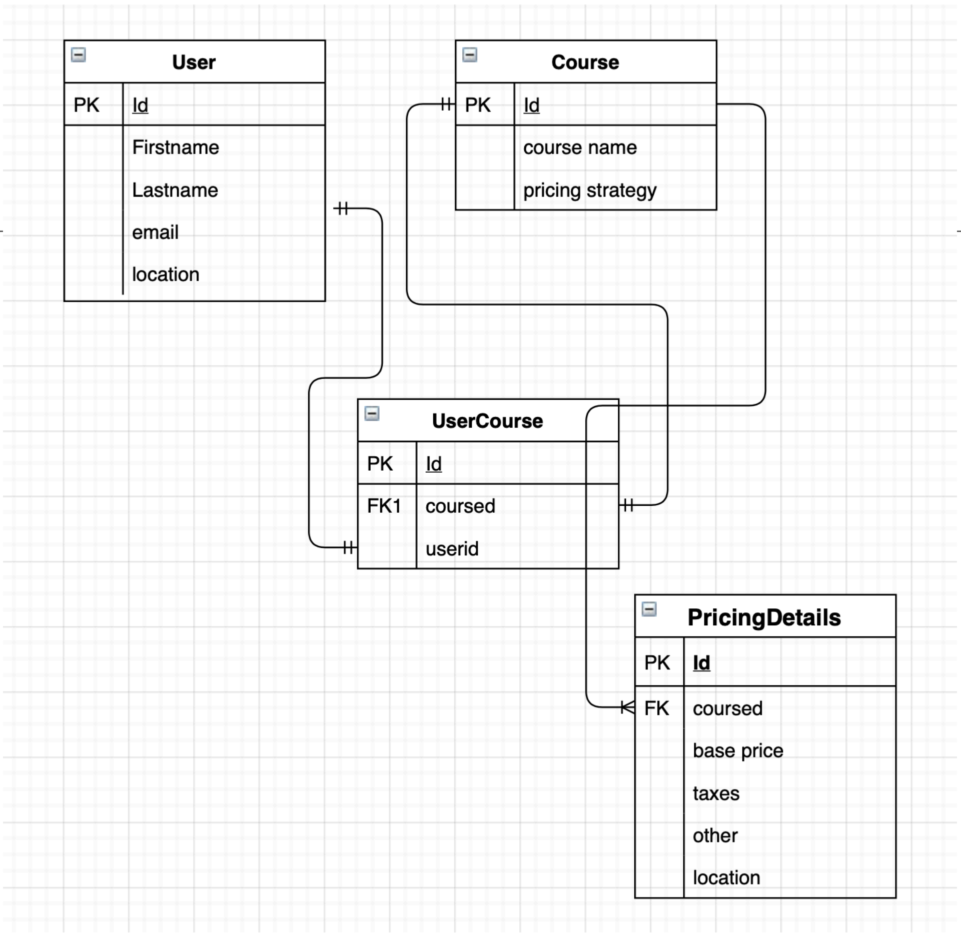
## Proposed API signature:

URL: localhost:8080/pricing/<courseID>/<userID>
Request: courseID (unsingned integer)
userID (unsigned integer)
Response:
Pricing Details

```
{
  "id": 2,
  "baseprice": 100,
  "taxes": 15,
  "other": 20,
  "location": "US",
  "course": {
    "id": 1,
    "coursename": "Python Getting Started",
    "pricingstrategy": "Free Courses”
 }
}
```

## Database Setup

ER Diagram



## Project Setup:

This application is a spring boot application developed using JPA and H2 database.

The database configurations can be found in application.properties file.

