Cairo University
Faculty of Engineering
Department of Computer Engineering

ERevive

A Graduation Project Report Submitted

to

Faculty of Engineering, Cairo University

in Partial Fulfillment of the requirements of the degree

of

Bachelor of Science in Computer Engineering.

Presented by

Menna Mahmoud Salah
Nihal Mansour Abd El-Bari
Nada Abdelmaboud Abdelrazik
Hager Ismail Abdelaty ali

Supervised by

Dr.Khaled Soradi

17 July. 2022

Abstract

Rapid Prototyping is an approach that aims to build a system that uses a certain technology or platform as quickly as possible and to test this system whether it satisfies the requirements or not, but it suffers from different problems like the additional costs needed as a result of the technologies used and faster turnaround required in building systems.

ERevive addresses some of the problems in rapid prototyping like wasted resources by supporting fast indication of system requirements, bridging gaps between users and developers in understanding systems requirements and saving much more time and effort in creating such systems.

ERevive helps in rapid prototyping of software systems that require dealing with structured query languages (SQL) by taking an entity relationship diagram (ERD) of the required system then using Python we apply some preprocessing techniques on this image that will result in extracting some information about the system like detecting existed entities, attributes, relations and connectivity between them, moreover detecting participation and cardinalities then generating initial schema which will require the developer to perform some editing to guarantee that everything detected is correct and to generate the final schema that will proceed to the search engine module that will search for relevant queries with in huge number of datasets then perform some mapping and transformations from the returned queries to the system final schema and order them regarding how much they are relevant to the required system then it will proceed in building a fast automated web dashboard in Vue that is connected to a rest API in Flask-RESTX to allow the developer to interact with the system, moreover it allows the user to write some natural language sentences written in English indicating his/her required SQL queries in the system that will be then transformed to SQL queries in which developer can also add to the system.

الملخص

النماذج الأولية السريعة هي نهج يهدف إلى بناء نظام يستخدم تقنية أو منصة معينة في أسرع وقت ممكن واختبار هذا النظام سواء كان يلبي المتطلبات أم لا ، ولكنه يعاني من مشاكل مختلفة مثل التكاليف الإضافية المطلوبة نتيجة التقنيات المستخدمة وسرعة التحول المطلوبة في أنظمة البناء.
يعالج هذا المشروع بعض المشكلات في النماذج الأولية السريعة مثل الموارد المهدرة من خلال دعم الإشارة السريعة لمتطلبات النظام ، وسد الفجوات بين المستخدمين والمطورين في فهم متطلبات الأنظمة وتوفير المزيد من الوقت والجهد في إنشاء مثل هذه الأنظمة.
يساعد هذا المشروع في النماذج الأولية السريعة لأنظمة البرامج التي تتطلب التعامل مع لغات الاستعلام المهيكلة (SQL) من خلال أخذ مخطط علاقة كيان النظام المطلوب ثم تطبيق بعض تقنيات المعالجة المسبقة على هذه الصورة التي ستؤدي إلى استخراج بعض المعلومات حول النظام مثل اكتشاف الكيانات الموجودة والسمات والعلاقات والاتصال بينها ، علاوة على اكتشاف المشاركة والأصول ثم إنشاء مخطط أولي يتطلب من المطور إجراء بعض التغييرات لضمان صحة كل ما تم اكتشافه وإنشاء المخطط النهائي الذي سينتقل إلى وحدة محرك البحث التي ستقوم بالبحث عن الاستعلامات ذات الصلة بعدد كبير من مجموعات البيانات ، ثم تقوم بإجراء بعض التعيين والتحويلات من الاستعلامات التي تم إرجاعها إلى المخطط النهائي للنظام وطلبها فيما يتعلق بمدى ارتباطها بالنظام المطلوب ، ثم ستستمر في بناء لوحة تحكم ويب مؤتمتة سريعة متصلة بقاعدة بيانات للسماح للمطور بالتفاعل مع النظام ، علاوة على أنه يسمح للمستخدم بكتابة بعض الجمل المكتوبة باللغة الإنجليزية تشير إلى استعلامات SQL المطلوبة في النظام والتي سيتم تحويلها بعد ذلك إلى استعلامات SQL والتي يمكن للمطور أن يضيفها أيضًا إلى النظام .

ACKNOWLEDGMENT

We would like to thank Dr. Khaled Soradi for his patience and great support in suggesting new ideas for solving problems that faced us throughout the whole process of the project and encouraged us to do our best to develop a complete featured project.

We would like to give great thanks to all professors that have the credit after Allah in what we are today and in our progress in the technical field of software development.

Table of Contents

Abstract iii
الملخص iv
Acknowledgment v
Table of Contents vi
List of Figures ix
List of Tables x
List of Abbreviations xi
Contacts xiii
Chapter 1: Introduction
1.1. Motivation and Justification 1
1.2. Project Objectives and Problem Definition 1
1.3. Project Outcomes 2
1.4. Document Organization 2

Chapter 2: Visibility Study
2.1. Target Customers 3
2.2. Market Survey 3
2.2.1. UML to SQL Schema 3
2.2.2. Also Asked 4
2.2.3. Natural SQL 4
2.2.4. Github Copilot 5
2.3 Business Case and Financial Analysis 5

Chapter 3: Literature Survey
3.1. Image Processing 10
3.1.1. Thresholding 10
3.1.1.1. Simple Thresholding 11
3.1.1.2. Isodata Algorithm 11
3.1.1.3. Otsu Thresholding 12
3.1.2. Convex Hull 13
3.1.3. Geometric Shapes Detection 13
3.1.3.1. Statistical Approach 14
3.2 Search Engine 15
3.2.1. Crawling 15
3.2.2. Indexing 15
3.2.3. Search Results Ranking 16
3.3. Implemented Approach 16
3.3.1. Image Preprocessing 16
3.3.2. Search Engine 17

Chapter 4: System Design and Architecture
4.1. Overview and Assumptions 19
4.2. System Architecture 20

4.2.1. Block Diagram 20
4.3. Image Processing 20
4.3.1. Functional Description 20
4.3.2. Modular Decomposition 21
4.3.2.1. Image Preprocessing 21
4.3.2.2. Segmentation 21
4.3.3. Design Constraints 22
4.4. Schema Information Extraction 22
4.4.1. Functional Description 22
4.4.2. Modular Decomposition 22
4.4.2.1. Shapes Classification 22
4.4.2.2. Text Detection 23
4.4.2.3. Datatypes Detection 23
4.4.2.3.1. Dataset Collection 23
4.4.2.3.2. Datatypes Model 25
4.4.3. Design Constraints 25
4.5. Schema Generation 25
4.5.1. Functional Description 25
4.5.2. Modular Decomposition 26
4.5.2.1. Participation Detection 26
4.5.2.2. Cardinality Detection 26
4.5.2.3. Connect Components 27
4.5.3. Design Constraints 28
4.6. Search Engine 28
4.6.1. Functional Description 28
4.6.2. Modular Decomposition 28
4.6.2.1. Dataset Collection and Parser 28
4.6.2.2. Indexer 29
4.6.2.3. Mapper and Joiner 29
4.6.2.4. Clustering and Ranking 29
4.6.2.5. Query Construction 31
4.6.3. Design Constraints 31
4.7. Natural Language to SQL 31
4.7.1. Functional Description 31
4.7.2. Modular Decomposition 31
4.7.3. Design Constraints 32
4.8. API and UI generation 33
4.8.1. Functional Description 33
4.8.2. Modular Decomposition 33
4.8.2.1. API Generation 33
4.8.2.2. Database Seeds Generation 34
4.8.2.3. UI generation 35

Chapter 5: System Testing and Verification

5.1. Testing Setup 37

5.2. Testing Plan and Strategy.....37
5.2.1. Module Testing 39
5.2.1.1. Image Processing Testing 39
5.2.1.1.1. Customer Relationship Management System 39
5.2.1.1.2. Schoology Learning Management System 42
5.2.1.1.3. Company System 45
5.2.1.2. Search Engine 48
5.2.1.2.1. Schoology Learning Management System 48
5.2.1.2.2. Airport System 54
5.2.2. Integration Testing 56
5.3. Test Schedule.....56
5.4. Comparative Results to previous work 56

Chapter 6: Conclusions and Future Work
6.1. Faced Challenges 57
6.2. Gained Experience 58
6.3. Conclusions 58
6.4. Future Work 59
References 60

Appendix A: Development Platforms and Tools 62
A.1. Software Tools 62
A.1.1. Programming Languages 62
A.1.2. Libraries and Framework 62
A.1.3. Tools and Platforms 62

Appendix B: User Guide 63

Appendix C: Feasibility Study 70
C.1. Product Description 70
C.2. Technical Feasibility 70
C.3. Product Market Place 71
C.4. Financial Feasibility 71
C.5. Legal Feasibility 71

List of Figures

Figure 3.1: Thresholding. On the left is the RGB image and on the right is the binarized image ……11
Figure 3.2: Isodata Thresholding. On the left is a grayscale image that has the objects with white and the background black. On the left is the thresholded image……....12
Figure 3.3: Otsu Thresholding……12
Figure 3.4: Convex Hull (the gray polygon).....13
Figure 3.5: pipeline……14
Figure 3.6: some classified shapes……....14
Figure 4.1: System modules design……..…20
Figure 4.2: Final dataset statistics……..24
Figure 4.3: Welcome page of generated UI……..36
Figure 4.4: Get request dashboard in the generated UI……..36
Figure 4.5: Post and put requests form in the generated UI……..37
Figure 5.1: ER diagram for FreshDesk CRM system……..39
Figure 5.2: EREVIVE Image Processing Output For FreshDesk ER……41
Figure 5.3: ER diagram for Schoology LMS system……....42
Figure 5.4: EREVIVE Image Processing Output For Schoology ER……...44
Figure 5.5: ER diagram for company system……45
Figure 5.6: ER diagram for Schoology LMS system……..……47
Figure 5.7: ER diagram for airport system……..…54
Figure B.1 : Home page……..…63
Figure B.2 : System information page……..…..63
Figure B.3 : Upload image page……....64
Figure B.4 : Image processing output page……..64
Figure B.5 : continue Image processing output page……..…65
Figure B.6 : Validation page……65
Figure B.7 : Clusters page……...66
Figure B.8 : Queries for certain cluster page……..…66
Figure B.9 : Editing and deleting certain query within a cluster……66
Figure B.10 : NL to SQL page……..……67
Figure B.11: last page……...……..67
Figure B.12 : welcome page in generated UI……..…68
Figure B.13: Showing queries for a certain cluster in generated UI……..…68
Figure B.14: post and put in the generated UI……..……69
Figure B.15: get dashboard in the generated UI……..…69

List of Tables

Table 2.1: Different available modes from ERevive……..……6
Table 2.2: Business case in ERevive……6
Table 2.3: Capex and opex……..……8
Table 2.4: Cash flow table over a year……9
Table 5.1: Detected shapes in FreshDesk……40
Table 5.2: Detected primary keys in FreshDesk……...40
Table 5.3: Detected relations in FreshDesk……..40
Table 5.4: Detected text in FreshDesk……...…40
Table 5.5: Detected shapes in Schoology……....43
Table 5.6: Detected primary keys in Schoology……..…43
Table 5.7: Detected relations in Schoology……...……43
Table 5.8: Detected text in Schoology……43
Table 5.9: Detected shapes in company……....46
Table 5.10: Detected primary keys in company……46
Table 5.11: Detected relations in company……..…46
Table 5.12: Detected text in company……...…46
Table 5.13: Generated queries for School entity……..48
Table 5.14: Generated queries for Building entity……48
Table 5.15: Generated queries for User entity……..49
Table 5.16: Generated queries for Grade entity……..…49
Table 5.17: Generated queries for Grade, Class section, Assignment entities……...…50
Table 5.18: Generated queries for Course, User, Building entities……...50
Table 5.19: Generated queries for Assignment entity……...……..50
Table 5.20: Generated queries forAssignment, Course, Course section entities……51
Table 5.21: Generated queries for Document entity……51
Table 5.22: Generated queries for Enrollment entity……...52
Table 5.23: Generated queries for Events entity……..…52
Table 5.24: Generated queries for Blog entity……..52
Table 5.25: Generated queries for Blog comments entities……....…53
Table 5.26: Generated queries for User, Blog, Blog comments entities……...……53
Table 5.27: Generated queries for User, Blog entities……..……53
Table 5.28: Generated queries for Airport entity……...……...……54
Table 5.29: Generated queries for Airport entity……..……55

List of Abbreviation

API	Application Programming Interface
UI	User Interface
CapEx	Capital Expenditure
CNN	Convolutional Neural Network
CRUD	Create Retrieve Update Delete
DB	Database
ERD	Entity Relationship Diagram
HOG	Histogram of Oriented Gradients
NL	Natural Language
NLIDBs	Natural Language Interface to Database
NLTK	Natural Language Tool Kit
NN	Neural Networks
OCR	Optical Character Recognition
OpEx	Operating Expenditure
POS	Parts of Speech
REST API	Representational State Transfer Application Programming Interface
SIFT	Scale Invariant Feature Transform
SQL	Structured Query Languages
SVM	Support Vector Machine
UML	Unified Modeling Language

Contacts

Team Members

Name	Email	Phone Number
Menna Mahmoud Salah	menna123mahmoud@gmail.com	+2 01066761053
Nada Abdelmaboud Abdelrazik	nada5aled52@gmail.com	+2 01283176585
Nihal Mansour Abd El-Bari	nihalmansour0599@gmail.com	+2 01097737726
Hager Ismail Abdelaty	hager.aismail@gmail.com	+2 01065655744

Supervisor

Name	Email	Number
Dr. Khaled Soradi	khsoradi@eng.cu.edu.eg	+2 01277377726

This page is left intentionally empty

Chapter 1: Introduction

Achieving customers requirement is not an easy and fast target to achieve, it may require lots of time and effort to understand well these requirements and implement them in an efficient way, moreover this process will increase the initial cost of these projects due to technologies used and fast implementation needed to achieve a complete working system with all customer requirements, so ERevive will play a great role in this long and exhausting process by providing rapid and initial prototyping for these systems that will shorten the development process.

