POLITECNICO DI MILANO

****

DIGITAL INNOVATION LAB 2020-2021

prof. Marco Brambilla

**DELIVERABLE 12**

**IMPLEMENTATION REPORT**

**Group ID**

**10**

**Deliverable Title**

**Implementation Report**

**Team Member**

| **ID** | **Surname** | **Name** |
| --- | --- | --- |
| **10803610** | **Morelli** | **Gabriele** |
| **10581051** | **Dossena** | **Simone** |
| **10632820** | **De Guzman** | **Gian Alessandro** |

***TITLE: DEL.12 - Implementation Report***

***SCENARIO: University Life***

***PROBLEM: Lack of proven and complete study material***

***AUTHORS: Gabriele Morelli, Gian Alessandro De Guzman, Simone Dossena***

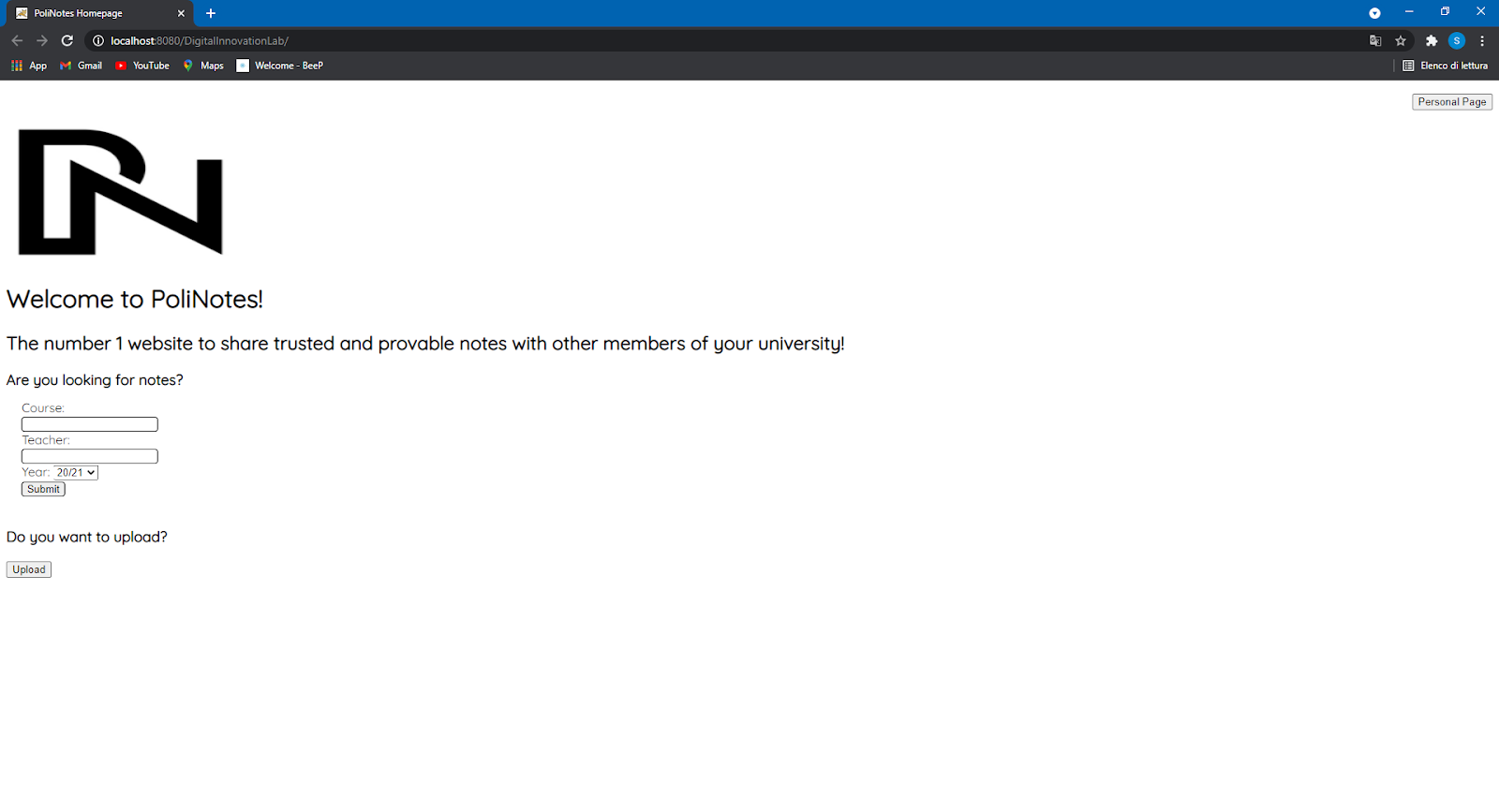
***Application Part***

Regarding the application part, we decided to implement a simplified version of the webapp that represents the prototype of our platform. In particular, we have implemented 2 main features of our platform that can be tested through the prototype:

* Search of notes (that implies the search of the courses);
* Rating and leaving a review of the notes purchased by a given user.