**Installation steps:**

Step1. Clone the repository to a local directory

Step2. Cd into project directory

Step3. Run “mvn clean install”

Step4. Run “java -jar target/demo-0.0.1-SNAPSHOT.jar”

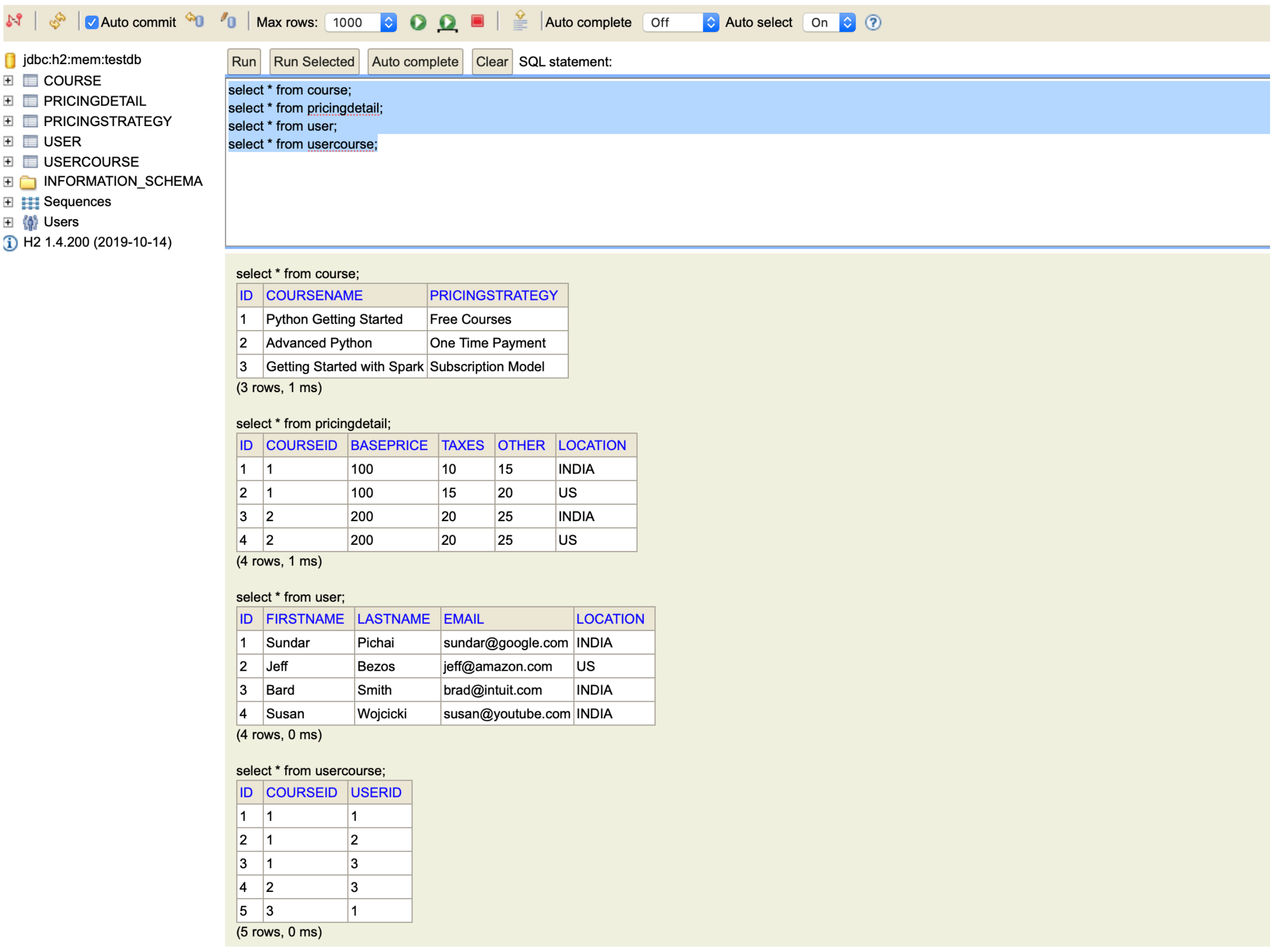
This will generate the H2 database schema as also initialise it with the master data.