1.1. Motivation and Justification

Software systems that require dealing with structured query languages (SQL) is a complicated process to be automated and dynamically generated without interfering with developers which also acquire lots of effort and consume much more time and money, so current existing systems only implement some modules of this process like “Also Asked” product which gets most relevant questions from keywords entered to the system, “Natural SQL” that only converts natural language to SQL query, but ERevive go through the whole process including predicting the most relevant SQL queries then building automated web dashboard system that is connected to rest Api which gives the flexibility of dealing with the database through this generated user-friendly interface, and that will help the developer in his/her first steps in building the real system required by the customer by provided further developing on the generated code and features.

1.2. Project Objectives and Problem Definition

Our main project objectives are providing fast prototyping for the given system as in reality this process consumes time and effort so our project will build them in a short period of time compared to the real process taken by developers, moreover provide a detailed indication about system requirements by predicting the most relevant queries to the system which will facilitate the development process and save time and efforts without any additional initial cost that is most probably needed in the real process of generating fast prototyping to the systems. To sum up, the problem we need to solve by our project is generating rapid prototyping of systems in a short time by predicting the most relevant queries needed to be in the system for a more efficient development process.

1.3. Project Outcomes

ERevive outcome is mainly predicting the most relevant queries of the given system which will then be shown in the form of an integrated web dashboard that allows the developer to deal with a database that can be filled with dummy data by running data seeds to allow flexibility of testing the outcome of these queries, also the developer has the ability to add some natural sentence written in English and the system will convert it to SQL query which the user can add later to the system.

1.4. Document Organization

This document contains 6 chapters and a reference section. In chapter 1 we give a brief introduction about our system then we mention our motivation for choosing this project then we state our problem definition and our project objectives, and finally, we talk about the outcomes of our project. Chapter 2 discusses the feasibility study of our project and our target customers and our market survey and then we show our business case and financial analysis. In chapter 3 we discuss our literature survey about image processing and search engines and then we discuss our implemented approach. In chapter 4 we state our overview and assumptions and then we talk about our system design and architecture that describes our implemented modules which are image processing, schema information extraction, schema generation, search engine, natural language to SQL, API, and UI generation. Chapter 5 shows system testing and verification that includes the testing setup information and our testing plan and strategy for the implemented modules, then the testing schedule and comparative results to previous work. In chapter 6 we talk about our conclusion and future work including the challenges that faced us and the experience we gained from working on this project. After all of the chapters, we list all the references that were included throughout the whole document.

Chapter 2: Market Visibility Study

Automating software systems that require dealing with SQL databases and predicting SQL queries don't have a wide market as there is no existing product that goes through the whole process of creating such systems as most developers turn to the traditional way of building these systems by first collecting requirements from the user then start the development process.

There is no existing product that automates the process of creating the initial prototype of a system given only the ERD, however, there are some products that perform some modules separately for other reasons other than creating integrated software systems.

2.1. Targeted Customers

Our intended customers:

- Students Developers
    - ERevive can help student developers through their learning journey of SQL queries and database systems subjects while keeping eyes on the requirements of the market for better learning and visualization.

- Software Companies
    - ERevive will save money, time, and effort by providing fast prototyping that will help in bridging the gap between developers' and customers' requirements.

2.2. Market Survey

In this section we will list some of the products that perform sub-modules of our project but not all of the modules supported by ERevive.

2.2.1. UML to SQL Schema

Spark Systems introduces Enterprise Architect which is a modeling tool that has a lot of features including the feature of converting the UML diagram to a SQL schema which is very similar to part of our process which takes as an input an ERD and outputs SQL schema.

ERD is a little competitive compared to UML, as datatypes need to be predicted from ERD while it is already given in UML, also this product depends on the fact that it already knows the position of entities, attributes, and relations and their is no need to scan an image or perform image processing techniques.

Pros: It is more accurate so there is no need for any validation from the developer side like what we need in ERevive, as we mainly depend on image processing which may contain some errors.

Cons: It has some issues when mapping Inheritance onto a relational model.

2.2.2. Also Asked

It is a non-free website product that provides the service of taking keywords and predicting the most asked questions related to these given keywords, this approach is related in some way to part of our project as we take from the final schema generated some keywords that represent the detected entities, attributes and relations and try to find the closest relevant queries to these keywords by searching within large datasets using these keywords and their synonyms for better searching before creating the SQL query by mapping the returned queries to the system final schema and returning an ordered list of the generated relevant queries.

pros: predicting most relevant and accurate questions from keywords provided as it uses large datasets which will lead to better results in predicting questions.

cons: this product is not a free product, although it supports free trials for a certain number of times where customers can try the service before paying money for it.

2.2.3. Natural SQL

Natural SQL introduces Photon which is a tool for querying DB using NL. Photon takes as an input a natural language question and a schema then it will convert the question to a SQL query and then run it to acquire data from datasets.
Photon is a natural language Interface to database (NLIDB) system that is similar to our NLP to SQL module which enables the user to input a NL question that will be converted to a SQL query. Our module also gives the user the ability to add this query to the generated REST API.

pros: Photon is an encoder-decoder model which will handle more complicated questions to be converted to SQL queries while our module is a rule-based approach so Photon is more accurate.

cons: the transformation rules used in the encoder-decoder model can produce some errors which can affect the performance.

2.2.4. Github Copilot

It is a new service that GitHub provided earlier to suggest code snippets for developers while implementing their systems. This service is trained on a very huge dataset nearly millions of data that contains codes that existed in all public repositories in GitHub so this suggestion is nearly very accurate and efficient, moreover, it learns the developer style in writing code and suggests its code snippets in the same styling of its user. Regarding ERevive, we perform suggestions for SQL queries code either from searching for relevant queries by using a search engine or in the NLP to SQL module which suggests SQL query based on the NL question asked, we also support developers with the source code of generating the Vue application and API while also providing the ability to use its initial structure to complete the required system.

pros: it generates code snippets quickly, and is perfect for suggesting well-known algorithms like BFS, DFS and others, moreover it is good for suggesting packages and libraries needed to be imported into the code.

cons: it is an expensive service that might suggest code snippets out of the scope of what needs to be implemented in the system to perform its desired target.

2.3. Business Case and Financial Analysis

The product of our company will be presented in the form of a website that can be sold in both ways either for software companies or even published to all users through the internet with some limited features like blocking converting from NL to SQL query and blocking the availability of editing the predicted queries till upgrading to one of the premium modes, but we will support a certain number of trials to the system before purchasing. Moreover, we can select some professional advertisements to be shown on our public website that is almost related to the software development field.

Regarding the premium modes, we will have 3 modes:

Mode	Free Mode	1st Premium Mode	2nd Premium Mode	3rd Premium Mode
Cost	0$	20$	50$	150$
Number of usage times	3	10	30	100

Table 2.1: Different available modes from ERevive

Selected advertisements will pay nearly from 250$ to 350$ per month to be shown on the website, taking into consideration the advertisement market value that may affect the price of showing advertisements on our website during the next 5 years.

In the following table we will show the expected number of product sales over the next 5 years, and how we will adjust the price of our product to face any competitions that may be existing in the market during this period of time:

Years	Year 1	Year 2	Year 3	Year 4	Year 5
Average number of users for 1st premium mode	1000	2000	3500	5000	10000
Average number of users for 2nd premium mode	3000	6000	10000	15000	30000
Average number of users for 3rd premium mode	100	200	350	500	1000
Average advertising earning	10$ per day	15$ per day	17$ per day	20$ per day	25$ per day
Average usage days per year	100	150	200	250	300
Average profit of 1st premium	20000$	40000$	70000$	100000$	200000$

mode					
Average profit of 2nd premium mode	150000$	300000$	500000$	750000$	1500000$
Average profit of 3rd premium mode	15000$	30000$	52500$	75000$	150000$
Total average profit for the 3 premium modes	185000$	370000$	622500$	925000$	1850000$
Average profit from advertising	1000$	2250$	3400$	5000$	7500$
Total Profit	186000$	372250$	625900$	930000$	1857500$

Table 2.2: Business case in ERevive

Based on the previously mentioned analysis of the business case we will now perform financial analysis in order to estimate the capex and opex.

First the capex:-

We estimate that we will have a middle sized company with few computers for improving the existing features or even support more features like collecting more and more datasets that will greatly improve the output of search engine, providing UML as an input besides the ERD, or even allow users to draw either ERD or UML with in the application by dragging and dropping some of defined shapes to enter their required system design, so we will also need some employees that will help us in improving the product much more. First, we will need some computers for our employees with an estimated cost of 10000$, and in case we increase our dataset much more we will need servers with an estimated cost of 15000$ depending on the number of servers needed to store the dataset, then we will need some costs for the preparation of offices like tables, chairs, air conditioning systems, and many others with an estimated cost of 7000$, then we may buy some buses for employees that will nearly cost 10000$, so total estimated cost reaches 42000$.

Second the OpEx:-

We estimated that our monthly spending will be distributed among different aspects like paying the bills of water and electricity with an estimated cost of 200$, renting costs may reach upto 1500$, then for paying salaries with benefits to our employees that will cost us nearly 1000$ per employee so if we start with 12 employees, the

total costs of salaries per month will reach 12000$ but the number of employees supposed to increase if we decide to widen our company, finally we may need some costs for repairing any failures that might be existed with the estimated cost of 1000$, so the total cost of OpEx reaches 14700$ monthly.

To sum the costs mentioned above in Capex and Opex:

Required Items	Costs
The Capex	
Computers	10000$
Servers (may be required if the dataset is increased)	15000$
Preparing offices and decorations	7000$
Buses (for employees)	10000$
Total cost	**42000$**
The Opex	
Required bills	200$
Renting	1500$
Salary and benefits of employees (with 1000$ per employee and assuming we start our company with 12 employees)	12000$
Repairing and maintenance	1000$
Total cost	**14700$**

Table 2.3: Capex and opex

Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Capex												
Computers	833	833	833	833	833	833	833	833	833	833	833	833
Servers	1250	1250	1250	1250	1250	1250	1250	1250	1250	1250	1250	1250
Office Furniture	583	583	583	583	583	583	583	583	583	583	583	583
Buses	833	833	833	833	833	833	833	833	833	833	833	833
Opex												
Salaries and Benefits	12000	12000	12000	12000	12000	12000	12000	12000	12000	12000	12000	12000
Rent	1500	1500	1500	1500	1500	1500	1500	1500	1500	1500	1500	1500
Repairs and Maintenance	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
Bills	200	200	200	200	200	200	200	200	200	200	200	200
Total	18200	18200	18200	18200	18200	18200	18200	18200	18200	18200	18200	18200
Revenues												
1st Premium Mode	100	150	250	400	700	1000	1500	2000	2400	3000	4000	4500
2nd Premium Mode	700	1150	1900	3000	5300	7500	11200	15000	18000	22500	30000	33750
3rd Premium Mode	150	300	320	550	660	780	850	990	1200	1400	3200	4600
Advertising	70	70	80	80	80	90	80	90	80	90	90	100
Total	1020	1670	2550	4030	6740	9370	13630	18080	21680	26990	37290	42950
Profit	-17180	-16530	-15650	-14170	-11460	-8830	-4570	-120	3480	8790	19090	24750

Table 2.4: Cash flow table over a year

As we can see from the cash flow table the break even point is September and from that month onward we start making profit. It also could be noticed that our main profit comes from the 2nd premium mode as it has the highest contribution compared with

other revenues. We also notice that our least profit comes from advertising so it's not a dependable source for profit.

Chapter 3: Literature Survey

Our project mainly focuses on two large modules, the first one is image preprocessing and the second one is a search engine, so in the next section, we will discuss in detail the literature survey in both modules.

3.1. Image Preprocessing

Image preprocessing is an essential step for extracting useful information from images as before extracting any information from the image it needs to be converted to a digital form. Image preprocessing has various techniques that allow us to convert an image into a digital form that could be used to gain insights into that image.

In this section, we will talk about Thresholding, convex hull, and geometric shapes detection which are essential components of our project.

3.1.1. Thresholding

Thresholding is one of the simplest techniques used to segment an image which is a really important step in segmenting the image and makes it much easier to analyze the image.

The main idea of thresholding is to convert grayscale or RGB images to binary images (black and white pixels only).

Thresholding has different techniques and each one varies from the other in the speed and the accuracy of the binarization. In this section, we will discuss some of those techniques.

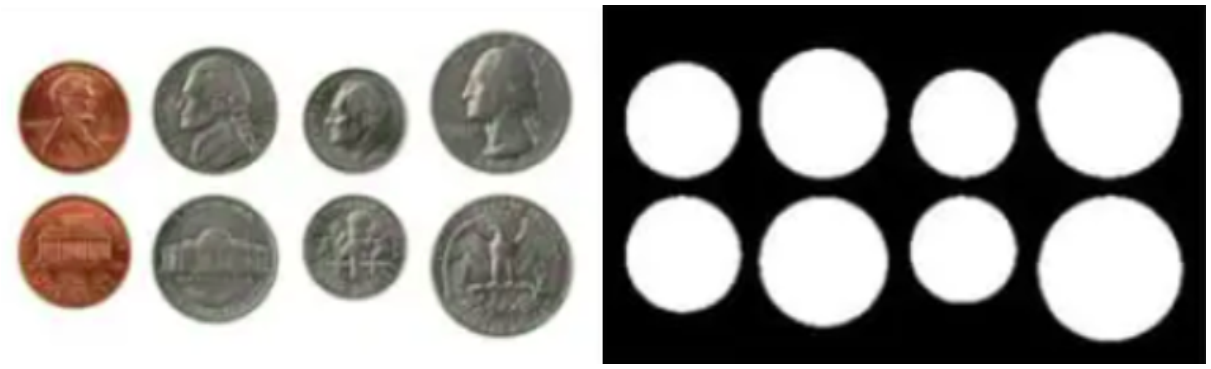


Figure 3.1: Thresholding. On the left is the RGB image and on the right is the binarized image (Adrian Rosebrock 2021)

3.1.1.1. Simple Thresholding

Simple thresholding requires human interaction as a specific thresholding value T is set by the person and then every pixel in the input image m that is larger than that threshold is considered a white pixel and the output image b and each pixel that is lower than that threshold is considered a black pixel.

$$b(i,j) = 1 \; if \; m(i,j) >= T$$
$$b(i,j) = 0 \; if \; m(i,j) < T$$

Simple thresholding is very fast and can be used to binarize the image in real-time but on the other hand, it's very simple so it doesn't produce greater segmentation.

