

A COMPREHENSIVE APPLICATION ARCHITECTURE FOR UNLOCKING THE POTENTIAL OF STUDENT FEEDBACK

Dana SIMIAN ^{*,1} and Marin-Eusebiu ŞERBAN ²

Dedicated to Professor Radu Păltănea on the occasion of his 70th anniversary

Abstract

This article proposes a novel solution aimed at streamlining the student feedback, enabling them to assess various aspects and activities within their institution. Different from the classical solution of collecting students feedback using a set of questions grouped in one or many forms, we designed a solution capable to empower students of all age groups to enhance their higher education experiences by openly expressing opinions about their enrolled faculties, providing valuable guidance to prospective college students. Our proposal introduces a generic application architecture designed to integrate the presentation of the main opportunities and offers from many universities with the student reviews on all aspects of these universities. Thus, we avoid a number of factors that reduce the quantity and objectivity of student feedback. Important elements in our architecture refer to reviewing and rating system as well as the search system which facilitate the use of the existing evaluations on different aspects of an university. We devised a generic architecture for a client-server application, specifying a minimum set of required functionalities. For validation, we designed and implemented a client-server application based on the generic architecture we outlined. The application offers several key advantages, including robust information security measures, and diverse communication channels for authenticated users within the platform. The application's architecture includes an encryption system based on multiple encryption techniques. Additionally, an algorithm for search optimization within the platform have been developed, further enhancing its utility.

2000 *Mathematics Subject Classification*: 68U35, 68W99.

Key words: client-server application, searching algorithm, student feedback

^{1*} *Corresponding author*, *Lucian Blaga* University of Sibiu, Romania, e-mail: dana.simian@ulbsibiu.ro

²Faculty of Sciences, *Lucian Blaga* University of Sibiu, Romania, e-mail: eusebiu.serban@ulbsibiu.ro

1 Introduction

Enhancing student feedback in higher education is a topic intensively discussed in articles within the field of education sciences. The review [5] and the references therein discuss strategies and tools for effectively collecting and utilizing student feedback. Faculty evaluation presents challenges and best practices, with a particular focus on specific areas such as medical sciences [4], teaching practices [1], and the professional development of teaching staff. Various applications within learning management systems, including but not limited to Blackboard, Canopy Education, Moodle, Campus Lab, and others, often incorporate features for conducting course evaluations and surveys to gather student feedback on courses and instructors. Many universities also employ their own student evaluation of teaching systems or custom evaluation systems to assess faculty performance. Utilizing student feedback is considered one of the best practices in faculty evaluation and development, as highlighted by [13]. Obtaining feedback from a representative and comprehensive sample of students is crucial for improving various activities within universities. Typically, universities collect student feedback through surveys and questionnaires, and this feedback is usually managed centrally at the faculty or university level. It is primarily used to inform and improve the development strategies of these academic institutions and is not typically made publicly available. The objective feedback from students could also be a valuable guide for potential candidates in choosing a university and their educational path.

An empirical study undertaken within a group of students from our university, revealed two aspects that decrease the number of students willing to provide their feedback: the fear that their identity may be revealed independently of their desire and the uncertainty that their feedback will have a considerable effect. To counteract these issues, we propose the use of a rating method of different aspects from universities, including teaching and administrative aspects. This rating module should be encapsulated in a more complex application, and address as many universities as possible.

The open expression of opinions, both positive and negative, as perceived by students, can help reduce their reluctance to share feedback. Additionally, it can have a significant impact on prospective candidates who rely on these reviews for insights. The use of a rating system has demonstrated its effectiveness in other domains, such as tourism and commerce.

Using a rating system based on reviews beside the classical official surveys and questionnaires, aims to enhance the utilization of this feedback for various purposes, including:

- Streamlining and improving the decision-making process for prospective students when selecting a faculty.
- Increasing students' confidence in the impact of their feedback.
- Boosting student participation in providing feedback on university activities and processes.

We are conducting a study to identify potential features for integration into an application aimed at enhancing student engagement in providing open feedback on various aspects of their university experience. Additionally, we are researching technologies suitable for implementing this application. Our findings are validated through the development of an application based on our requirements. The first author of this article was responsible for application implementation.

