
      "id": 2,
      "firstName": "Raj",
      "lastName": "Kumar",
      "relationship": "Father",
      "email": "abc@gmail.com",
      "mobileNumber": "1234567890"
    }
  ]
}
```

 Download

2nd Service to check which courses are available to enroll for student.

GET

/api/v1/courses

Parameters

Cancel

Name	Description
paginationDTO * required object (query)	<pre>{ "pageNumber": 0, "pageSize": 10, "sortDirection": "Asc", "sortBy": "id"} </pre>

Execute

Clear

localhost:8082/swagger-ui/index.html#/course-controller/getAllCourses

YouTube Maps News Translate Gmail All Bookmarks

Server response

Code

Details

200

Response body

```
{  "content": [    {      "id": 1,      "name": "Introduction to Programming",      "description": "Learn the basics of computer programming",      "cost": 100,      "studentLimit": 100    },    {      "id": 2,      "name": "Biology 101",      "description": "Explore the fundamentals of life",      "cost": 150,      "studentLimit": 100    },    {      "id": 3,      "name": "Literature",      "description": "Dive into classic and modern literature",      "cost": 200,      "studentLimit": 50    },    {      "id": 4,      "name": "Introduction to Science",      "description": "Science"    }  ]}
```

Download

Response headers

3rd service to enroll into course based on the course id

enrollment-controller

POST /api/v1/enrollments/

Parameters

No parameters

Request body required

application/json

```
[
  {
    "joiningYear": 2020,
    "studentId": 2,
    "courseId": 4,
    "cost": 0
  }
]
```

Request URL

http://localhost:8082/api/v1/enrollments/

Server response

Code

Details

201

Undocumented

Response body

```
[
  {
    "id": 1,
    "joiningYear": 2020,
    "courseId": 4,
    "status": "PENDING",
    "reason": "Initiated"
  }
]
```

Download

Response headers

```
connection: keep-alive
content-type: application/json
date: Wed, 14 Feb 2024 05:12:29 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Responses

Code

Description

Links

When the student is enrolled for the course. Initial Status will be pending until we get response from fee process microservice. This communication is done using Kafka broker.

Once we get response from Kafka Broker , about the status. The status will be updated. We can verify the status by calling getStudent api based on Student id. The status will be updated to Success. This is event driven communication.

4th service to check enrollment status based on student Id

The image shows the Swagger UI for the endpoint `GET /api/v1/students/{id}`. The **Parameters** tab is active, showing a required path parameter `id` of type `integer($int64)` with a value of `2`. Below the parameter field are `Execute` and `Clear` buttons. The **Responses** tab is also visible, showing a `Curl` command and the `Request URL`.

Parameters

Name	Description
<code>id</code> * required	
<code>integer(\$int64)</code>	
(path)	

`Execute` `Clear`

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8082/api/v1/students/2' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8082/api/v1/students/2
```

The image shows a web browser displaying the response body for the endpoint `localhost:8082/swagger-ui/index.html#/student-controller/getStudentById`. The status is `200`. The response body is a JSON object containing student details, guardians, and enrollment information.

Response body

```
{
  "id": 2,
  "street": "1",
  "city": "dubai",
  "state": "UAE",
  "postalCode": "1234",
  "guardians": [
    {
      "id": 2,
      "firstName": "Raj",
      "lastName": "Kumar",
      "relationship": "Father",
      "email": "abc@gmail.com",
      "mobileNumber": "1234567890"
    }
  ],
  "enrollments": [
    {
      "id": 1,
      "joiningYear": 2020,
      "courseId": 4,
      "status": "SUCCESS",
      "reason": "SUCCESS"
    }
  ]
}
```

Response headers

```
connection: keep-alive
content-type: application/json
date: Wed, 14 Feb 2024 05:19:23 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Student Management service sends enrollment event to enrollment event Kaka topic.
FeeProcessService listens to the event and initiates Payment and sends the status to payment event Kaka topic. The Student Management service listens to this topic and updates status in its database in enrollment table.

Swagger URL - <http://localhost:8085/swagger-ui/index.html>

5th Service to generate Receipt Based on the Student ID. This is done using webclient

The screenshot displays the Swagger UI interface for a service named 'receipt-controller'. The selected endpoint is a GET request to '/api/receipt/getData/{studentid}'. A parameter 'studentid' of type 'integer(\$int64)' is required and has a value of '2' entered. The 'Execute' button is visible. Below the endpoint details, the 'Responses' section shows a 200 status code with a detailed JSON response body and headers. The response body contains a 'Combined Result' object with student, guardian, enrollment, and payment information. The response headers include 'connection: keep-alive', 'content-length: 752', 'content-type: text/plain; charset=UTF-8', 'date: Wed, 14 Feb 2024 05:29:09 GMT', and 'keep-alive: timeout=60'.