3.1.1.2. Isodata Algorithm

Isodata is an iterative algorithm that is used to determine the thresholding value. Isodata algorithm first makes a random guess for the threshold value T then it uses this value to separate the image into classes after that the means m1, and m2 are calculated for each class then the new value for the threshold is the average of means m1,m2. The algorithm keeps iterating till the thresholding value T converges.[1]

$$T = \frac{m1 + m2}{2}$$

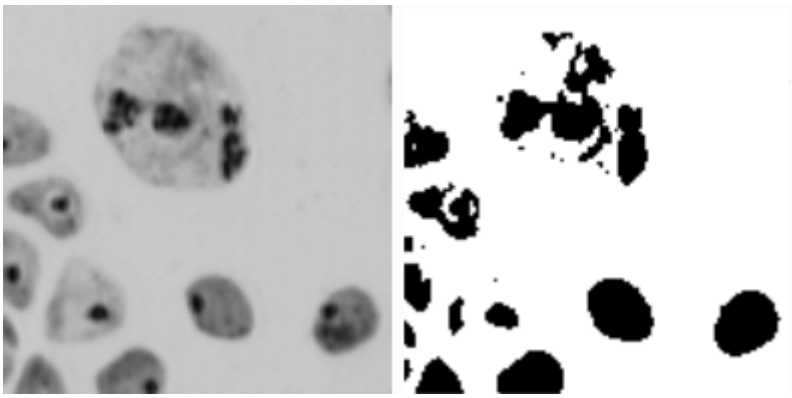


Figure 3.2: Isodata Thresholding. On the left is a grayscale image that has the objects in white and the background black. On the left is the thresholded image (Auto Threshold 2019)

Isodata algorithm is efficient and doesn't require human interaction as it calculates the thresholding value iteratively, However it could take some time to converge so it's not as fast as the simple thresholding.

3.1.1.3. Otsu thresholding

Otsu thresholding is an iterative global thresholding method that searches for the threshold that will either minimize the intra-class intensity variance or maximize the inter-class variance. Otsu thresholding is suitable for bi-modal images (images whose histogram has two dominant peaks).

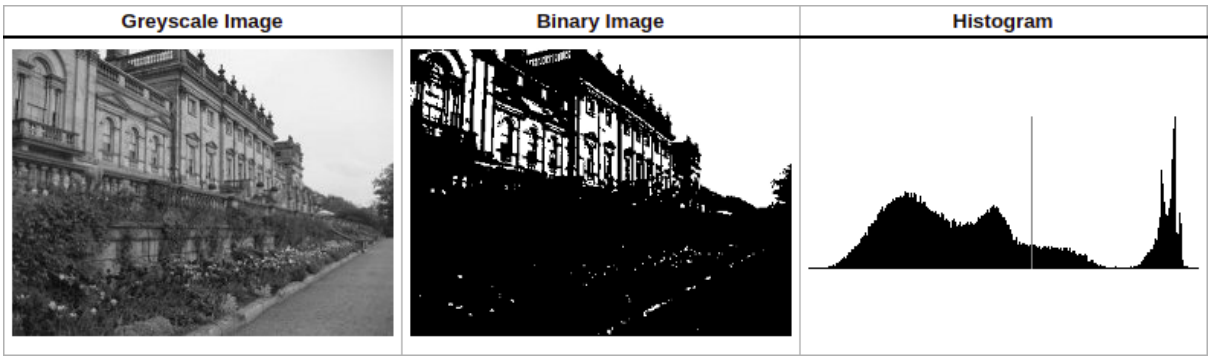


Figure 3.3: Otsu Thresholding (Wikipedia 2022)

Otsu thresholding is fast and not sensitive to outliers. it also is not affected by the brightness of the image and if the image a bi-modal it mostly generates accurate thresholding. On the other hand, it's sensitive to noise as it can change the histogram distribution.

3.1.2. Convex Hull

Convex hull is the smallest polygon that encloses a set of points and in the case of a binarized image, it's the smallest polygon around a shape that encloses all the white pixels of the shape in it. the convex hull is very helpful in shape analysis as it can approximate the complicated shapes to polygons and then that shape can be classified with the classification of its convex hull.

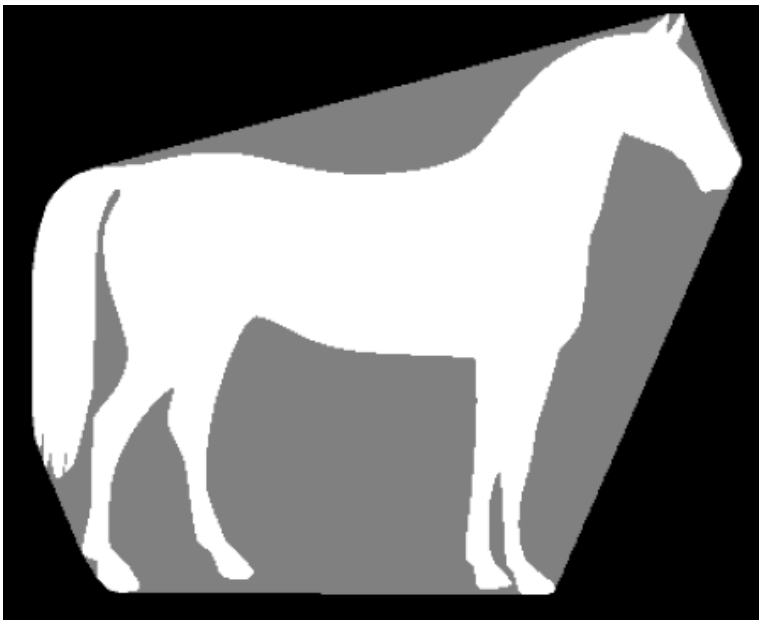


Figure 3.4: Convex Hull (the gray polygon) (scikit image)

3.1.3. Geometric shapes detection

Geometric shapes detection is a very important task based on the target need to be achieved in most computer vision projects, but for us, it is essential as detecting shapes will define entities, relations, and attributes that represent the building blocks of ERD.

Geometric shape detection could be implemented by many approaches. There are statistical, Machine learning, and deep learning approaches. For the machine learning approaches, features are extracted like the high-pressure regions of the objects (which differ from one geometric object to another), SIFT, and HOG and after that, a machine learning model like SVM is used for shape detection. The deep learning approaches include using NN and CNN for detecting shapes but it is known that deep learning solutions need a large dataset and preprocessing power. The final set of approaches is the statistical approach. We will discuss an approach from the statistical approaches in this section.

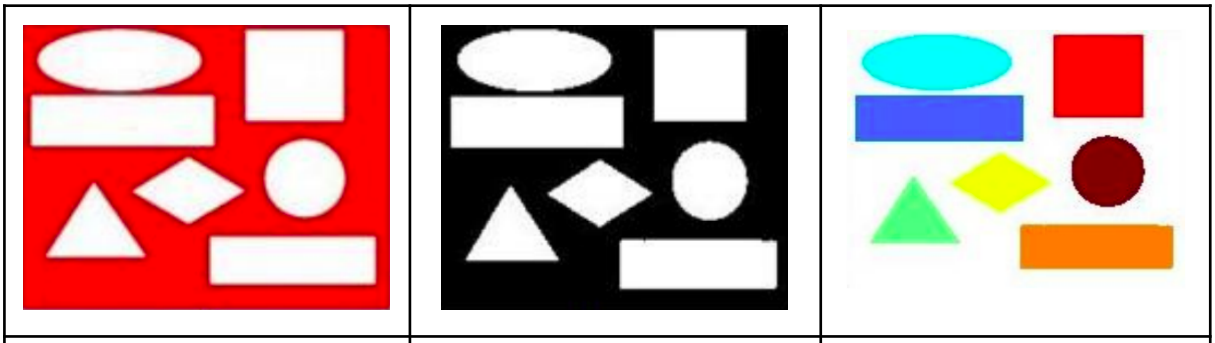
3.1.3.1. Statistical approach

The following approach was proposed in 2013 [2]:

- The image is first preprocessed and noise is removed then the image is binarized.
- After preprocessing the image than the process of shapes labeling starts which iterates on each non-zero pixel in the image and assigns a label for each of those pixels and then runs a union-find algorithm to group pixels together under their label.
- After the shapes are labeled the shape factor is calculated by the following formula:

$$shape\ factor\ = \frac{area}{(diameter)^{2}}$$

- Then the shapes are classified based on the shape factor of the labeled region as each shape has a specific range for its shape factor so if the labeled region shapes factor lies between the range then the region is classified as this shape.



Input Image	Preprocessed Image	Labelled Shapes

Figure 3.5: pipeline

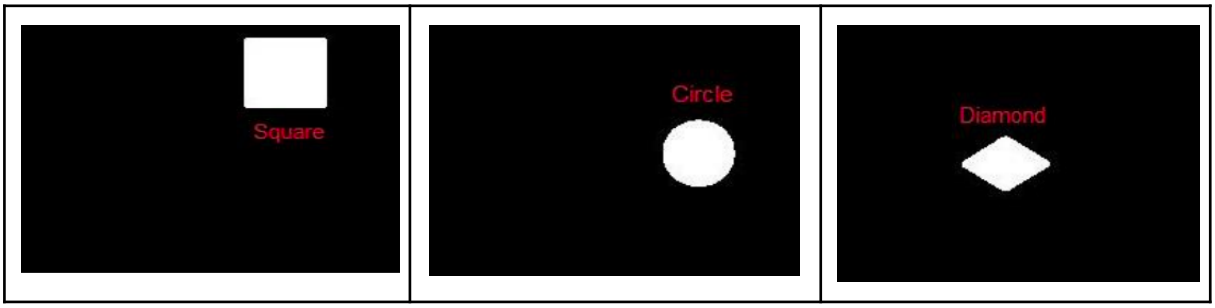


Figure 3.6: some classified shapes

3.2. Search Engine

We can not specify the word search engine without mentioning the great search engine of Google that handles over 3.5 billion searches per day in an efficient and super fast way, that's why it is the most popular search engine that is used by millions of people per day.

Basically, search engines take a query from users and then use it to return a list of records that are sorted from the most to the least relevant to the query asked, so behind the scenes there is a series of steps are taken to achieve better results in recommending the most relevant topics required, we will now discuss these steps in detail.

3.2.1. Crawling

It is a search process in which the search engine checks for new and updated contents by what is called crawlers, initially this process starts by fetching some known web pages and then will move on to the existed URLs in these pages to widen the crawling process and discover much more content to be used later in the process of searching and retrieving data according to the asked query.

3.2.2. Indexing

It is a process that is done after crawling to understand the content of pages by analyzing the text content existing in these pages as well as the key content tags and alt attributes like images and videos.

After determining the content of found pages a clustering process is applied to collect pages with similar topics together then only one will be selected from every cluster group to act as the group representative while other pages in the group are also stored but in the form of alternatives for the same topic, after that all of this information collected about clusters and their representative pages are stored in the index which is a large database that is hosted on a large number of computers to be used later for fast searching and retrieving of data.

Searching within individual pages for relevant keywords for the asked query is a very slow process unless search engines use what is called inverted index or reverse index which is a faster approach that stores text elements with pointers to documents where they exist, then search engine will perform the process of tokenization which is breaking up a sentence into a set of words or keywords called tokens which will be very helpful while searching with to retrieve data.

Regarding Google search engine, it takes also into consideration other factors that affect the retrieved data other than searching for relevant keywords to the user query like users' location and language that's why results returned for the same topic may differ from one country to another like searching for the nearest car repair shop which will show different results depending on the location of the user.

3.2.3. Search Results Ranking

After matching some pages related to the query of the user, search engines perform some ranking algorithm on the returned results so that they can order them from the most to the least relevant to better-helping users reach what they want quickly and efficiently.

Page ranking algorithm can simply be described as measuring some score that indicates the importance of this page over all other pages and this score mainly depends on calculating the number of links that points to it.

3.3. Implemented Approach

We will now discuss our implemented approach regarding the above two sections.

3.3.1. Image Preprocessing

First we want to fix the perspective of the given image for the purpose of correctly detecting the text and shapes existing within the given image so we have to first detect the corners of the image to be able to make some transformations to these corner points in order to fix the perspective of the image, for fixing the perspective of the image we used OpenCV library but it needs the contour to be fixed and the needed contour to be mapped with so our mission now is to pass perfect detected contour around the content of image so that this library will work efficiently, so we will perform blurring on the image to remove text from image then performing dilation to remove text in the image to reduce noise then perform edges detection and contours detection then choose the greatest contour to be the one which will be passed to OpenCV library as input to fix perspective.

Then we perform some thresholding techniques for changing the image into a binary image to be able to further process it. Further processing includes shapes segmentation and detection, we apply a rule-based approach in geometric shapes detection by calculating the minimum rectangle area, minimum rotated rectangle area, and minimum ellipse area that is all enclosing the shape then get the area of the convex hull of the shape then by analyzing these values for each type of shapes

we perform a set of range conditions to these areas divided by the hull area and each range represents a type of shapes.

3.3.2. Search Engine

The goal of this module is to recommend queries to the input system by searching in the previous similar system so we applied the search engine steps but here searching is done with entities names as search keywords as well as enhancements to map returned results to the input schema, the following is a brief of how the system is approached:

- Dataset collection: SQL queries were collected from multiple resources and APIs including GitHub API, IMDB open API, great SQL, yelp dataset and a lot more.