We also have implemented a solution to optimize the search of information about faculties or universities. The search results will be generated using Levenshtein distances, based on the input text and a set noise level.

Our architecture is conceptually distinct from existing applications that primarily support the learning process and communication between students and teachers. While these applications offer tools for designing didactic materials, conducting surveys, and collecting feedback, our primary objective is to provide information about the services provided by universities (including faculty structures, study programs, activities, accommodation, etc.) and to offer an open evaluation system for students to assess these services and public access to these evaluations. From this perspective, our solution aligns more closely with a social platform than with traditional learning management systems. An intriguing possibility is the integration of our proposed architecture with existing learning management system solutions.

The rest of the article is organized as follows. The minimum requirements and the presentation of the proposed architecture are provided in Section 2. Section 3 presents a case study for our generic solution serving as validation of the proposed generic architecture. A customized searching algorithm is also presented in this section. Section 4 contains conclusions and outlines future directions of study.

2 Proposed architecture

To achieve the objectives outlined in Section 1, we propose the utilization of a client-server application. This application should enable users to conduct searches on a wide range of topics related to universities. Searches can be made by the name of a faculty or university and by a specific field of study, by a specific activity, such as research, accommodation, artistic activities, practice, a specific course, etc. Users can search within existing ratings and comments on these topics and gain access to the official websites of universities with ratings, within the application. The application is required to offer several key functionalities:

- Implementation of a rating system, accompanied by relevant comments that explain the ratings, serving as the primary method for collecting open student feedback.
- A system for approving comments related to ratings, ensuring that the provided reviews meet the standards of decency and appropriateness.
- Establishment of an environment where current students can express their opinions regarding teaching activities and faculty infrastructure.

- Provision of information and student ratings for a substantial number of universities.
- Creation of a communication channel for users logged into the application.
- Implementation of secure user information management to the highest possible degree.
- Customized access levels to application functionalities based on user types.
- User-friendly interface suitable for users of all ages.

These functionalities aim to enhance the quality and relevance of student feedback while also assisting prospective students and interested individuals in finding credible information. The application should have at least the following modules:

- Connection and Authentication Module
- Search
- Chat
- Rating

The design of the database should take into account the normalization operation. The normalization should be balanced with the specific needs of application. The normalization level should assure redundancy reduction, data integrity and simplified maintenance. On the other hand, over-normalization can lead to complex queries and performance issues, so it's essential to strike a balance between normalization and denormalization based on your use case and query patterns, as we will exemplify in Section 3.2. The connection and authentication module categorizes users into two groups: students (authenticated individuals) and visitors, each with different levels of access to application features. Visitors can only view ratings and their associated comments and do not have access to chat functionality. Authenticated users enjoy more capabilities, including integrated messaging, advanced search, access to account settings, and the ability to create reviews. The search module offers different features for the two user roles mentioned above. Visitors can only search for information related to a faculty or university. In contrast, students can search for other students within the same university or locate chat groups. The search diagram is depicted in Figure 1. Considering that each university has its own institutional platform (e.g., Google, Microsoft), student authentication can occur either by directly using institutional credentials or by creating a personal account after authentication via the institutional email address. Only authenticated individuals (students) can submit reviews and ratings. The rating process is associated with the university to which the student belongs. Users can choose between anonymous and public review/rating options.

A communication channel between students is facilitated through chat. Authenticated students can freely communicate within the application, regardless of