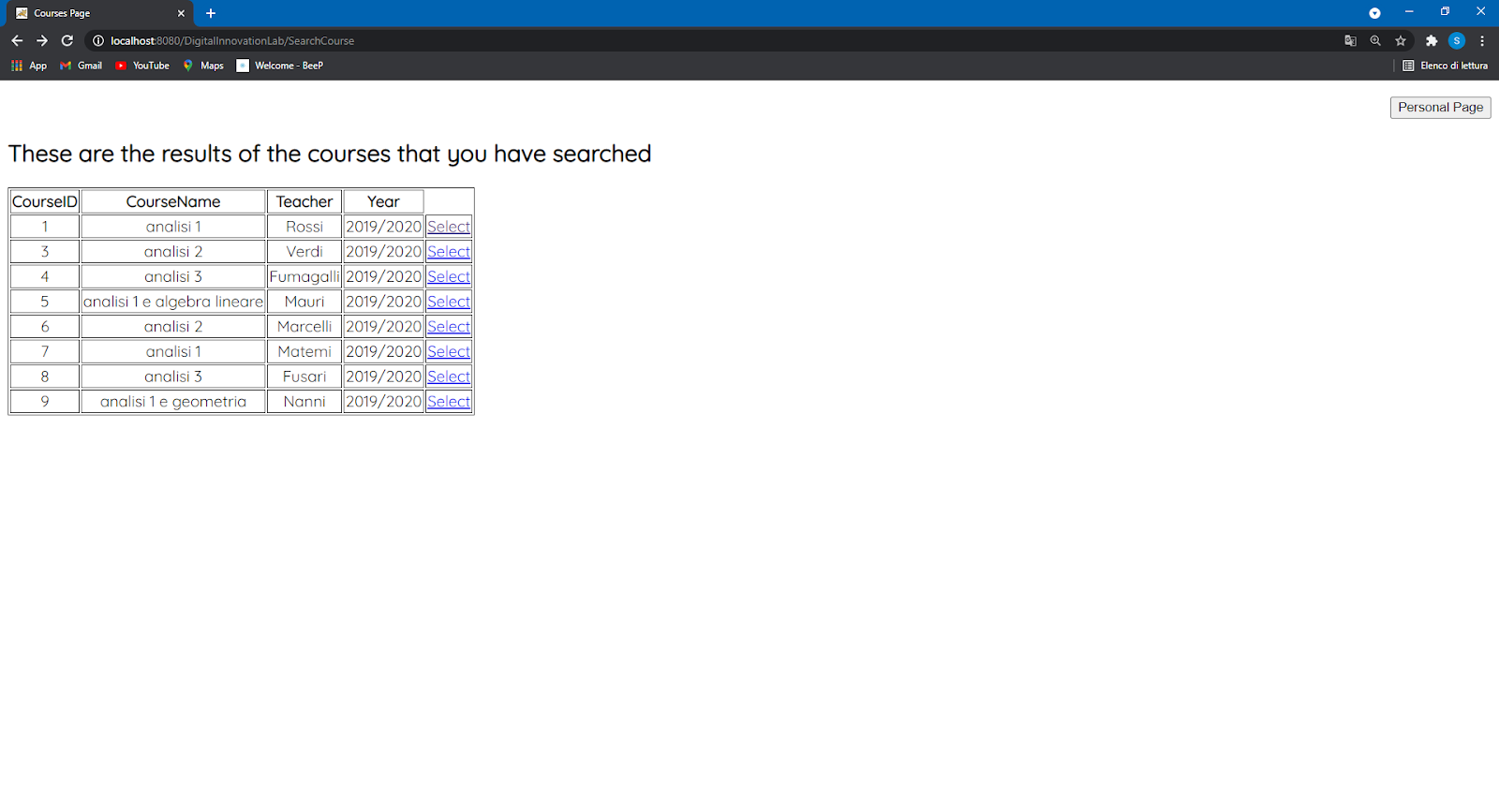
Starting the webapp, we can immediately see the homepage of the webapp. The homepage contains 3 most important elements:

* Form to search the courses that the user is interested in (it is the first phase in the procedure of searching notes)
* Upload button to start the procedure of uploading notes (the following part has not been implemented in this prototype)
* Personal page button to access the personal area of the user (so, we can understand that actually this page in the real system will be displayed only after the login that we decided to not represent because being integrated with the system of Politecnico, the login is not so significant for us to be implemented in a prototype).