- Parser: In parsing the collected SQL is tokenized and lemmatized then we extract entities handled in this query as well as attributes of different types and associated operations if any:

    - Select attributes
    - Aggregation attributes
    - Where attributes and the operation performed on it
    - Having attributes
    - Group by attributes
    - Order by attributes

- Indexer: Each query in the dataset is represented in the form of one hot vector with each keyword(entity or attributes) and their synonyms extracted from the parsing is turned to one. The input query(combination of entity names) is also turned into one hot vector. Then dot product is performed with queries in the datasets. The resulting number is the number of hits our search query has with every query in the data set.

- Mapper and Joiner: Queries that performed hits are mapped with either exact match of synonyms match to our entities' names and attributes names then the mapped entities are mapped through relations represented in Foreign keys to get the most compact group including the goal entities found in the query returned from the search.

- Queries clustering: We cluster the mapped queries by entities so that each cluster has a set of queries performed on the same set of entities then we apply merging on each cluster by merging queries that share the same filtering attributes into one query that collects their select attributes.

- Ranker: Each query is represented in a matrix filled with word vectors then we get the Matrix correlation coefficient between search keyword and retrieved query to get only syntactically queries, we generated dictionaries for Ngrams from the dataset that represent the probabilities of entities, attributes and the combination between them, then we get the final queries by ranking them first by the closeness of entities in the original and mapped queries second by attributes and entities coverage than by the grams probabilities for the combination of the query entities and attributes.

Chapter 4: System Design and Architecture

4.1. Overview and Assumptions

Our product is a website that takes as an input ERD image and performs some image processing techniques to extract information from the image which will be used in the generation of the initial schema, then we will allow the user to make some changes to this schema and to validate it in order to make sure that it is correct without any errors for further processing as we will take some keywords from the final schema then move to the search engine which will return a list of relevant queries that will be ranked from the most to the least relevant to the system, then we give the developer some flexibility in editing and deleting the generated queries before moving to the API and UI generation, moreover, the developer has the ability to write some questions in the NL to SQL module and his question will be automatically converted to the corresponding SQL query, the developer also has the ability to run the generated queries within the generated UI so that he/she can deal with the database which can be also dynamically generated through running the database seeds.

4.2. System Architecture

4.2.1. Block Diagram

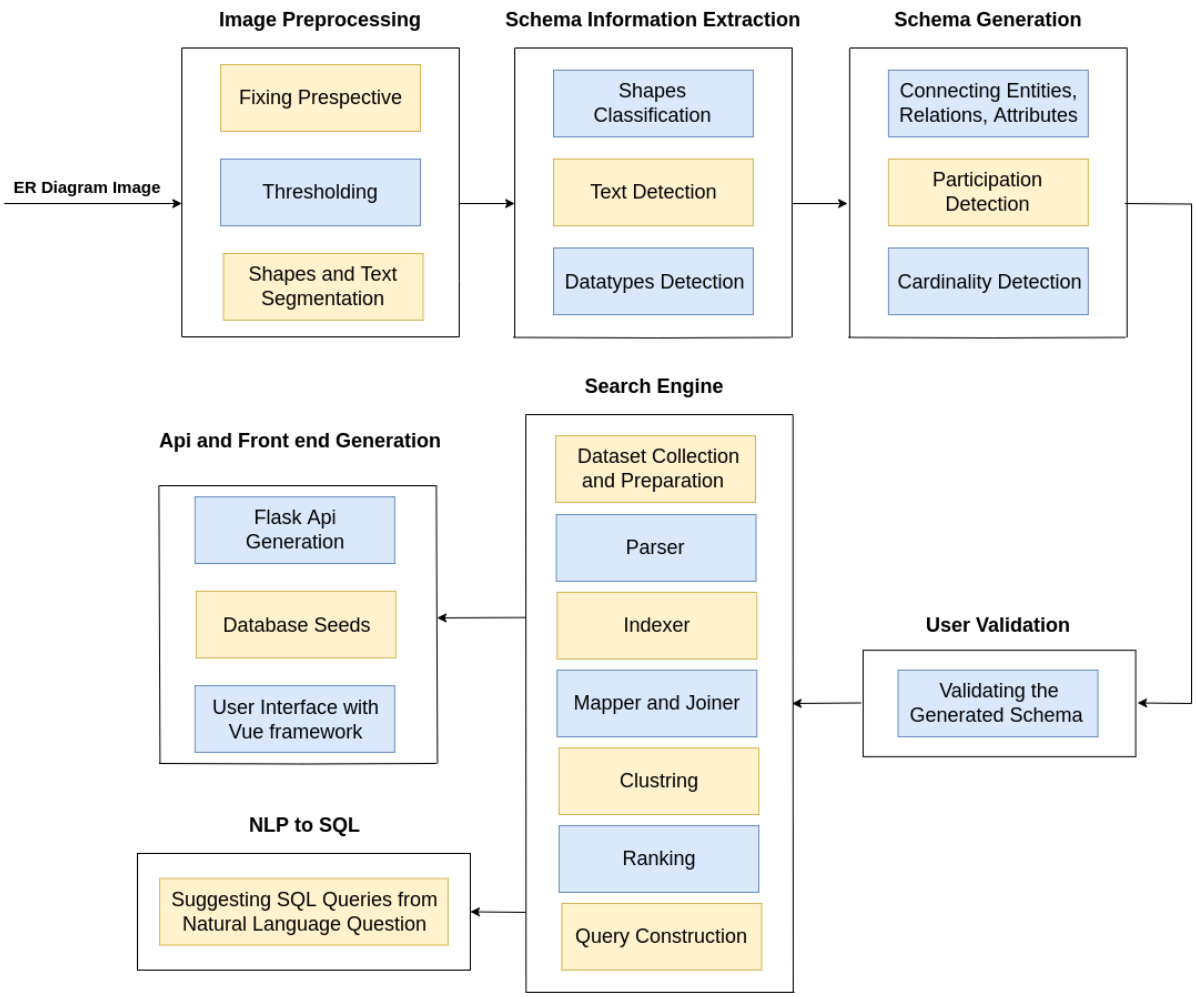


Figure 4.1: System modules design

4.3. Image Processing

4.3.1. Functional Description

This module is responsible for preprocessing to get a binarized image and then segmenting shapes and text inside it.

4.3.2. Modular Decomposition

4.3.2.1. Image preprocessing

Perspective warping is done to combat perspective distortion of the given image captured by a camera so, we have to first detect corners of the image to be able to make transformations to these corner points to fix the perspective of the image (OpenCV library is used) so we will perform blurring on the image to remove text from image then performing dilation to remove text in the image to reduce noise then perform edges detection and contours detection then choose the greatest contour to be the one which will be passed to OpenCV library as input to fix perspective.

Also doing erosion to close small openings in shapes and then applying convex hull function which smoothes outer contour and also closes shapes with small openings.

Finally binarizing the image to be easily handled by the rest of the pipeline.

4.3.2.2. Segmentation

In order to segment the shapes the lines connecting the shapes need to be removed so flood filling the image with black from its corners results in lines being consumed and shapes remaining isolated from each other but there are two challenges that need to be handled:

- open shapes can be internally flooded and be lost, this is handled by erosion and convex hulls done in preprocessing.

- voids between shapes can form closed loops which can be falsely detected as a shape this is handled by getting all shapes and getting their areas excluding shapes with areas deviating with a big z score(considered outliers and discarded) also shapes not containing any text (inner contours with the small area) are discarded.

To detect weak shapes run-length encoding(after removing text) is performed if a pattern of four lines is found, and a weak shape is detected.

The shapes and text are separated from the original image by taking the detected contours of the shapes and then for each contour the filled image of the shape is retrieved by its contour from the filled image then the binarized image that contains the text (text is white on a black background) and the filled shape (shape is white on

a black background) is indeed so the out image has only the white text on a black background and then the text image is forwarded to the OCR and the shape is forwarded to the detection of the shape module.

4.3.3. Design Constraints

Shapes have to be relatively closed (the system has tolerance for slightly open shapes).

4.4. Schema Information Extraction

4.4.1. Functional Description

This module is responsible for analyzing segmented shapes to classify it to entities, attributes, and relations and perform optical character recognition to text images then datatype recognition using multinomial naive Bayes is performed on text associated with attributes.

4.4.2. Modular Decomposition

4.4.2.1. Shapes Classification

In order to classify the shapes we apply a rule-based approach:

- Get the convex hull of the shape and its area.
- Get the area of the minimum non-rotated rectangle enclosing the shape.
- Get the area of the minimum rotated rectangle enclosing the shape.
- Get the area of the minimum ellipse fitting the shape.
- Divide each area of the three areas by the hull area.
- Each type of the three shapes (Rectangle, Diamond, Ellipse) has an allowed range of values for the divided areas.
- By analyzing these ranges boundaries we notice that rectangles have Arect_rotated > 0.8 and Arect_non_rotated > 0.8 and Ellipse < 0.95,

diamonds have Arect_rotated > 0.6 and Arect_non_rotated < 0.6 and Ellipse < 0.95, other than this it's an oval.

4.4.2.2 Text Detection

This module aims to detect two things, the first one is the primary keys and the second one is the text that exists in the image.

This module input is the images containing only texts generated from the segmentation step that separates texts away from shapes, so here the image entered in this step is only with text and a line in case this image represents a primary key text of attribute.

First images pass through little preprocessing techniques like reading them as grayscale images then applying some thresholding on them, then trying to detect if this image represents the text of the primary key or not.

Detecting primary keys is done by first getting the bounding rectangles or contours of all text and lines exited in the image, then by checking the position, length, and width of the generated contours, we can determine if this image represents a primary key or not as the condition taken is that the line is placed at the second half of the image (in the bottom of image) with a width greater than the quarter of the image and height less than the quarter of image then we save the contour found with these descriptions to be later used as it is necessary to delete this line before passing the image to the OCR so that it will not be considered as noise and affect text detection, so we color all the pixels within this contour with white color then passing this image to the OCR which is using library Pytessract that takes an image containing text and extracting the contained text.

4.4.2.3. Datatypes Detection

This submodule inputs the text detected from the text detection module and then applies datatypes prediction.

4.4.2.3.1. Dataset Collection

In this module there was no previous dataset for predicting SQL column name data types so the dataset was gathered from the following resources:

- Yale University COSQL dataset [5] which is a dataset for NLP to SQL prediction this dataset contained NLP queries and the schemas of which those Queries are made on so the schemas were gathered from the COSQL dataset and then a script was written to extract each column name and its datatype from those schemas.

- Guest Train is a database that contains multiple SQL databases the schemas of those SQL databases were exported and a script was written to extract each column name and its datatype.

- ManyTypes4py [6] is a benchmark for type inference in python. It has over 5,382 python projects with each variable name and the inferred data type. This dataset was added to our dataset for type detection.

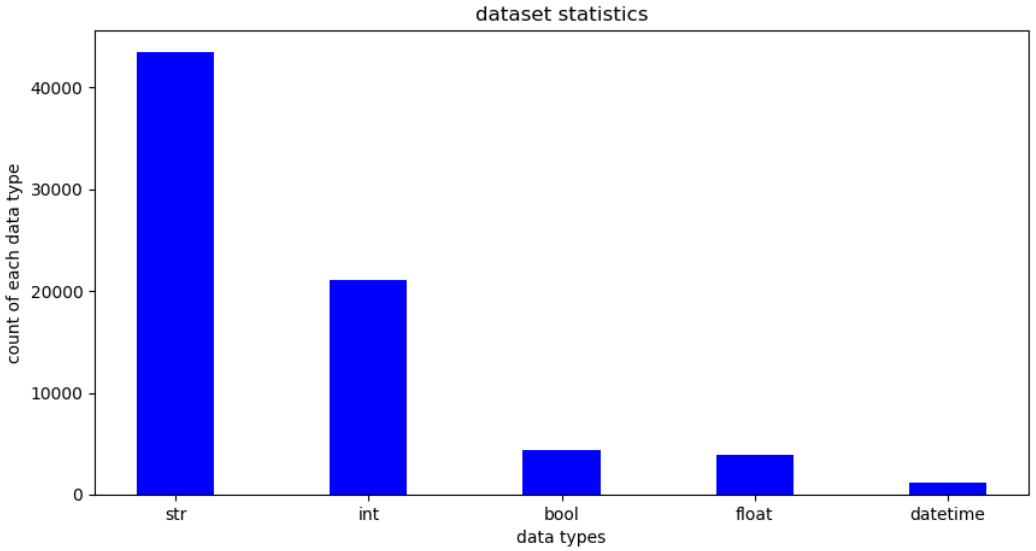


Figure 4.2: Final dataset statistics

Feature Extraction

when the column name is entered into the module the following preprocessing is made on the column name:

- validate the column name and check if the column names are only numbers.

- convert camel cases, snake cases, and pascal formats into space-separated words.

- remove numbers at the beginning of the column name.

- convert the column name to lowercase

After the previous preprocessing if the column name is an empty string then a default prediction (string) is returned. If the column name is valid then the TFIDF is extracted for that column name.

4.4.2.3.2. Datatypes Model

As could be concluded from the dataset statistics in figure 4.1, the dataset is imbalanced and also is not suitable for a deep learning approach, Multiple models were tried like decision trees, SVM, and logistic regression but the best model was multinomial naive Bayes model with an accuracy of 72% which is somehow expected as naive Bayes works well with imbalanced datasets.

4.4.3. Design Constraints

This model only predicts 5 data types which are (int, float, string, bool, DateTime).

4.5. Schema Generation

4.5.1. Functional Description

In order to generate the SQL schema:

- Attributes have to be associated with the entities.
- N-M relations and multivalued attributes have to be converted into independent entities.
- Foreign keys had to be added to tables on the N side of the relation and full participation.
- Foreign keys had to be added to tables on either side of the relation in case of 1-1 relations and partial participation.