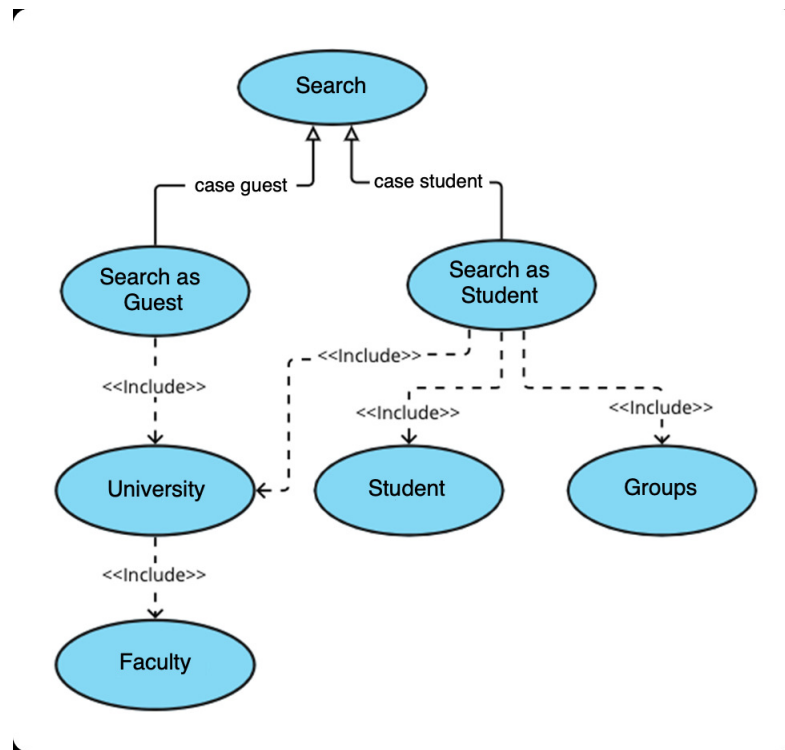


Figure 1: Search method for visitors and students

their university affiliation. An automated verification system for messages transmitted via chat and for reviews is in place, utilizing a dedicated AI module to detect and prevent the use of toxic language.

The information managed within the application is stored in a relational database. We recommend the inclusion of at least the following tables:

- accounts: To store defining user fields, such as university and faculty affiliation, title, and institutional account verification.
- university: To store information about all universities currently accessible within the application, including universities by region, country, or type.
- faculty: To store information about faculties within the universities, including links to faculty websites, review counts, etc.
- rating: To store reviews and ratings associated with specific faculties.

The design of the database should take into account the normalization operation. The normalization should be balanced with the specific needs of application. The normalization level should assure redundancy reduction, data integrity and simplified maintenance. On the other hand, over-normalization can lead to complex queries and performance issues, so it's essential to strike a balance between normalization and denormalization based on your use case and query patterns, as we will exemplify in Section 3.2.

The application must prioritize information security by implementing various cryptographic methods to encrypt messages between users and responses sent from the server to the client. Additionally, a crucial requirement for the application is its compatibility with all currently available platforms.

3 Validation of the proposed architecture. Case study

To evaluate the feasibility of our proposed solution, we have successfully implemented the suggested architecture in a moderately complex client-server environment. The objective of this case study is to demonstrate that the minimum specified requirements of the proposed architecture can be met using moderate computational resources. Additionally, we aimed to gauge the interest of students and prospective students in such applications by providing a test application. The refinement of the proposed architecture within a broader framework also requires practical validation and testing.

This section is dedicated to showcasing the practical implementation of the key functionalities outlined in Section 2 within our case study application. At present, the application exclusively manages information from Romanian universities, but it is designed to be easily expandable to incorporate data from other universities. As previously mentioned, our system employs a client-server architecture/application.

3.1 Client-Side implementation

To evaluate the feasibility of our proposed solution, we have successfully implemented the suggested architecture in a moderately complex client-server environment. The objective of this case study is to demonstrate that the minimum specified requirements of the proposed architecture can be met using moderate computational resources. Additionally, we aimed to gauge the interest of students and prospective students in such applications by providing a test application. The refinement of the proposed architecture within a broader framework also requires practical validation and testing. For the client-side implementation, we utilized the Angular framework to ensure compatibility with all major web browsers. By leveraging the "One Page Application" technology offered by Angular, we minimize the time required to update elements when users navigate the page, as it retains static elements. This approach enhances stability and speed, as described by Kornienko in 2021 [7]. The client part of the application is structured into several components, each of them composing the pages found within the application and keeping various functionalities. This structure enables us to reuse variable and class names across components without encountering conflicts. The key pages currently available in the application include:

- Home page: Provides an overview and introduction.
- Rankings page: Ranks universities or faculties based on their ratings.

- Search page: Displays search results.
- Settings page: Allows users to manage their profiles and settings.