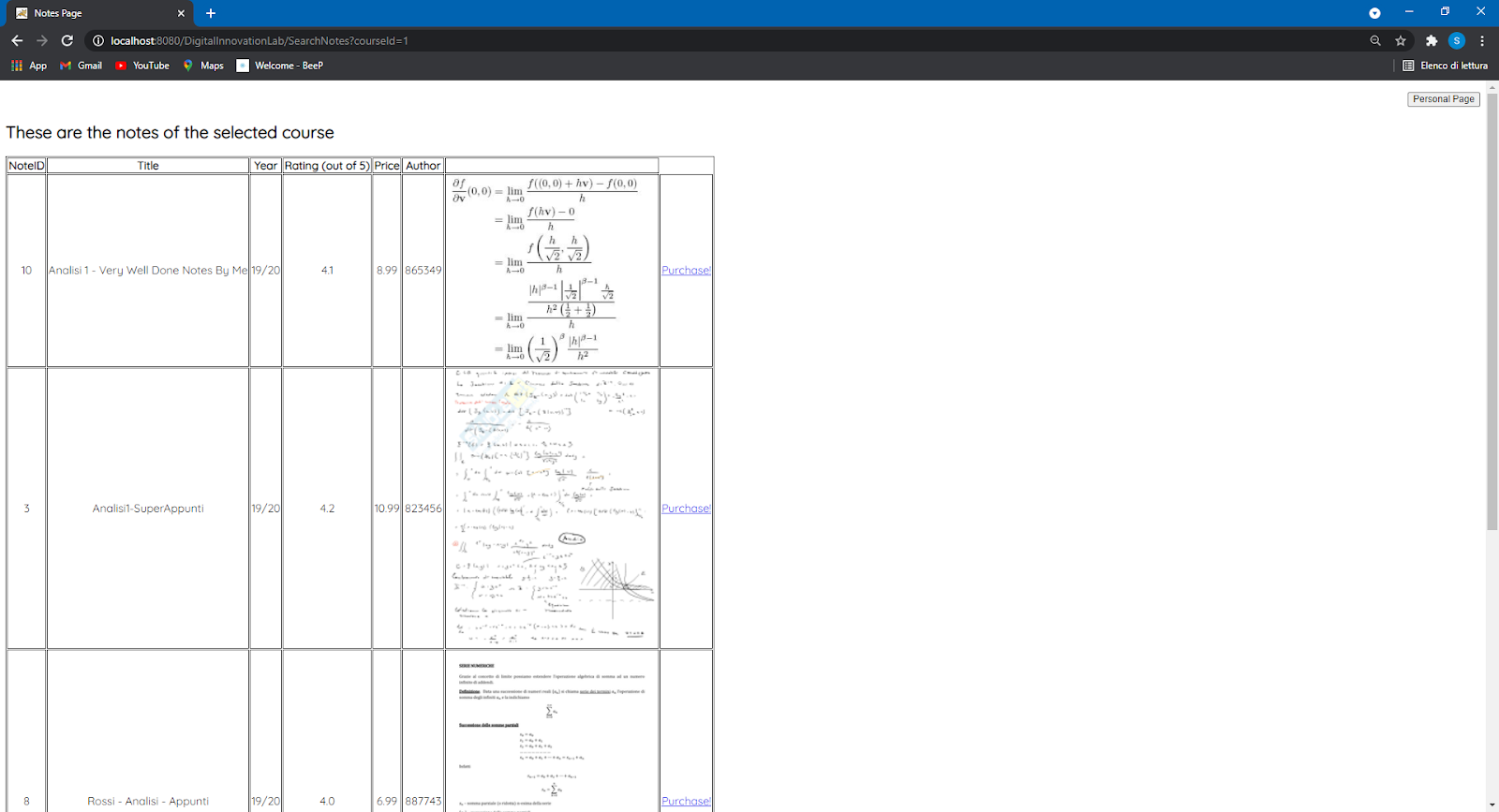


*Search Notes Procedure:*

Filling the form, we can start the search for the desired notes. It is important to mention that actually only the Course and Year field are mandatory to fill to start the procedure. Once, we have filled the form we can see the “Course Page”.  
We can say that the best way to implement the searching of courses could be developing a search engine, especially when the number of courses becomes very high in the system. However, for simplicity in the prototype we have used a simple SQL query with the % operator.

These are the results inserting “analisi” in the Course field and “19/20” as year.

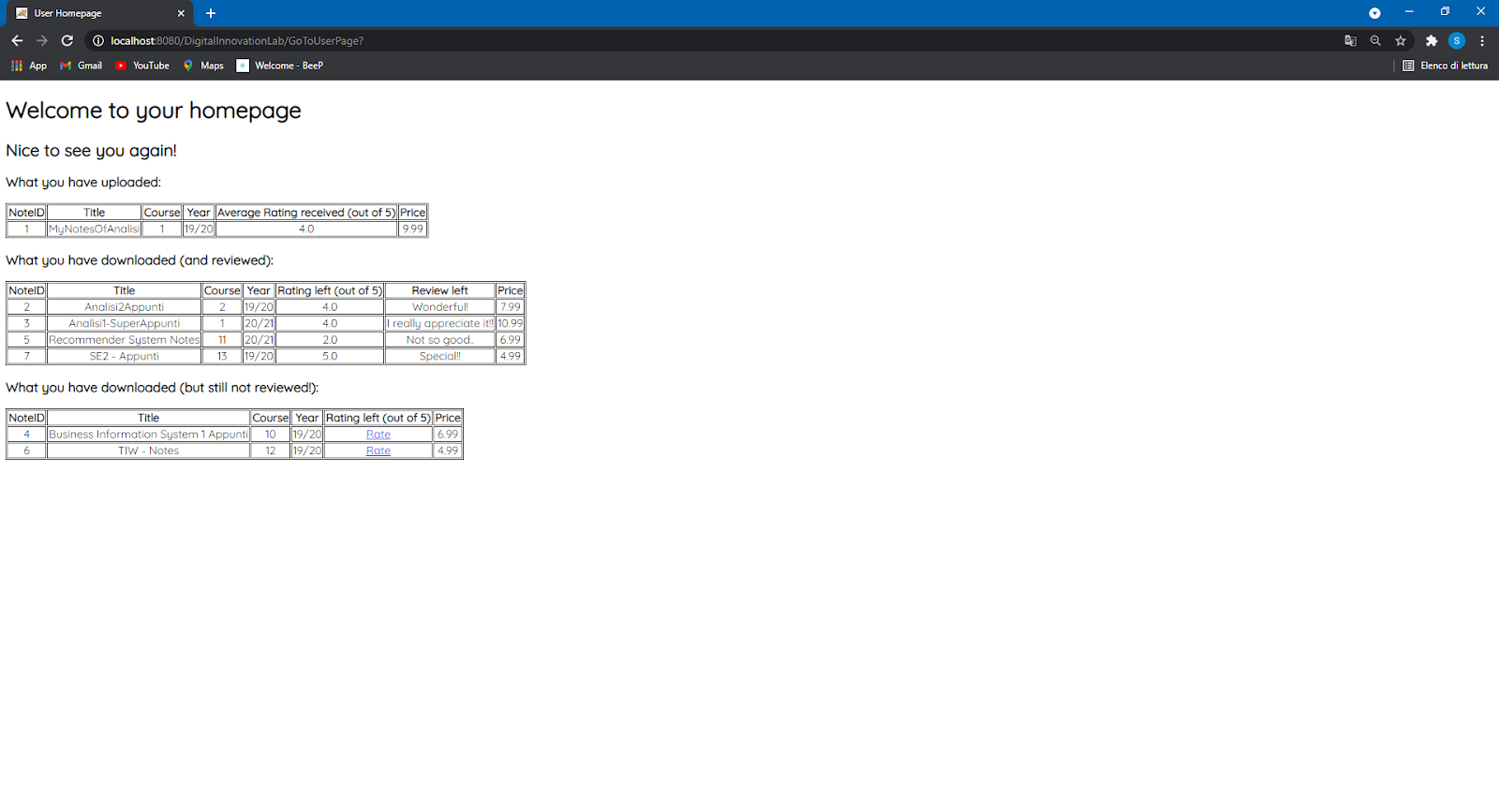
At this point, the user can select the course for which he wants to see the available notes that have been uploaded on the platform.   
For example, clicking on “Select” for the first result it will be displayed the “Notes Page”:



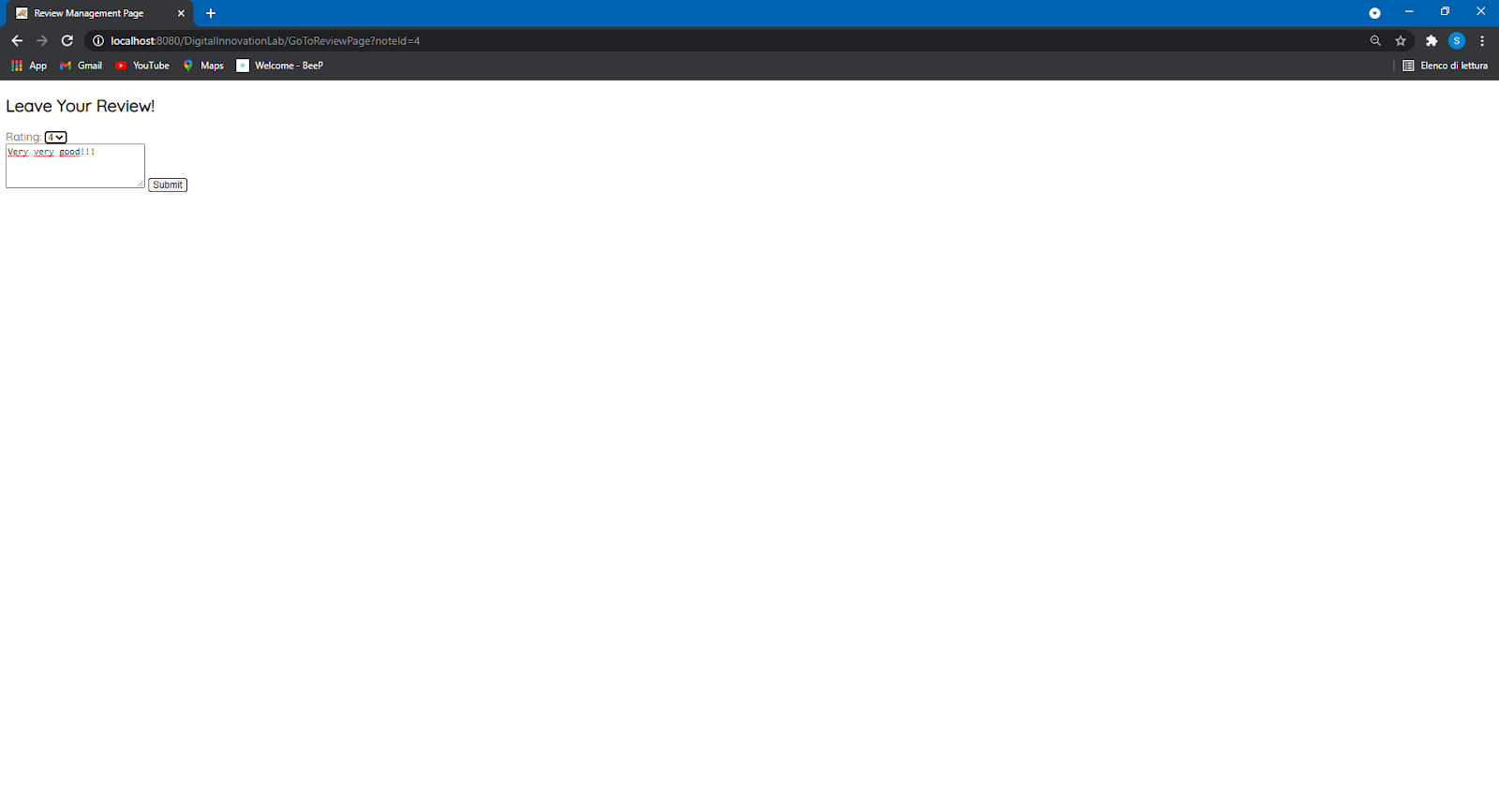
In this page the user can see all the information about the notes uploaded for that course as the title, the year, the uploader, the price, the average rating received and also a preview of the notes. At this point, if the user is convinced by one of these notes he can click on the “Purchase” link that will start the procedure for actually paying the notes after which the user will receive the pdf file of the notes (this procedure has not been implemented).