4.5.2. Modular Decomposition

4.5.2.1 Participation detection

In order to detect participation between each relation and its entities:

- Get two random points on the contours of the relation and entity respectively.
- Apply BFS between these two points and get the path points.
- Filter the path points to contain only the direct path points between the relation and entity without any points on their contours.
- Get a minimum of (10, path length) of random points on the path.
- For each one of these points gets the closest point that is separated from the first point by a max of 10 pixels.
- If more than half of the random points have these closest separated points then it's full participation else it’s partial participation.

4.5.2.2 Cardinality detection

In order to detect cardinality between each relation and its entities:

- For each relation we get its contour and remove it from the image then do the same with all the paths of this relation that connect it with its entities.
- We get the closest point for each path to the center of the relation to detect the path position relative to the relation (right, left, top, bottom).
- Given the detected position we take a window with an area half of the relation area.
- We then extract this window from the image and input it to the OCR that outputs the cardinality text (1/N/M).

4.5.2.3 Connect components

In order to detect the connections between all the objects in the ER like detecting the attributes connected to each entity:

- We first skeletonize the image which contains the detected shapes.

- we create the colored contours image which is an image that has the detected shapes colored by the index of each shape. So for example, if the shape index is 0 then the pixels inside that contour are colored as 0. This will make it easier to detect if a pixel is inside a shape or not and the index of the shape to which this pixel belongs.

- we first try to find the dead ends by running BFS from the shapes and storing the ends of each path that doesn't find a shape at the end of it:

    - The BFS is given the contour of the entity shape and then we traverse the contour row by row till we find a white pixel (top left pixel of the entity).
    - Then we start the BFS from that white pixel and each traversed pixel's color is changed to white (we erase the black pixels we traverse so we don't traverse them again).
    - When we pop a pixel from the queue we check if this pixel belongs to another shape or not by checking the color of the pixel in the colored contours image. if the color of the pixel is not white then we detected a new shape. so we mark that shape as visited and include it in the list of shapes detected for this entity.
    - If while we are traversing a path from the entity and at the end of that path we didn't find any shape, we store that point in the dead ends list as this could mean that there is a shape nearby but is disconnected from the line with few pixels so we keep those dead ends to try to connect them to the nearest shape to them.

- After we gather all the dead ends we try to connect dead ends to dead ends by getting the distance between the dead ends and if the distance is less than 5 pixels then both dead ends will probably belong to the same shape.

- We try to find if we extended the path in the direction we were going by 5 pixels would find a shape or not and if we found a shape we connect the dead end with that shape.

- then after connecting all the deadends we run BFS from all shapes and store the information of the connected shapes to each shape.

4.5.3. Design Constraints

Cardinalities need to be near the relations in order to be correctly detected, also shapes have to be detected (not necessarily connected as the system has tolerance by slightly disconnected lines near shapes).

4.6. Search Engine

4.6.1. Functional Description

The search engine module takes a schema as an input and outputs the top suggested queries to this schema.

4.6.2. Modular Decomposition

4.6.2.1. Dataset collection and Parser

- **Dataset collection**
    - We first collect queries from different datasets of SQL queries, these datasets are: Cosql [5] - Spider [7] - Academic [8], Advising [9], nvBench [10], Geography [11], IMBD [12], Splash [13], yelp [14], Sparc [15], Restaurants - Scholar, Then we input the collected queries to the Parser module.
- **Parser**
    - The parser first tokenizes the input query.
    - Loop on each token and check if it's a start of a new clause then we start adding the following tokens to this token to the info array attached to the new clause.
    - Clean the token and get its synonyms and add them to a synonyms dictionary.
    - At the end of the parser we add the clause info to a query dictionary and then move on to the next queries.

4.6.2.2. Indexer

In order to search in a fast way we chose to index each query using the sum of all one hot vector of keywords found in the query, where each word and its synonyms are represented using the same one-hot vector.

The input keywords we use to search for queries are combinations of different entities' names also turned into one hot vector then we find the dot product with queries in the datasets. The resulting number is the number of hits our search query has with every query in the data set.

4.6.2.3. Mapper and Joiner

- Mapper: Mapping is done first on entities. Each entity is written in, camel case or pascal case, or snake case, are done then we calculate the hits the same way we did in the indexer and we map to the highest scoring entity in our schema. Attributes are mapped first to the entity that preceded its name(entity.attribute) if any then to any mapped entity attributes then in entities that were added during the join.

- Joiner: the Algorithm for joining entities is inspired by distributed multicasting joining algorithm in networks where each entity start searching (multisource Breadth-first search) for other entities that it is trying to join with and once all entities are joined path are backtracked to remove any path that did not add new required entities on its way. This leads to the most compact group including the goal entities found in the query returned from the search.

4.6.2.4 Clustering and Ranking

- **Clustering**

  In order to reduce the number of repeated generated queries we perform clustering and merging.

  - Queries that have the same entities are grouped in the same cluster.
  - In each cluster we merge queries that have the same where attributes, group by attributes, having attributes, and order by attributes to the same query by merging their select attributes and aggregation attributes then update the group by attributes to be valid with any newly added aggregation attributes.

- **Ranking**

  The ranker rank the queries returned by the indexer by:

  - Getting the keywords of the query and the input schema.
  - Getting the word2vec vector [16] for each query and the word2vec for the input schema.
  - Calculating the correlation with the RV coefficient by hoggorm library in python then sorting the queries by the highest correlation.
  - Filter queries by eliminating the queries that have less than 0.5 correlation.

  Then We rank the final mapped queries by the following approach:

  - Generating Unigram dictionaries from the dataset by calculating the probabilities for each entity and attribute with their synonyms.
  - Generating N grams dictionary by calculating the probabilities of the combination of entities with attributes representing each entity and attribute with One Hot Vector so the Ngrams dictionary doesn't perform exact matching but synonyms matching.
  - For each mapped query in a cluster we calculate:
    - Entities closeness: the absolute difference between the original queries and mapped query.
    - Attributes coverage: sum of coverage for each query which is calculated by the count of matched words between the original attribute and the mapped attribute, given that matching words is done by the dot product between the One Hot Vector of the two words to perform synonyms matching not exact word matching.
    - Entities coverage: same as Attributes coverage but on Entities.
    - Length of entities goals: entities goals are an array of entities generated by the Joiner module to join two correctly mapped entities through the shortest possible join path of entities.
  - We sort the queries first by the length of goals, then by the entities' closeness, then by the sum of the Attributes coverage and Entities.

- After getting the most relevant queries from the first sorting we sort the equalities by the sum of the Ngrams values of the select and where attributes of this query.

4.6.2.5 Query construction

We take the final queries mapped info as select attributes, where attributes, aggregation attributes, having attributes, order by and entities then convert them to the relevant SQL query.

4.6.3. Design Constraints

The system is as powerful as the data is. so, searching for completely unseen systems will result in a small number of good queries and suggestions.

4.7. Natural language to SQL

4.7.1. Functional Description

This module is a rule-based approach whose main function is converting natural language questions written by the user in the English language into a SQL query.

The input of this module is the natural language question and the final schema generated.

4.7.2. Modular Decomposition

The general idea of how we approached this module is that we map tokens after knowing the POS and mapping whether a noun word is an entity or an attribute and in which clause of the sentence we are in for example when we find an “if” we assume we are in a where clause so this natural language token is reduced to known token (a# for attribute token, w# for a token that marked the start of where clause) then we start consuming these known tags one by one and mapping matched rules/clauses to SQL.

- First apply tokenization to the sentence.

- For every token we get the POS tagging to determine whether this word is a noun or verb if so we get the lemma of the word and store it inside the list otherwise we store the word as it is.

- We will now detect whether the detected nouns and verbs represent entities or attributes found in the given schema then we will store them for further checking.

- Now we will separate the values in the question that are placed between two square brackets.

- Search for conjunctions found in the question so we can collect them in one set, to do so we first scan the question searching for conjunctions, and send the previous and next word to another function that will detect its type and store it in a dictionary along with its specified type while also writing an equivalent question to the related one but this time with names indicating the type of word placed here, for example, we replace the aggregation detected words with g, where keywords with w, attributes found with an order by keywords with o, group by keywords with gb, the condition detected with c and finally the way of ordering the returned data either descendingly which will be replaced with d or not writing anything for default sorting ascendingly.

- now we will scan the newly created sentence with symbols for generating the final dictionary that will be used in query construction, first, we want to distribute the aggregation keywords detected with its attribute, then perform the same thing with the group by and append all of its attributes and finally do the same for the order by while taking in consideration to also append the specified direction of ordering.

- Select attributes are also detected to be all of the attributes found in the reconstructed sentence that is presented before the entity so keeping the order of attributes to be mentioned before the entity is very important.

- Finally construct the query from this collected information that is stored with its query type and name in a dictionary so we only use this dictionary for the query construction with the specified rules of SQL queries.

4.7.3. Design Constraints

- Since this module is a rule-based one then it is limited in keywords detection to only keywords found in its dictionaries.

- Queries here applied for only one entity.

- Having inside queries will be done for only one condition and on one column.

- ordering by multiple columns will be done using the keyword “then”.

- order the sentence to be in the form of aggregation then attributes, and the where clause to be followed with attributes, conditions, values (not necessary in the same order but they are all required to be mentioned), and the order of writing attributes to be selected before the entity.

- To select some attributes followed by multiple aggregations it is necessary to place “then” between them.

- Values inside the question should be written between two square brackets.

4.8. API and UI Generation

4.8.1. Functional Description

User of our system is provided with the source code of an integrated website for his system. Our goal in this module is to automate the process of converting SQL schema and queries to API endpoints with Flask and SQL alchemy then generate a user interface that consumes this API and navigates through different routes displaying different functionality of the websites.

4.8.2. Modular Decomposition

4.8.2.1 API generation

- Creating an automated script that converts the input schema to SQL alchemy database models taking into consideration the right way of handling primary keys and foreign keys constraints like on delete and update perform cascading, also mapping the data types previously detected to the defined data types of SQL alchemy to guarantee correct construction of the models and database.

- Creating the CRUD endpoints on each model in the schema.

- Creating the API for each generated queries cluster:

    - we create the namespace for the cluster and then for each predicted SQL query we convert the SQL query to a Flask-RESTX resource with a get endpoint

        - select -> query
        - where -> filter_by
        - group by -> group_by
        - having -> having
        - orderby -> order_by
    - The aggregation functions like (count, min, max) are mapped to the sql alchemy syntax.
        - count -> func.count
        - min -> func.min
        - max -> func.max
    - The operators like (in, between, or, like) are also mapped.
        - in -> in_(values_list)
        - between -> between(val1,val2)
        - like -> like(val)
    - After converting the SQL query to an endpoint we write the endpoint to its file in the API folder.

- After the clusters' endpoints are created, the following files are created:

    - The app init which contains all the imports for the clusters and is responsible for creating the main API that has all the namespaces registered to and connecting to the database.
    - app utils that contain common functions used by other modules.
    - The requirements file contains all libraries needed for setting up the API environment.
    - The app setup and runs scripts that are responsible for creating the API environment and running the API.
    - An env file contains the username, password, and name for the MySQL database
    - A generated UML image for the database schema

4.8.2.2 Database Seeds generation

After the API is generated the script that generates the seeds for that specific schema will run and the seeds files will be generated. When the user clicks on the database icon, the seeds files will be inserted into the database.
the generated dummy data is generated using a python package called faker which is a package that has multiple providers for each type of data. For example, faker has providers for generating ssn, names, dates, etc.

The process of generating the seeds files is:

- First each column name in the schema is mapped to a faker class. For example, the first name is mapped to the first name provider in faker, ssn is mapped to the ssn provider in faker, and so on.

- Then data is generated for the primary key first so we can get the values for the foreign keys.

- After that the insert SQL statements are constructed and each insert statement is written in a .sql file inside the seeds folder.

4.8.2.3 UI generation

By taking the Api generated each endpoint is granted a component depending on the method type so a post or put request has a form component to allow the user to fill with the distinction that put form is prefilled with object data whereas getting endpoints has either cards or tables to show data with the ability to fill filters for specific endpoints also delete icon is displayed for delete endpoint.

These components are displayed in different routes as a cluster of APIs are displayed on a single page and the user can navigate to different endpoints from it.
We also provide a single data store for the website which is responsible for keeping fetched data and fire requests to fetch this data.

Below we will show some of the images for the generated UI.

The following image shows the welcome page to the generated UI after query suggestion.