Our application supports login through the Google Institutional Platform, with implementation facilitated by the `angularx-social-login` library. For enhanced security, users can opt for IP authorization, utilizing a validation email sent during the account creation process. The server component has been developed and deployed using Node.js and Express. This library plays a pivotal role in establishing a robust and secure API by leveraging the algorithms specified within the application. The API directly communicates with both the database and the server to effectively respond to user requests.

3.2 Database Implementation

In terms of the database, we employed MySQL, resulting in a database comprising 23 tables, as illustrated in Figure 2, along with their interrelationships. In addition to the tables mentioned in Section 2 (accounts, universities, faculties, rating), we would like to highlight the following:

- `secure_login` table: Manages IP validation if the IP validation login option is chosen.
- `comments` table: An extension of the rating table, storing comments left by other users regarding reviews.
- `distances` table: Contains data on distances between cities in Romania, primarily utilized for recommendations in the search section.
- `chat.keys` table: Stores encryption keys for communication between two users.
- `data_set_training` table: Contains training data for developing a AI recommendation system.

The "accounts" table is directly linked to the majority of the database tables and contains valuable information for identifying users performing activities within specific branches of the application. In the design of our database, we struck a balance between normalization and the specific needs of the search process. Our database is in the second normal form (2NF), ensuring that all non-key attributes are fully functionally dependent on the entire primary key. To enhance the efficiency of information retrieval regarding faculties and universities, we made a deliberate deviation from the requirements of the third normal form (3NF). This decision was driven by the recognition that Boolean searches, a key component of our search algorithm, are not performed efficiently across multiple fields, especially those from linked tables using foreign keys. In line with our query patterns, we have employed a composite field that combines essential faculty and university data (such as name, abbreviation, city, and country) for more effective searching.

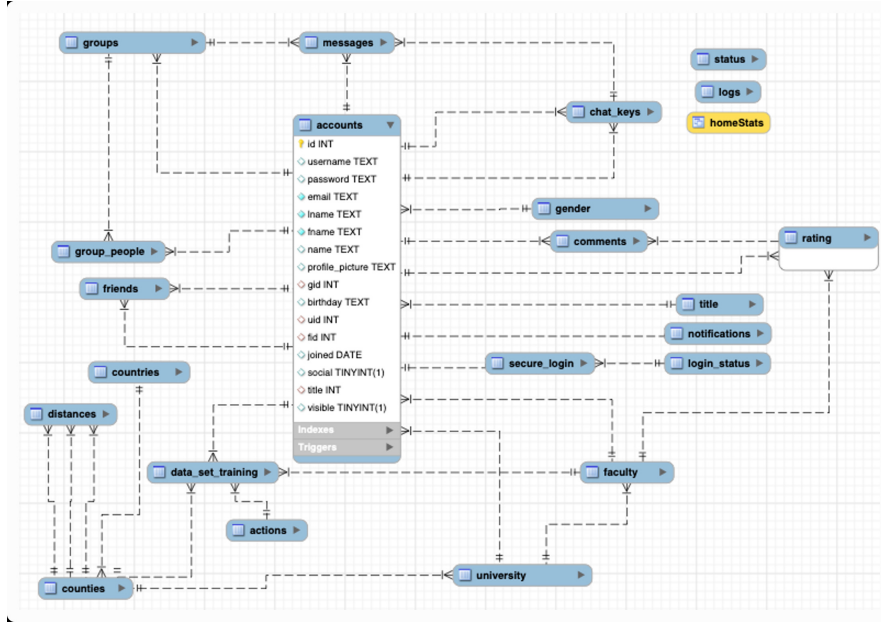


Figure 2: Data base: Tables and interrelationships

3.3 Server – side implementation

In the server-side component, a substantially more intricate architectural framework is employed in contrast to the client-side counterpart. This section serves as the repository for all processes responsible for the interpretation of client requests, functioning as a direct conduit to interface with the underlying database.