The insert scripts can be found in src/resources/data.sql and the schema definition is located in /src/resources/schema.sql

**Accessing the H2 console:**

H2 console can be accessed using localhost:8080/h2

Ensure that the connection URL is set to the following: “jdbc:h2:mem:testdb”



```
select * from course;
select * from pricingdetail;
select * from user;
select * from usercourse;
```

select * from course;

| ID | COURSENAME | PRICINGSTRATEGY |
|---|---|---|
| 1 | Python Getting Started | Free Courses |
| 2 | Advanced Python | One Time Payment |
| 3 | Getting Started with Spark | Subscription Model |

(3 rows, 1 ms)

select * from pricingdetail;

| ID | COURSEID | BASEPRICE | TAXES | OTHER | LOCATION |
|---|---|---|---|---|---|
| 1 | 1 | 100 | 10 | 15 | INDIA |
| 2 | 1 | 100 | 15 | 20 | US |
| 3 | 2 | 200 | 20 | 25 | INDIA |
| 4 | 2 | 200 | 20 | 25 | US |

(4 rows, 1 ms)

select * from user;

| ID | FIRSTNAME | LASTNAME | EMAIL | LOCATION |
|---|---|---|---|---|
| 1 | Sundar | Pichai | sundar@google.com | INDIA |
| 2 | Jeff | Bezos | jeff@amazon.com | US |
| 3 | Bard | Smith | brad@intuit.com | INDIA |
| 4 | Susan | Wojcicki | susan@youtube.com | INDIA |

(4 rows, 0 ms)

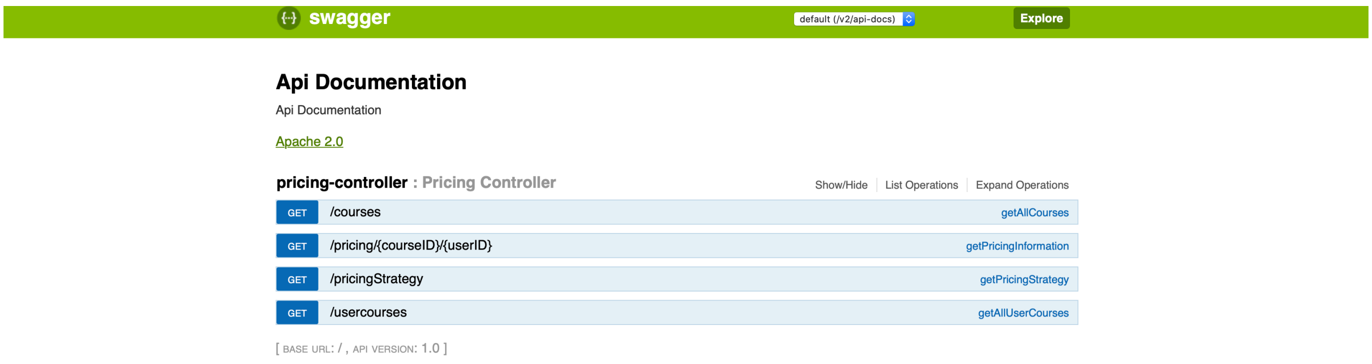
select * from usercourse;

| ID | COURSEID | USERID |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 1 | 2 |
| 3 | 1 | 3 |
| 4 | 2 | 3 |
| 5 | 3 | 1 |

(5 rows, 0 ms)

## Accessing the Swagger documentation:

http://localhost:8080/swagger-ui.html



### Api Documentation

Api Documentation

Apache 2.0

**pricing-controller** : Pricing Controller

| | | |
|---|---|---|
| GET | /courses | getAllCourses |
| GET | /pricing/{courseID}/{userID} | getPricingInformation |
| GET | /pricingStrategy | getPricingStrategy |
| GET | /usercourses | getAllUserCourses |

[ BASE URL: / , API VERSION: 1.0 ]

## Use Cases covered

1. Different users from different locations, shall see different pricing information based on their location

    e.g. http://localhost:8080/pricing/1/2

    http://localhost:8080/pricing/1/1

2. Getting selective fields:

Projections can be used for this scenario to return only the required fields from the repository (https://docs.spring.io/spring-data/rest/docs/current/reference/html/#projections-excerpts)

(could not implement due to time constraint)

## Cache considerations

Mapping of courseid with course name should be maintained in a cache.
Similarly different pricing strategies should also be cached.

## Pending

Flexibility in the API so as to warrant minimum changes for any additional payment components