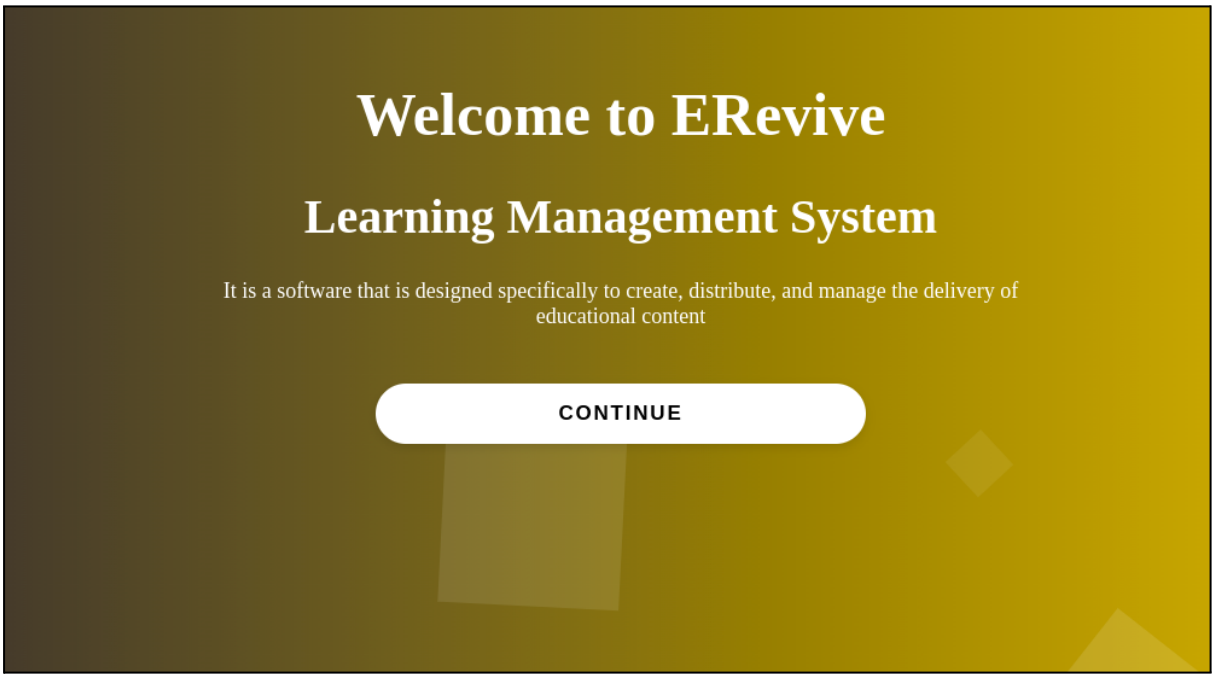


Figure 4.3: Welcome page of generated UI

The following image shows the dashboard that will show the returned data as a result of getting the request:

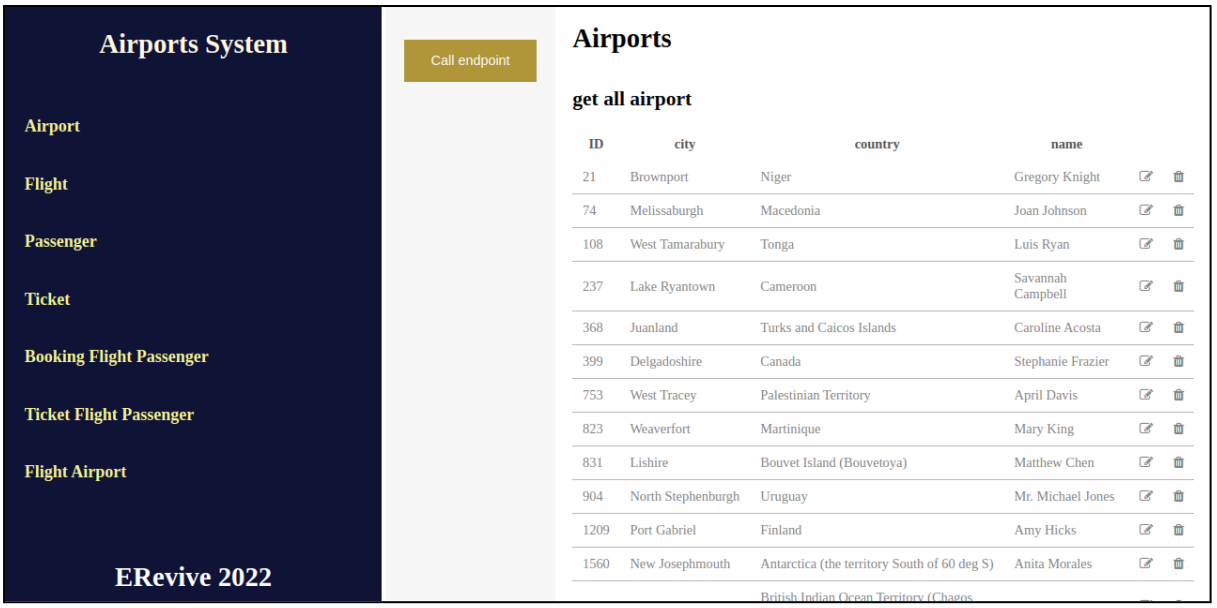


Figure 4.4: Get request dashboard in the generated UI

The following image shows the form that will be used for posting a new record to the database or even editing an existing one:

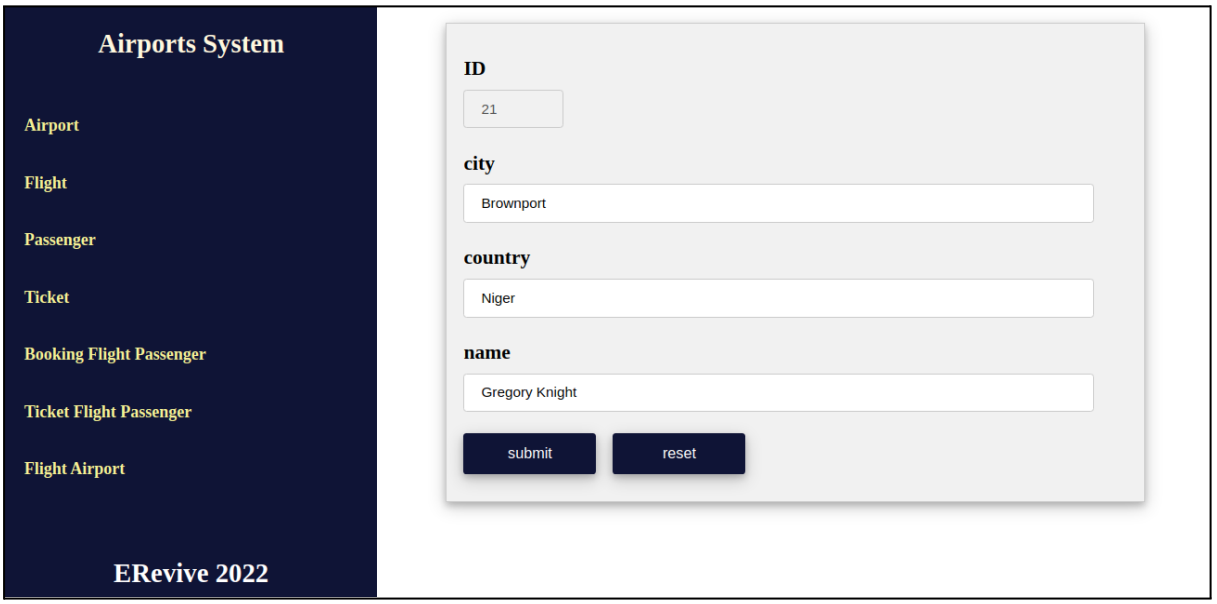


Figure 4.5: Post and put requests form in the generated UI

Chapter 5: System Testing and Verification

5.1. Testing Setup

In order to test our module we create ER diagrams for different systems manually with different image qualities and input these images into the system then test each module output to the correct output.

5.2. Testing Plan and Strategy

We create ER diagrams for different kinds of systems like Ecommerce systems , Movies systems , CRM systems and others then measure the correctness of each module with different criteria.

- **Image processing module**

    - No of detected shapes : #StrongEntities , #WeakEntities , #NormalAttributtes ,#MultivariateAttributes ,#NormalRelations , #IdentifyingRelations
    - No of detected primary keys
    - Detected relations types: No of 1-N ,No of M-N , No of 1-1 , No of NaryRelation , No of FullParticipation , No of PartialParticipation

- **Schema generation module**

    - Detected data types for each attribute : "str" , "int" , "datetime" , "float" , "bool".
    - Entity Schema generation
        - Entity name
        - Entity attributes
        - Entity primary keys
        - Entity foreign keys
        - Entity type : strong-weak

- **Search engine module**

    - Our project main objective is to suggest relevant popular queries to the input system so our testing strategy in the search engine module is to collect er diagrams for different kinds of systems like Ecommerce systems , Movies systems , CRM systems and others then measure the relevancy and popularity of our queries to the queries usually needed in these kinds of systems.

    - We categorize the search engine output queries into 4 different categories for each test case
        1. Matched queries between our system and another system.
        2. Good queries: queries relevant to the test system that are likely to be used in other systems in the same systems type but not matched with the current test system.
        3. Neutral queries: queries have average or low importance.
        4. Bad queries: queries not relevant to the system or have zero importance.

- **Application generation module**

  We generate REST API in python, flask, SQL alchemy, and a dashboard in Vue for the final generated queries so we measure the following:

  - Number of correct mapped APIs.
  - Functionality of the dashboard with the API.

5.2.1. Module Testing

In the following section we will discuss the measurement of each module for different input study cases.

5.2.1.1 Image Processing Testing

5.2.1.1.1 Customer relationship management system

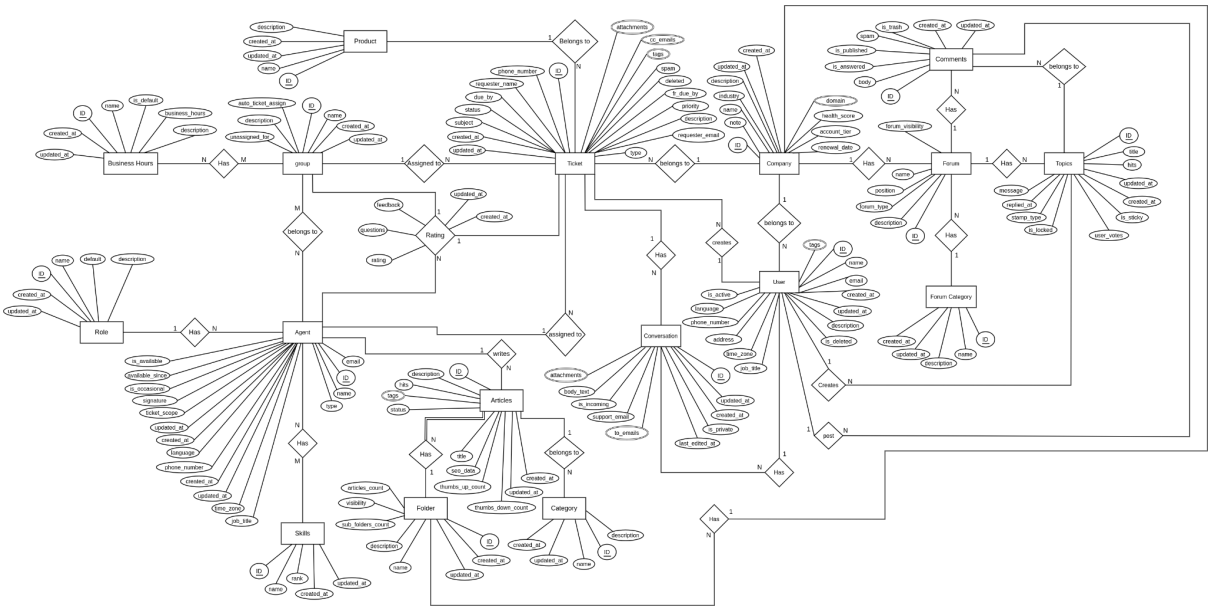


Figure 5.1: ER diagram for FreshDesk CRM system

Detected Shapes

Shape	Detected	Correct Output
Attribute	Multivalue: 8 Normal: 150	Multivalue: 8 Normal: 150

Entity	Strong: 17 Weak: 0	Strong: 17 Weak: 0
Relation	Identifying: 0 Normal: 24	Identifying: 0 Normal: 23

Table 5.1: Detected shapes in FreshDesk

Detected Primary Keys

Detected	**Correct Output**
17	17

Table 5.2: Detected primary keys in FreshDesk

Detected Relations

Relation type	**Detected**	**Correct Output**
One to many	19	19
Many to many	4	3
N-ary	1	1
Full participation	2%	2%
Partial participation	98%	98%