The server primarily serves as a conduit for communication with the application's API, encompassing an ensemble of routes, controllers, and models [6]. To ensure the reception of requests directed towards the API, a URL mapping must be established within the application, and this routing procedure is executed prior to the initiation of the HTTPS server. A route is defined within the Express application, crafted by the programmer, and is initialized through the 'use' function, which also configures the characteristics specific to these routes [6]. To ensure the seamless operation of routing, it is imperative to implement the 'cors' structure, which resides within the 'cors' library. This structure rigorously governs the utilization of methods as stipulated by the programmer, delineates the permissible origin of requests, and, of paramount importance, establishes a pre-defined interface that guides the processing of each request. To facilitate the exchange of data between the server and client, two values must be included within the 'Headers' section of the request. The 'bodyParser' library is harnessed to facilitate the processing of requests in JSON format within the dedicated request-handling section. Ultimately, these routes will be operationalized. The establishment of this connectivity employs a router, incorporating the 'router' function from the Express module. This method affords the utilization of techniques for managing various request types.

The server also provides WebSocket capabilities within its structure, aided by the socket.io library, to manage all inputs aimed at receiving and transmitting messages without the need for a request and without necessarily waiting for a response.

The application not only fulfills all the functionalities outlined in Section 2 but goes further by optimizing search operations using cutting-edge technologies. Novel contributions are present in the implementation of security measures for information transfer and in the execution of the search functionality. These contributions will be detailed in the subsequent subsections.

3.4 Security measures

In our system, robust security measures are employed to ensure the confidentiality and integrity of data during transmission between the client and server. These measures encompass various encryption and verification techniques:

Symmetric Encryption: For internal data security within the application, symmetric encryption is employed, relying on a single encryption key. Specific implementations of symmetric encryption include:

- Encryption of messages exchanged between users, with each user possessing a unique encryption key.
- Encryption of server responses sent to the client.

The encryption and decryption techniques based on a single key, implemented in crypto-js library, were used.

Asymmetric Encryption (RSA): Data transmission involves the use of asymmetric encryption based on two RSA keys.

This method ensures that data can be securely transmitted between parties using a pair of public and private keys.

JSON Web Tokens (JWT): Every request is encoded using JSON Web Tokens to provide message integrity and establish a time window during which the token remains valid. The token creation process involves the following steps:

- Segmentation of the token into two parts.
- Encryption of the first segment using asymmetric RSA encryption.
- Utilizing a server-side private key for decryption [11].

The node-forge library is used for RSA encryption and decryption. Tokens must possess a valid signature to reach the server; otherwise, they are rejected as a security measure against potential SQL injection attempts.


Client-Side Token Functionality: We designed and implemented a function for creating tokens on the client side. The principle of implementation is describe below:

- This function accepts content and token validity duration as parameters and generates a valid signature using a programmer-defined key.

- The header and content of the message are encoded in base64 using the CryptoJS library.
- The encoded content's first 10 characters are incorporated into the token.
- The token generation process results in two values: one representing the encoded header and content, and the other representing the previously encrypted token.

3.5 Searching algorithm

We have developed a specific searching algorithm that strikes a balance between simplicity and efficiency. Text manipulation techniques, as detailed in [3], were employed within this algorithm. The following section outlines the systematic approach we undertook to arrive at our final solution. The original algorithm was designed as an extension of the active search mechanism introduced in Figure 3. This method enabled precise phrase searches within specific fields, taking advantage of MySQL's ability to interpret queries in natural language.



```
myUniversity - faculty.model.js
1 SELECT * FROM table WHERE MATCH(col1,col2,col3,...) AGAINST ('search_term' IN NATURAL LANGUAGE MODE)
```

Figure 3: Active search using a key sentence

While this solution seemed ideal at first and was incorporated into the application's framework, it faced challenges in seamlessly integrating with foreign keys. One alternative to overwhelm this difficulty involved implementing the Levenshtein distance algorithm, which required the examination of fields and the interpretation of their contents as numerical representations [9]. For example, when the table contains a field with value "Lucian Blaga," and a user conducted a search for "L. Blaga," the algorithm yielded a distance metric of 6. This metric represented the count of character modifications required to yield the desired outcome, which, in this instance, entailed 5 insertions and 1 deletion. This method had the advantage of making predictions, but does not consider the context, so it lacked the precision necessary to produce accurate results. For instance, a search of a word that does not align with the context of the information in our database, e.g. a search for 'dog' might return totally altered results. To overcome these challenges, we returned to our original approach and designed an algorithm that constructs, starting with the search text, a filter for complex database queries. This filter is based on Levenshtein distance and fuzzy search techniques. Instead of searching across multiple fields in various tables, we search in a composite field containing comprehensive information about faculties and universities, including name, location, abbreviation, and country. The search text is firstly Latinized and then split into distinct words. Our algorithm also includes a word prediction component. Predictions are made using a predefined comparison dataset, encompassing all distinct words within the searching range (information about