*Rating and leaving a review*

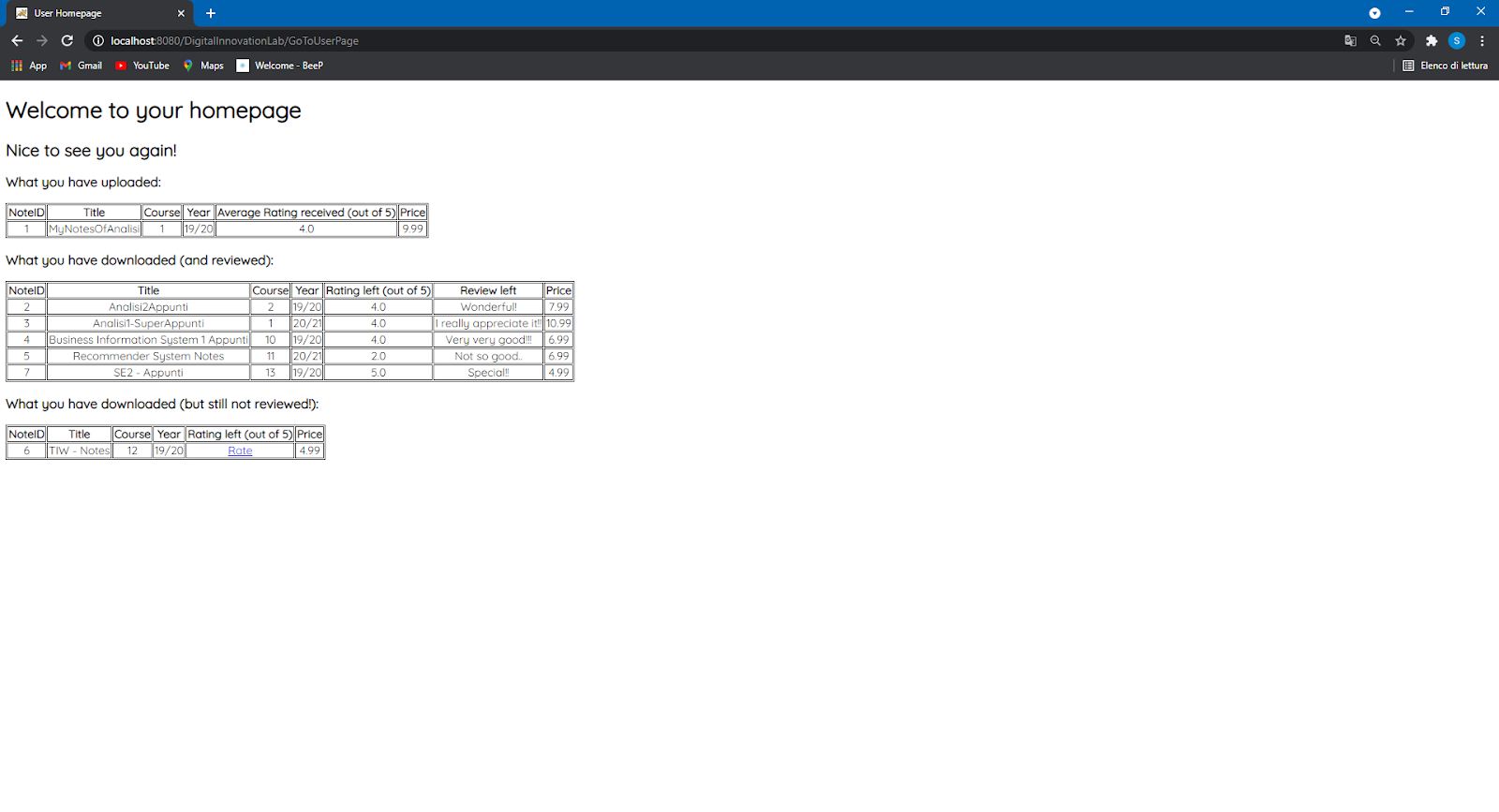
Starting from the homepage of webapp, if the user clicks on the “HomePage” button the personal page of the user will be displayed:



Since we have not implemented the login part, this personal page refers to a sample user.  
In this page, a given user can see the notes that he has previously uploaded together to the average rating received until that moment. Also, he can see the previous downloads that are divided in 2 groups: the download for which the user has already left a rating and a review or the downloads for which he didn’t. We decided to introduce this separation to remark how much is important for our application the fact the users leave a review.   
The unrated notes present the “Rate” link that brings the user to the “ReviewManagementPage”:



In this page, the user can select the rating to give to that note (from 1 to 5) and if he wants, he can also leave a review. Then, clicking on the submit button the user will return to his personal page that has been updated with the rating and the review that he has just left.



Regarding the technical part, the webapp has been developed as an HTML-Pure application. The backend has been implemented using JAVA Language; in particular we develop:

* Bean of the objects as JavaBean:
  + - CourseBean
    - NoteBean
    - UserBean
* Controllers: the servlet
  + - SearchNotes
    - SearchCourse
    - GoToReviewPage
    - GoToUserPage
    - LeaveReview
* Interaction with the database: DAOs
  + - CourseDAO
    - DownloadDAO
    - NoteDAO

The UI has been implemented with HTML Language and we also used the Thymeleaf template engine. The view is composed of 5 HTML Pages:

* homepage.html (the page that is displayed at the beginning)
* CoursePage.html
* NotesPage.html
* UserPage.html
* ReviewManagement.html

We also model the database that is used by the webapp using MySQL workbench. The connection with the database is handled using JDBC.

***Data Part***

We have adopted a type of NoSQL database for our application: the Graph Based Database, to store different types of relationships that, in our context, are very important to execute the main functionalities of the platform. In particular, we’ve used the Neo4J Graph Platform. This choice has been made based on different factors: the high performances it provides thanks to the native graph storage, the scalability and the ACID properties.

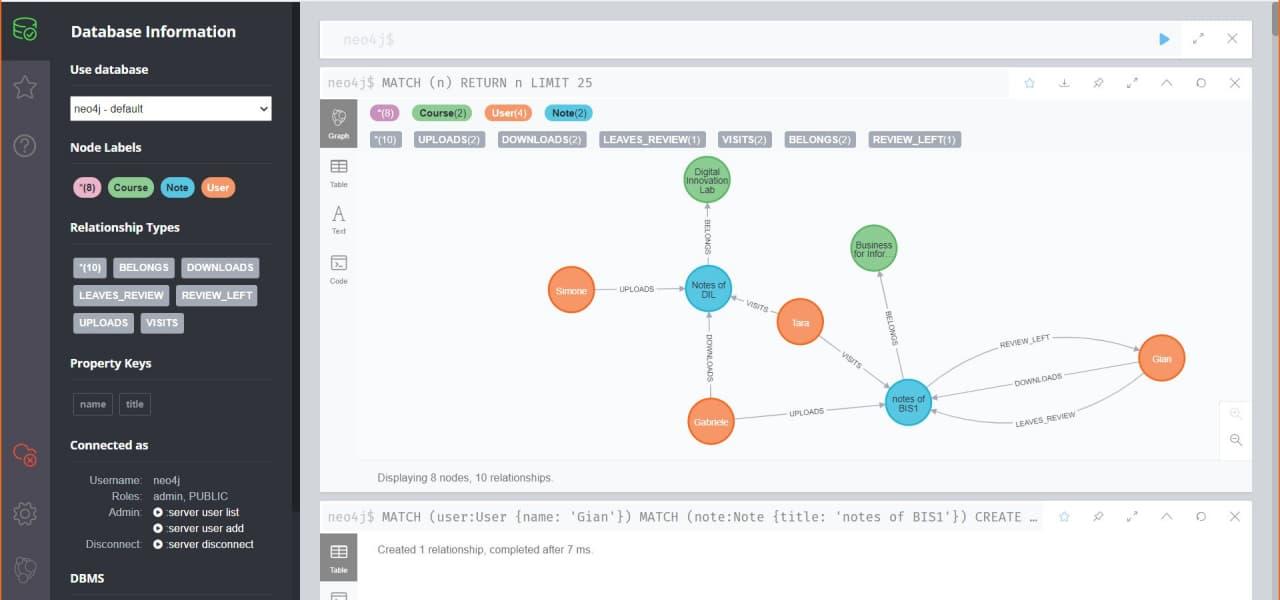
These relationships that link the nodes, represent:

* + User;
  + Note;
  + Course;

while, the type of relationships are:

* “Upload” between User and Note;
* “Download” between User and Note;
* “Visit” between User and Course;
* “Belong” between Note and Course;
* “Reviews” between User and Note;
* “Reviewed” between Note and User.