request-controller

GET /api/receipt/getData/{studentid}

Parameters

Name	Description
studentid ^{required}	
integer(\$int64)	
(path)	

Execute

Responses

200

Response body

```
Combined Result: {"id":2,"firstName":"Shivaraj","lastName":"string","schoolName":"RRHS","grade":1,"email":"shiva@gmail.com","mobileNumber":"1234567890","addresses":[{"id":2,"street":"1","city":"dubai","state":"UAE","postalCode":"1234"}],"guardians":[{"id":2,"firstName":"Raj","lastName":"kumar","relationship":"Father","email":"abc@gmail.com","mobileNumber":"1234567890"}],"enrollments":[{"id":1,"joiningYear":2020,"courseId":4,"status":"SUCCESS","reason":"SUCCESS"}],{"admissions":{"admissionId":1,"enrollmentId":1,"feeId":1,"studentId":2,"courseId":4,"joiningYear":2020,"cost":100.0},"payments":[{"paymentId":1,"feeId":null,"transactionId":1,"date":"2024-02-14T05:12:30.313+00:00","amount":100.0,"transactionStatus":"SUCCESS","referenceNumber":12}]}
```

Response headers

```
connection: keep-alive
content-length: 752
content-type: text/plain; charset=UTF-8
date: Wed, 14 Feb 2024 05:29:09 GMT
keep-alive: timeout=60
```

Responses

Code	Description	Links
200	OK	No links

These are the major services for workflow. Apart from these there are services to perform CRUD operations

H2 DataBase Table Screenshots

<http://localhost:8082/h2-console/login.do?sessionId=8a030d1a6dbc8504a00642502dbc0421>

org.h2.Driver

DB : jdbc:h2:mem:student_information

username : user

password : password

The screenshot displays the H2 Database console interface in a web browser. The address bar shows the URL: `localhost:8082/h2-console/login.do?sessionId=8a030d1a6dbc8504a00642502dbc0421`. The interface includes a sidebar on the left with a tree view of the database schema, a main area for SQL statements, and a right sidebar with helpful information.

Database Schema (Left Sidebar):

- COURSE**
 - COURSE_ID
 - NAME
 - DESCRIPTION
 - COST
 - STUDENTS_LIMIT
 - Indexes
- ENROLLMENT**
 - ENROLLMENT_ID
 - STUDENT_ID
 - COURSE_ID
 - JOINING_YEAR
 - STATUS
 - REASON
 - Indexes
- GUARDIAN**
- STUDENT**
 - STUDENT_ID
 - FIRST_NAME
 - LAST_NAME
 - SCHOOL_NAME
 - GRADE
 - EMAIL
 - MOBILE_NUMBER
 - Indexes
- INFORMATION_SCHEMA**
- Users**
- H2 2.2.224 (2023-09-17)

Main Area (Top):

Run Run Selected Auto complete Clear SQL statement:

Important Commands (Right Sidebar):

Icon	Command
?	Displays this Help Page
📜	Shows the Command History
🚀	Ctrl+Enter Executes the current SQL statement
👤	Shift+Enter Executes the SQL statement defined by the text selection
🔄	Ctrl+Space Auto complete
🔌	Disconnects from the database

Sample SQL Script (Right Sidebar):

```
Drop the table if it exists
Create a new table
with ID and NAME columns
Add a new row
```

```
DROP TABLE IF EXISTS TEST;
CREATE TABLE TEST(ID INT PRIMARY KEY,
NAME VARCHAR(255));
INSERT INTO TEST VALUES(1, 'Hello');
INSERT INTO TEST VALUES(2, 'Hello');
```

<http://localhost:8084/h2-console/login.jsp?sessionId=8a030d1a6dbc8504a00642502dbc0421>

org.h2.Driver

DB : jdbc:h2:mem:payment_information

username : user

password : password

The screenshot shows the H2 Database Console web interface in a browser. The address bar displays the URL: `localhost:8084/h2-console/login.do?sessionId=f4ae27039f6d4bbf830d506f6af1a0e7`. The interface includes a top navigation bar with links to YouTube, Maps, News, Translate, and Gmail. Below this is a toolbar with icons for running queries, auto-commit, max rows (set to 1000), auto-complete, and auto-select. The main content area is divided into two panels. The left panel shows a tree view of the database schema, including tables like `ADMISSIONS`, `FEE`, `PAYMENT`, and `TRANSACTION`, along with their indexes. The right panel contains a large text area for the SQL statement, a section titled "Important Commands" with a table of shortcuts, and a section titled "Sample SQL Script" with a table of sample queries.

Command	Description
?	Displays this Help Page
Ctrl+Enter	Executes the current SQL statement
Shift+Enter	Executes the SQL statement defined by the text selection
Ctrl+Space	Auto complete
Ctrl+Q	Disconnects from the database

Sample SQL Script	SQL Statement
Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Insert a new row	INSERT INTO TEST VALUES(1, 'Hello');