universities and faculties) from our dataset. We created this dataset using the minisearch library. To simulate the search context, we use a list of forbidden or "stop" words. "Stop" words are short prepositions or parts of words that should not appear in the context of the search filter we want to generate. We've introduced a text correction mechanism, incorporating fuzzy search and Levenshtein distance, to account for word nuances such as singular or plural forms and to correct the incomplete or erroneous search texts. The output of the algorithm is a complex filter used to generate queries in our database. The steps of the algorithm are described in Algorithm 1.

Algorithm 1 - Generating a Filter for a Search Query

```

INPUT: Search text
         Stop Word List //in our case StopWordList=['si', 'de', 'din', 'la']
         Comparison search dataset
         Noise level
         Composite field from data base
STEP 1: Latinizing the search text
STEP 2: Splitting the search text in distinct words using blanks and punctuation
         marks as delimiters
STEP 3: Generating search filter
         Initializing a blank filter F
         FOR each word in splited text
             //Predicting correct word in the context of search
             L= Generate List of Predicted words using Levenshtein distance and
             Comparison dataset
             // L(1) is the predictor with the best accuracy
             IF L <> empty
                 IF L(1) not in StopWordList
                     //L(1) in StopWordList means that the words from L does
                     not align with the context of search
                     F=F+L(1)
                     // "+" symbolizes concatenating a conjunctive condition
                     to the filter
                 END IF
             ELSE
                 // there is a mistake in Search text. A correction will be sought
                 //Predicting correct suggestion in the context of search
                 L= Generate List of Suggested words using Levenshtein distance,
                 the noise level and Comparison dataset
                 IF L <> empty
                     F=F+L(1)
                 ELSE
                     F=∅
                 END IF

```

```

        END IF
    END FOR
RETURN F

```

STEP 4: Generating a search query in the database

Apply filter F against the composite field.

An example of generating a database search query applying F is given below:

```

SELECT f.id AS fId, f.name AS fName, f.description AS fDesc, f.image AS
fImg, f.website AS fweb, f.reviews AS fReviews, f.rating AS fRating, f.url AS
fURL, u.id AS uId, u.name AS uName, u.academy AS Academy, u.private AS
uPrivate, u.rating AS uRating, u.reviews AS uReviews, u.url AS uURL, c.id
AS cId, c.county AS Name, c.region AS Code, ct.id AS ctId, ct.name AS ct-
Name, ct.countryCode AS ctCode FROM faculty f INNER JOIN university u
ON f.university = u.id INNER JOIN counties c ON c.id = u.city INNER
JOIN countries ct ON ct.id = c.country WHERE MATCH(f.description)
AGAINST (" ? " IN Boolean mode)