Table 5.3: Detected relations in FreshDesk

Detected Text

Correctly detected	**Total**
7	198

Table 5.4: Detected text in FreshDesk

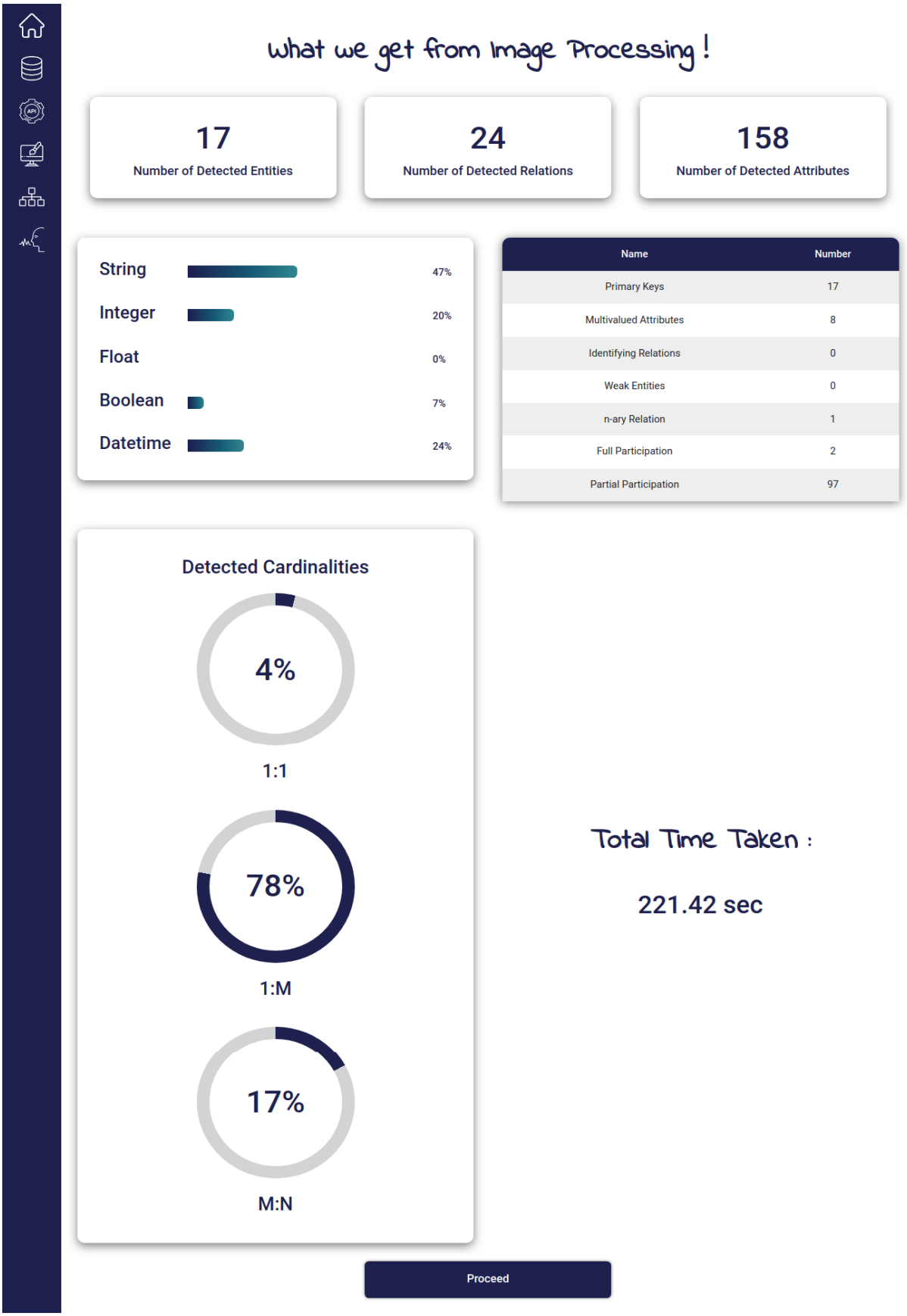


Figure 5.2: ERevive Image Processing Output For FreshDesk ER

5.2.1.1.2 Schoology learning management system

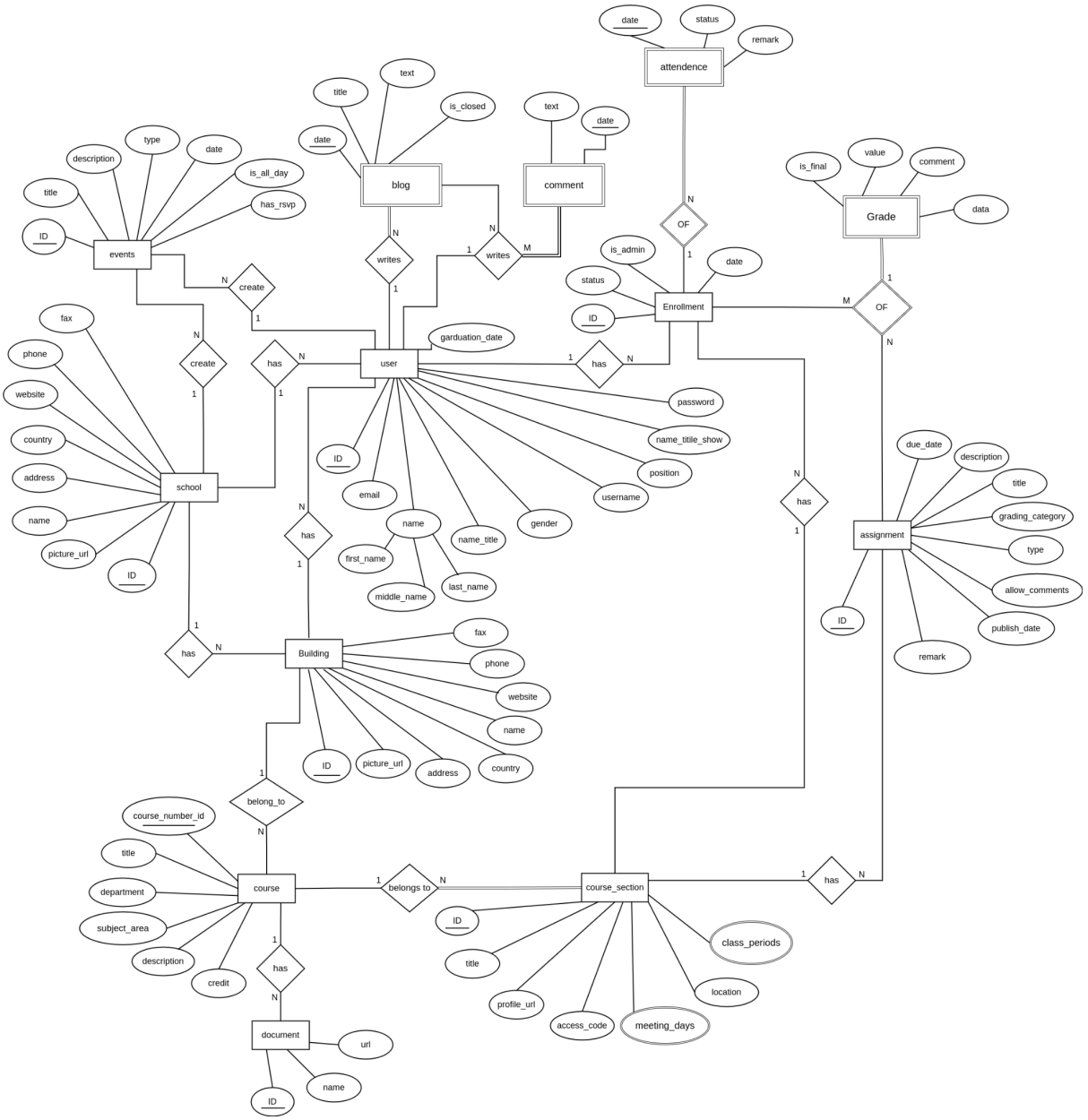


Figure 5.3: ER diagram for Schoology LMS system

Detected Shapes

Shape	Detected	Correct Output
Attribute	Multivalue: 2 Normal: 75	Multivalue: 2 Normal: 75
Entity	Strong: 10 Weak: 4	Strong: 10 Weak: 4
Relation	Identifying: 2 Normal: 13	Identifying: 2 Normal: 13

Table 5.5: Detected shapes in Schoology

Detected Primary Keys

Primary key type	Detected	Correct Output
For strong entities	10	10
For weak entities	3	3
For multivalue	2	2
Total	15	15

Table 5.6: Detected primary keys in Schoology

Detected Relations

Relation type	Detected	Correct Output
One to one	0	0
One to many	15	15
Many to many	0	0
N-ary	2	2
Full participation	12%	12%
Partial participation	88%	88%

Table 5.7: Detected relations in Schoology

Detected Text

Correctly detected	Total
102	106

Table 5.8: Detected text in Schoology

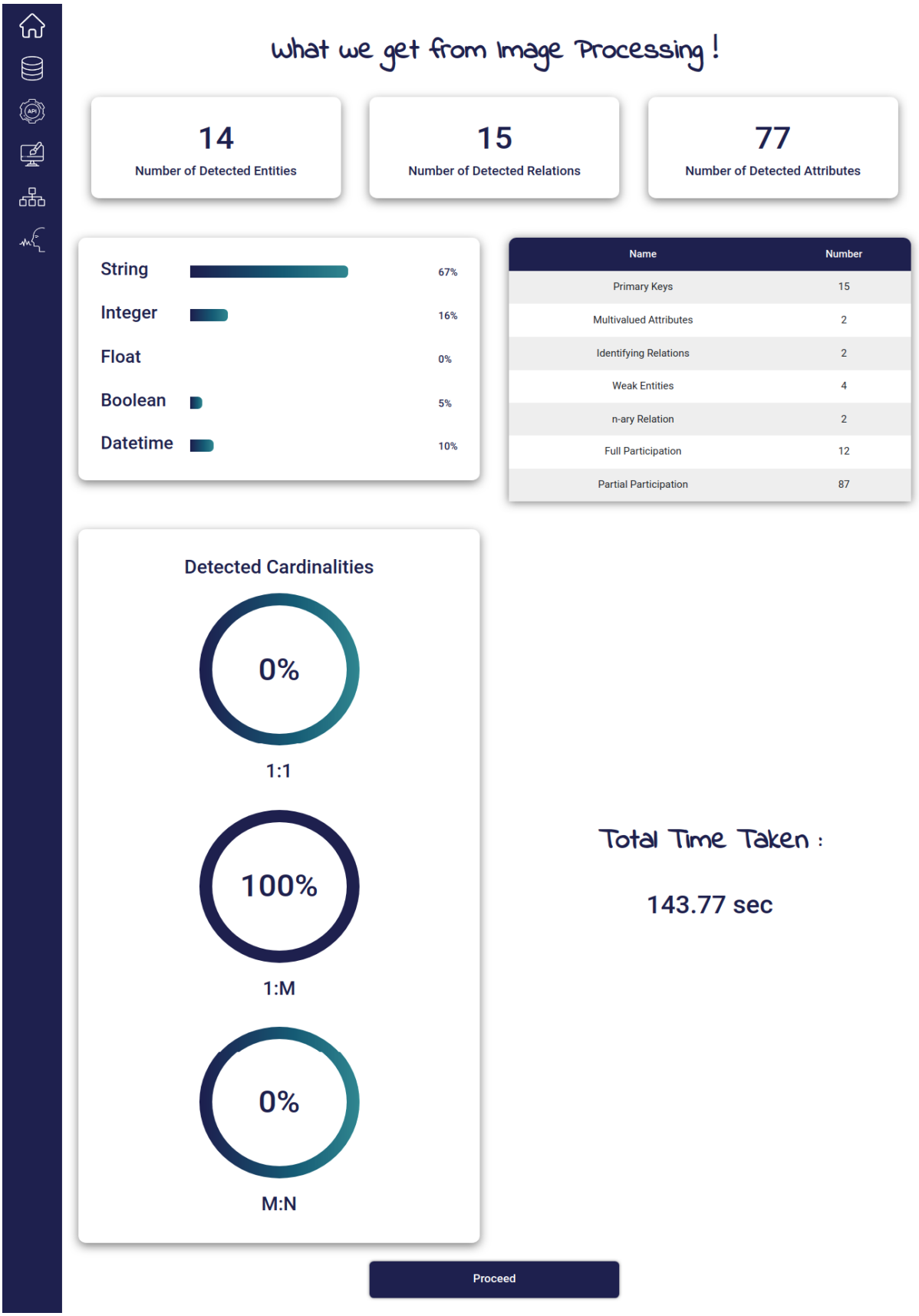


Figure 5.4: ERevive Image Processing Output For Schoology ER

5.2.1.1.3. Company system

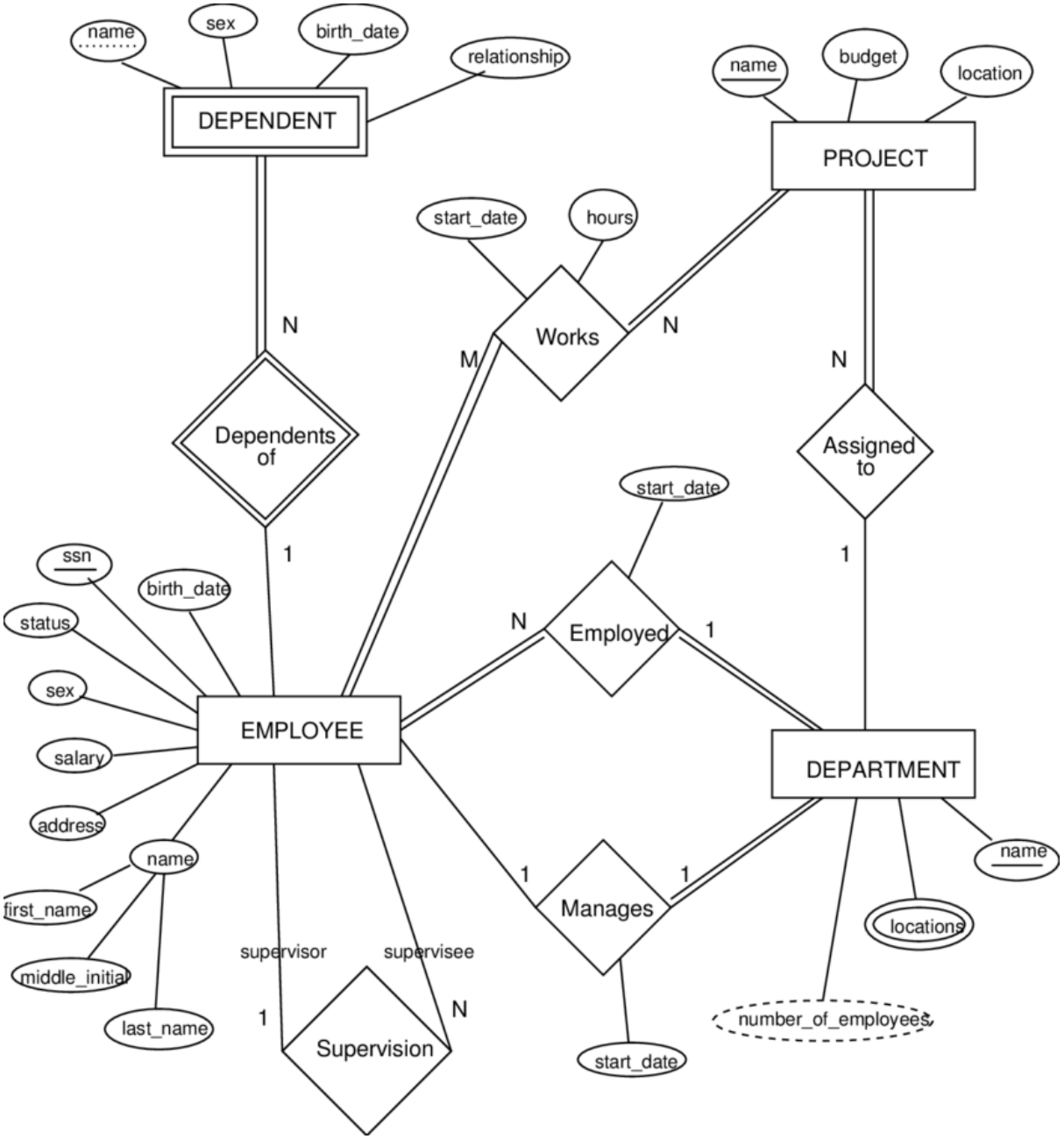


Figure 5.5: ER diagram for company system

Detected Shapes

Shape	Detected	Correct Output
Attribute	Multivalue: 1 Normal: 22	Multivalue: 1 Normal: 22
Entity	Strong: 1 Weak: 3	Strong: 1 Weak: 3
Relation	Identifying: 1 Normal: 5	Identifying: 1 Normal: 5