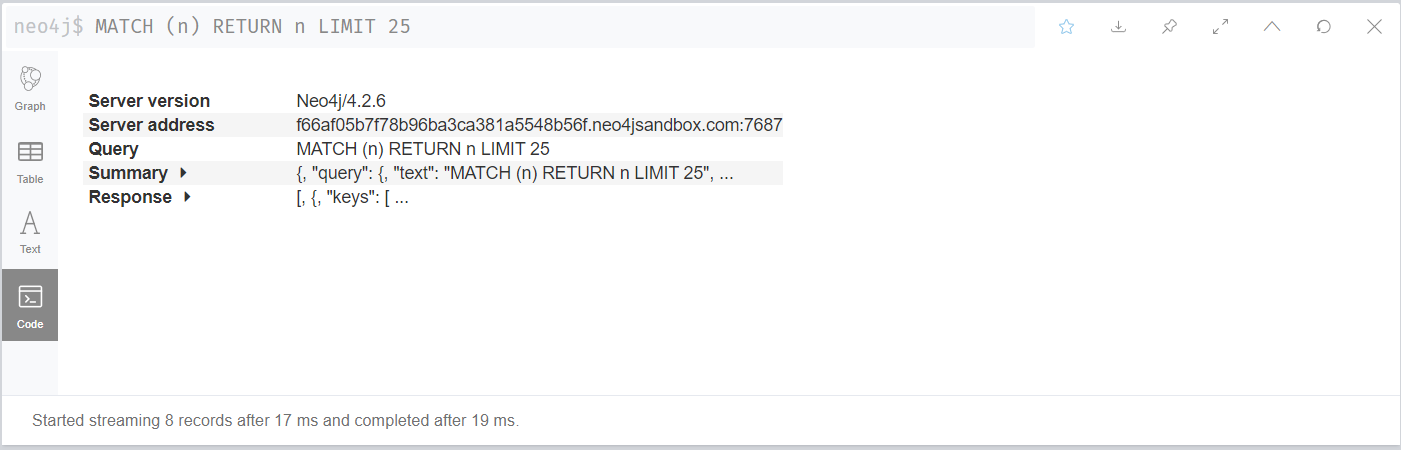
The Image below gives a very concise overview of the data model, highlighting how the nodes are linked between them by the edges.



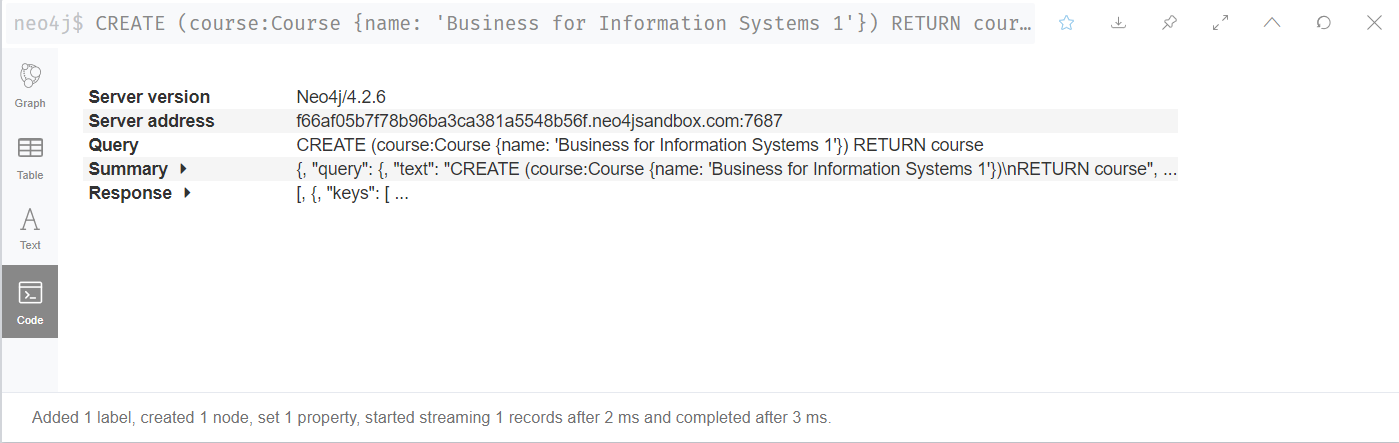
Some very basic Cypher queries for the creation of the graph database follows:

1. The query for obtaining the full database:

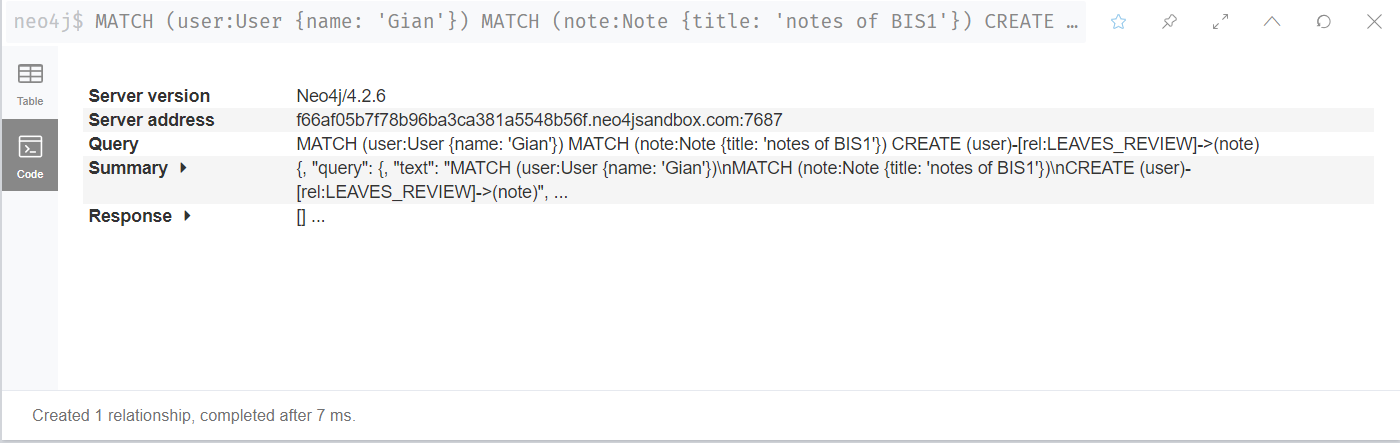
(The LIMIT has been set to 25 for simplicity, since the number of nodes is lower than 25)