```

A result of search is illustrated in Figure 4.

```

myUniversity - faculty.model.js
1  const { dataset } = require('./utils/predictions.service')
2  const MiniSearch = require('minisearch')
3  let stopWords = new Set(['si', 'de', 'din', 'la'])
4  var Conn = require('./config/db.config');
5  var latinize = require('latinize');
6
7  let predictor = new MiniSearch({
8    fields: ['name'],
9    storeFields: 'name',
10 })
11 predictor.addAll(dataset);
12
13 Faculty.getByFilter = (filter, result) => {
14   filter = latinize(filter);
15   var newFilter = "";
16   filter.split(/[s,]+/).forEach(word => {
17     if (word != '') {
18       var prediction = predictor.autoSuggest(word, { prefix: true })
19       if (prediction.length > 0) {
20         if (!stopWords.has(prediction[0].suggestion))
21           newFilter += '+' + prediction[0].suggestion + ' ';
22       } else {
23         prediction = predictor.autoSuggest(word, { prefix: true, fuzzy: 0.7 })
24         if (!stopWords.has(prediction[0].suggestion))
25           newFilter += '+' + prediction[0].suggestion + ' ';
26       }
27     }
28   })
29   Conn.then((connection) => {
30     connection.query('SELECT f.id AS fId, f.name as fName, f.description as fDesc, f.image AS fImg, f.website AS fWeb, f.re
31     views AS fReviews, f.rating AS fRating, f.url AS fURL, u.id AS uId, u.name AS uName, u.academy AS uAcademy, u.private AS uPriva
32     te, u.rating AS uRating, u.reviews AS uReviews, u.url AS uURL, c.id AS cId, c.county AS cName, c.region AS cCode, ct.id AS ctI
33     d, ct.name AS ctName, ct.countryCode AS ctCode FROM faculty f INNER JOIN university u ON f.university = u.id INNER JOIN countie
34     s c ON c.id = u.city INNER JOIN countries ct ON ct.id = c.country WHERE MATCH(f.description) AGAINST (" ? " IN Boolean mode)',
35     newFilter, (err, res) => {
36       if (err) result(err, null);
37       else result(null, res);
38     })
39   })
40 }

```

Figure 4: Example of search results with an incomplet input search text

4 Conclusions and further directions of study

In this research article, we suggest using a client-server architecture to gather student feedback about various aspects of a university in a less formal way. Instead of relying on questionnaires or surveys, we propose using open reviews and a rating system. Our solution makes use of multiple communication channels. To validate our proposal, we've implemented a client-server application based on this architectural design, and we provide comprehensive implementation details in this article. We've also developed an algorithm to improve search functionality within the application. The application's powerful search feature ensures that ratings and comments are easily accessible, making it a valuable tool for prospective students seeking recommendations. As future directions of study, we intend to assess how effective this architectural approach is in improving the quality of student feedback and instilling greater confidence in students about the importance of their feedback contributions.

References

- [1] Asgar, A., Satyanarayana, R., *An evaluation of faculty development programme on the design and development of self-learning materials for open distance learning*, Asian Association of Open Universities Journal **16** (2021), no. 1.
- [2] Dujeepa, S., Yeo, P. and Tan, H., *Obtaining impactful feedback from students: a continuous quality improvement approach to enhance the quality of students' feedback*, South East Asian Journal of Medical Education, **8**, (2014), 2-9.
- [3] Enge, E., Spencer, S. and Stricchiola, J.C., *The art of SEO: mastering search engine optimization*, O'Reilly Media, 2015.
- [4] Fernandez, N. and Audétat, M.C., *Faculty development program evaluation: a need to embrace complexity*, Adv. Med. Educ. Pract. **10** (2019), 191–199.
- [5] Haughney, K., Wakeman, S. and Hart, L., *Quality of feedback in higher education: a review of literature*, Educ. Sci., **10**, (2020), no. 3.
- [6] Holmes, S., *Getting MEAN with Mongo, Express, Angular and Node*, Manning, 2019.
- [7] Kornienko, D.V., Mishina, S.V. and Melnikov, M.O. *he Single page application architecture when developing secure Web services*, Journal of Physics: Conference Series, **2091**, (2021).
- [8] Lombardi, A., *WebSocket: lightweight client-server communications*, O'Reilly Media, Inc, USA, 2015.
- [9] Miller, F.P., Vandome, A.F. and McBrewster, J., *Levenshtein distance*, VDM Publishing, 2009.
- [10] Murray, N., Coury, F., Lerner, A. and Taborda, C., *The complete book on Angular 5*, Create Space Independent Publishing Platform, 2018.
- [11] Paar, C. and Pelzl, J., *Understanding cryptography*, Springer-Verlag, Berlin Heidelberg, 2010.
- [12] Shwartz, B., Zaitsev, P. and Tkachenko, V., *High performance MySQL*, O'Reilly Media, 2012.
- [13] Yu, S.O., *Using students' feedback to evaluate teachers' effectiveness*, Journal for Educators, Teachers and Trainers, **7**, (2016), no. 1, 182 – 192.