1. The query for creating a node:

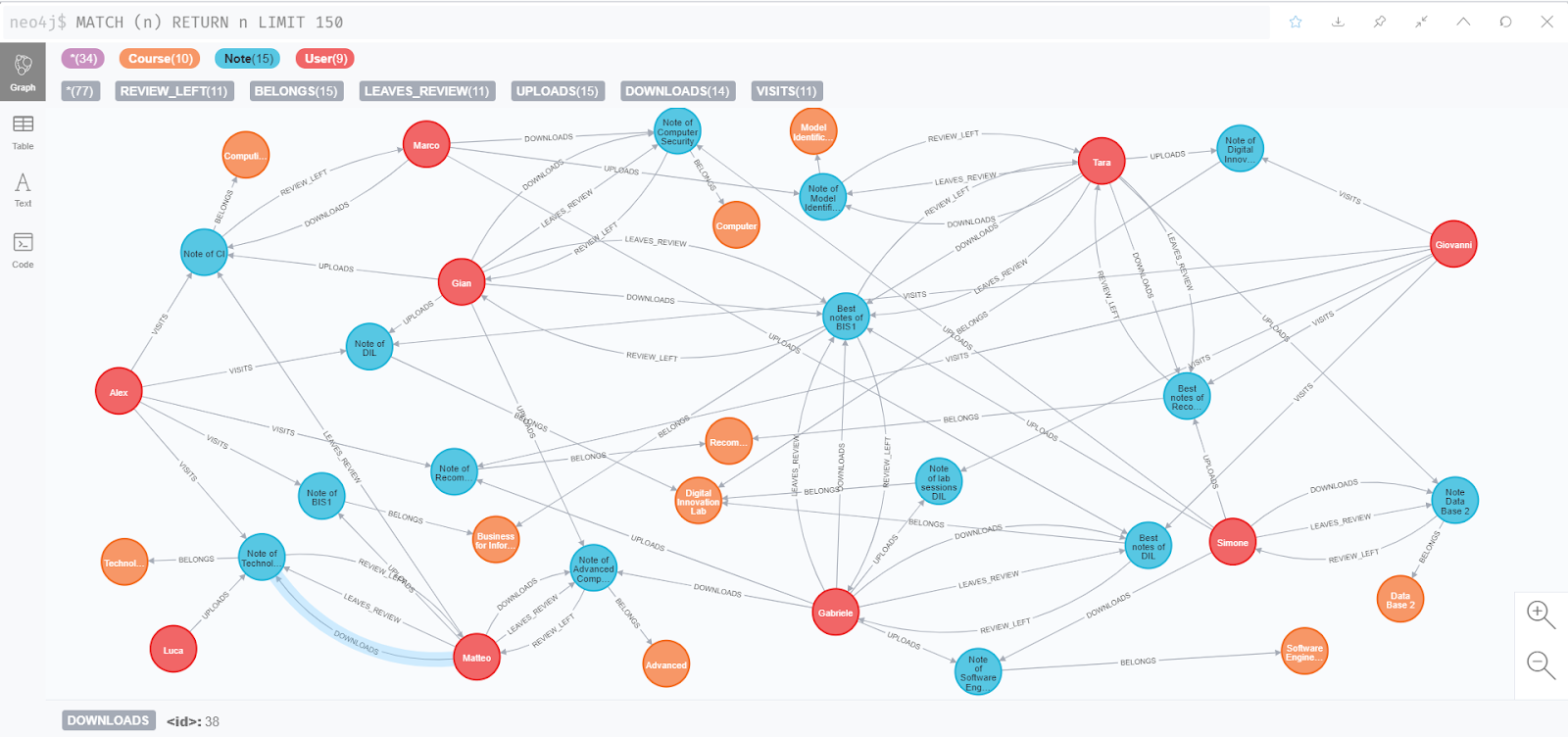


1. The query for creating a relationship between two nodes:



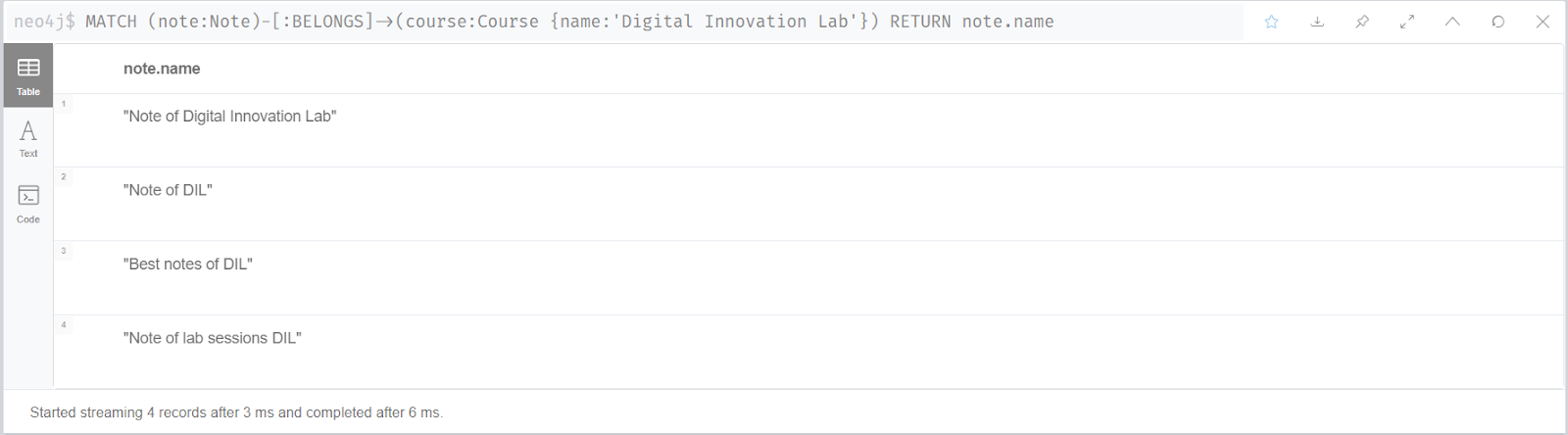
Then, to go further in detail in the implementation of the Graph database, we’ve chosen to implement it by inserting an amount of data which permits us to simulate a real case scenario.

The full database has become:

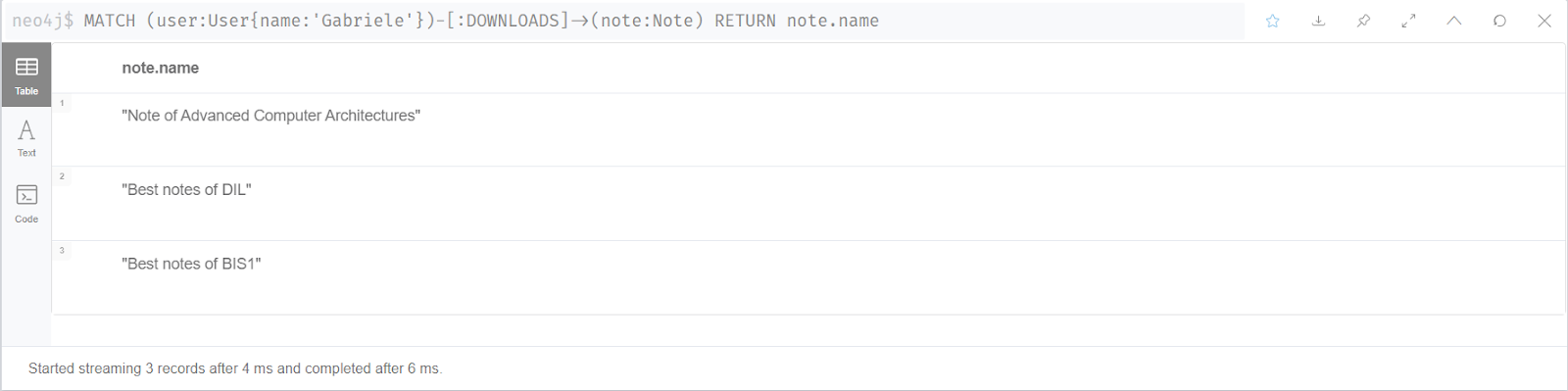


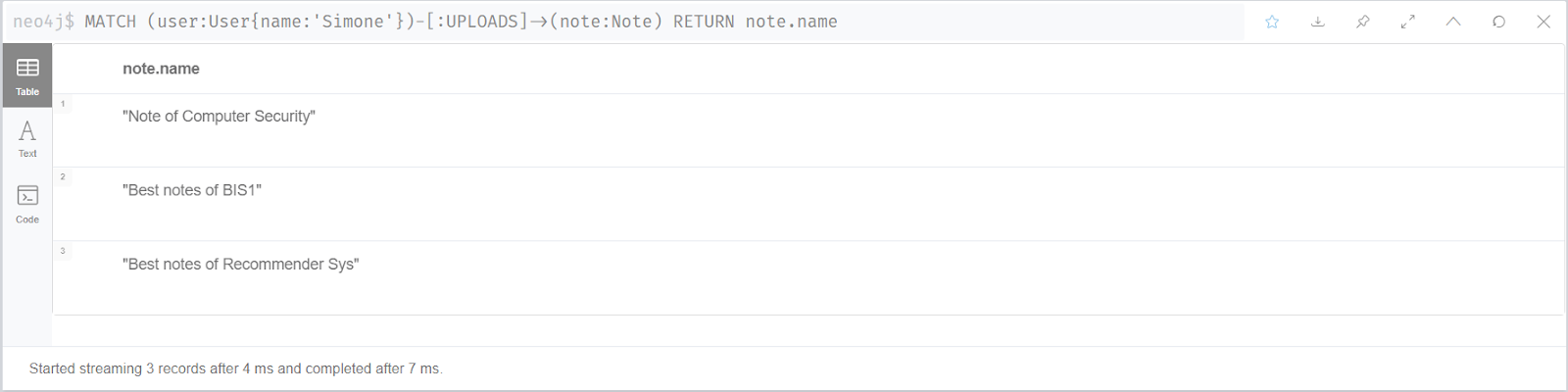
Besides, we’ve written the most significant Cypher queries for our application:

1. This query shows all the names of the “notes” which belongs to a particular “course”:



1. This query shows all the names of the “notes” which have been downloaded by a particular “user”:



1. This query shows all the names of the “notes” which have been uploaded by a particular “user”:
2. This query shows all the names of the “users” who left a review for a particular “note”:



1. This query shows all the names of the “users” that after having downloaded a particular “note”, haven’t left a review.

(The warning in this query is about the fact that the query contains a cartesian product between two disconnected patterns; however, this is intentionally intended for what we want to obtain.)