Table 5.9: Detected shapes in company

Detected Primary Keys

Primary key type	Detected	Correct Output
For strong entities	3	3
For weak entities	0	1
For multivalue	1	1
Total	4	5

Table 5.10: Detected primary keys in company

Detected Relations

Relation type	Detected	Correct Output
One to one	1	1
One to many	4	4
Many to many	1	1
N-ary	0	0
Full participation	64%	64%
Partial participation	36%	36%

Table 5.11: Detected relations in company

Detected Text

Correctly detected	Total
26	33

Table 5.12: Detected text in company

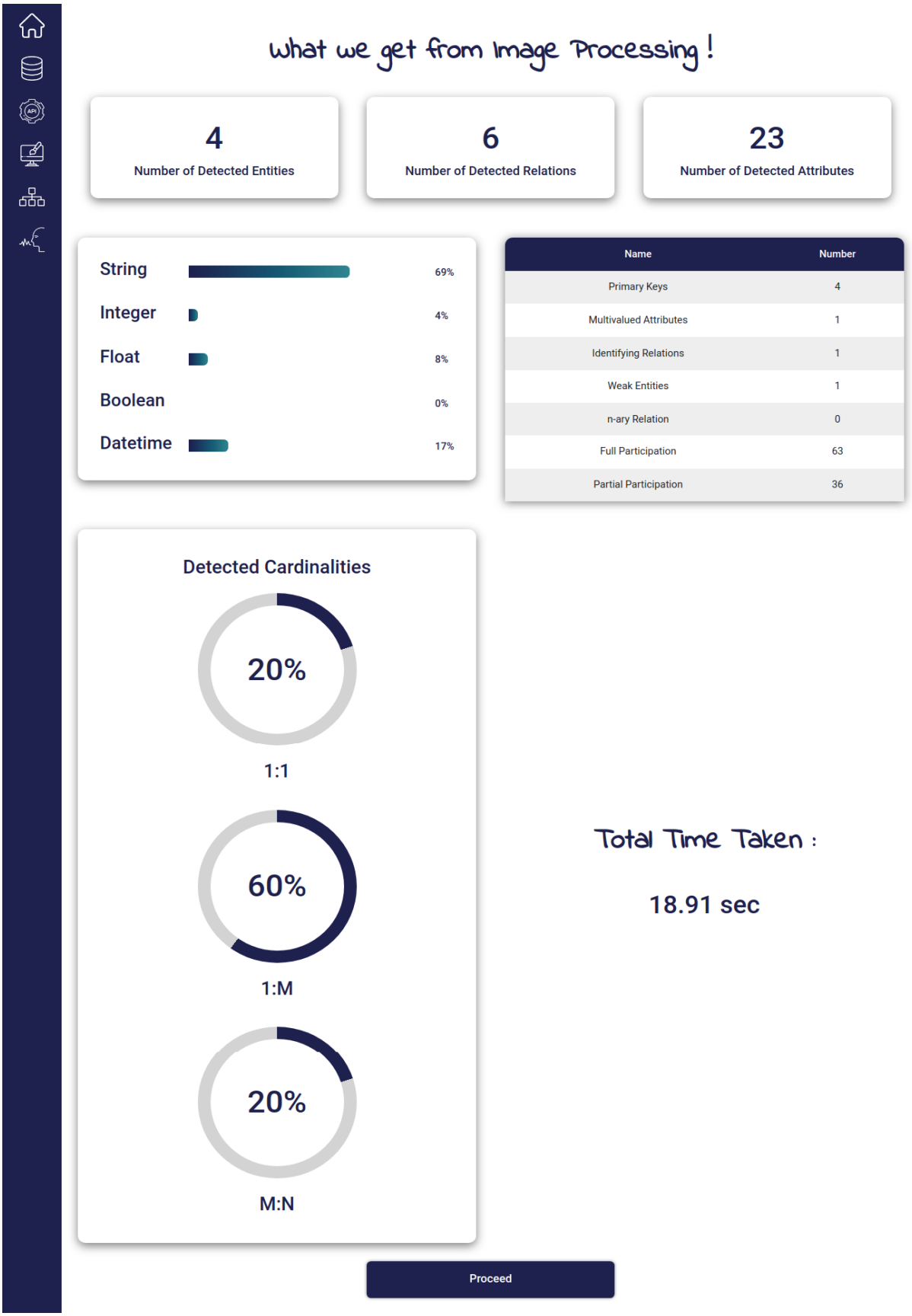


Figure 5.6: ER diagram for Schoology LMS system

5.2.1.2 Search Engine

Each system is divided into clusters containing one or more entities that the following queries are targeting.

5.2.1.2.1. Schoology learning management system

School Entity

Good Query	Bad Query	Matched with API
Filter by Id	Group by id	Get all schools
Order by name		Filter by id
Filter by name		
count schools in country		
count schools in an address		
Filter by name		

Table 5.13: Generated queries for School entity

Building Entity

Good Query	Neutral Query	Bad Query	Matched with API	Not Matched with API
filtered building by name	get building ordered by id	get building filtered by address grouped by name	get building filtered by id	Get school building
get building ordered by name		get building grouped by school id, building id, and name		
filter building by country		get building ordered by count_all		

filter by address				
count building with same name				

Table 5.14: Generated queries for Building entity

User Entity

Good Queries	Neutral Queries	Bad Queries	Matched Api
Filter by last name and order by first name	Order by id	Group by id	Get all users
User filtered by Id	User id less than value	group by last_name, first_name, id, password, email	Filter by id
Filter by username	Filter by first name and id	that t	
Filter by first name and Lastname	Group by the first name and password		
Filter by gender and graduation date	count first name occurrences and order by first name		
Filter by graduation date	Count users with the same username		
Order by username	Count user with same first name		
Filter and order by first name			

Table 5.15: Generated queries for User entity

Grade Entity

Good Query
get grade filtered by enrollment id, assignment id ordered by value
get grade filtered by enrollment id, assignment id and date

get min max and average grade filtered by assignment id

Table 5.16: Generated queries for Grade entity

Grade, class section, assignment Entities

Good Query	Matched with API
Count assignments for all course sections	get grades for course section
Count assignments for a specific course section	
get grades for course section	

Table 5.17: Generated queries for Grade, Class section, Assignment entities

Course, User, building Entity

Good Query	Neutral Query
count user having a given name in a building having a certain name	Count unique user names in a certain building with a certain course
count user having a given name in a building having a certain name offering a certain course	
Get user's names in a certain building that provides a specific course	

Table 5.18: Generated queries for Course, User, Building entities

Assignment Entity

Good Query	Bad Query	Matched with API
Filter by id	get assignment filtered by description grouped by allow_comments	Filter by id
Filter by ids in a certain array and assignment title	Group by id and title and order by allow_comments	Filter by section id
Filter by allowing comments and ordered by due date	get assignment grouped by type, allow_comments ordered by	
get assignment ordered by due_date , publish_date		

get assignment filtered by id, allow_comments		
get count assignment grouped by type, allow_comments ordered by type		
get assignment filtered by allow_comments		

Table 5.19: Generated queries for Assignment entity

Assignment, Course, course section Entities

Good Query	Neutral Query	Matched with API
get assignments course course_section data	get assignment course grouped by grading_category, title ordered by grading_category	filtered by assignment id, and section id
get assignment count in course grouped by course title		
get assignment in course_section filtered by id		
get assignment's course course_section ordered by grading_category		

Table 5.20: Generated queries forAssignment, Course, Course section entities

Document Entity

Good Query	**Bad Query**	**Matched with API**
get document filtered by name	get document filtered by name grouped by id	get document filtered by id
get the document in a certain course	get document filtered by name grouped by course id	
Count documents in	get document filtered by	

course	name ordered by count_id	
get document filtered by id , name		
get document filtered by id		

Table 5.21: Generated queries for Document entity

Enrollment Entity

Good Query	Bad Query	Matched with API	Not Matched with API
get enrollment filtered by course section id	get enrollment grouped by id ordered by count_all	get enrollment filtered by course section id	get enrollment filtered by user id
get enrollment filtered by enrollment id		get enrollment filtered by enrollment id	

Table 5.22: Generated queries for Enrollment entity

Events Entity

Good Query	**Matched with API**	**Not Matched with API**
get events grouped by date ordered by date	get all events	get events by id
		get event per school

Table 5.23: Generated queries for Events entity

Blog Entity

Good Query	**Bad Query**	**Matched with API**
get blog filtered by user id	get blog grouped by text , user id , date , title	get blog filtered by user id

Table 5.24: Generated queries for Blog entity

Blog comments Entity

Good Query	Bad Query
get blog_comment grouped by date	get blog_comment filtered by date grouped by text
get blog_comment ordered by date	

Table 5.25: Generated queries for Blog comments entities

User, Blog, Blog comments Entity

Good Query	Matched with API
Get all blogs and comments in the system with their user data	Get user, blog, and blog data comments filtered by user id
Get user, blog, and blog data comments filtered by user id	
get count blog_comment for a certain user	

Table 5.26: Generated queries for User, Blog, Blog comments entities

User, Blog Entity

Good Query	Bad Query	Matched with API
Get all blogs in the system with their user data	to get blog user grouped by id	Get user, blog, and blog data comments filtered by user id
Get users, blog filtered by user id		

Table 5.27: Generated queries for User, Blog entities

5.2.1.2.2. Airport system

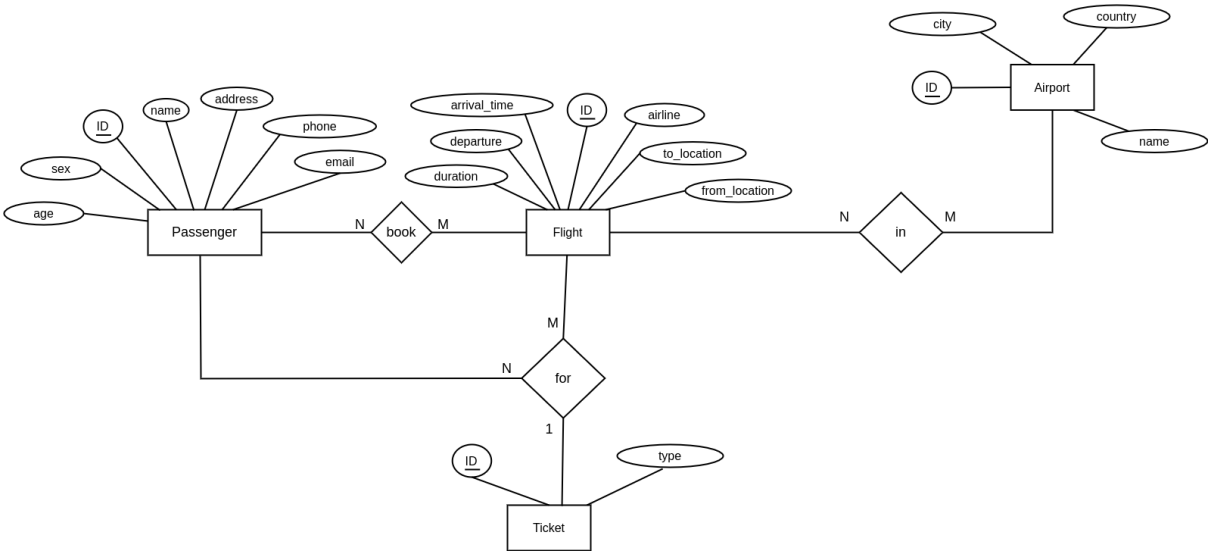


Figure 5.7: ER diagram for airport system

this system queries output is compared to 3 popular Airport and Traveling management APIs:

1. Airport Guide API
2. IATACodes Api
3. Amadeus Api

Airport Entity

Neutral Query	Matched With Apis
get airport ordered by name	get airport filtered by city
	get airports count in a city filtered by country
	Get cities filtered by country and airports count
	get airport code filtered by name
	get airport filtered by country

Table 5.28: Generated queries for Airport entity

Flight , Passenger , Ticket , Airport

Good Query	Bad Query	APIs Matched with
get flight grouped by airline ordered by count_all	get flight grouped by duration , id , arrival_time ,to_location , from_location , departure , airline ordered by count_all	get flight filtered by airline
get flight grouped by arrival_time , id	get ticket grouped by id	get flight filtered by to_location , duration
get passenger filtered by id		get flight filtered by arrival_time
get passenger filtered by name		get flight filtered by departure
Get passenger and count of flights grouped by passenger_id		get flights and airport filtered by airport code
get passengers grouped by flight_id ordered by count		get flights and airport filtered by arrival time and airport name
get a ticket filtered by id.		Get flights and airports grouped by city
Get count ticket and passengers filtered by count_ticket		get a ticket for all flights and passengers
		get tickets, flights for a passenger.

Table 5.29: Generated queries for Airport entity

5.2.2. Integration Testing

We performed manual testing for a complete integrated generated website from a given schema. We tested whether the generated website is behaving as expected from this given er. We wanted to test if all components are displayed in their expected view and that it consumes the right API and that the API is behaving as expected.

5.3. Testing Schedule

Each module was tested as soon as it was done and by the start of June, we started creating ER from openly known APIs and comparing it to our results.

5.4. Comparative Results to Previous Work

Although our system is inspired by different other products which do similar or one job of our system, there is no previous complete product like ours.

Chapter 6: Conclusions and Future Work

In this chapter we will conclude our project, our supported features, and the challenges that faced us as a result of implementing a new idea project that is not previously implemented by any other developers as a whole, so we tried different approaches till reaching the final working approach of the project, so we learned a lot from this project as we nearly implement all of the modules from start to end with little usage of helping libraries and packages, and we aim to continue improving our project and adding additional features that will facilitate more the process towards users and companies and to be fully automated without any interfering from the user.

6.1. Faced Challenges

This project is a new idea. There is no previous existing module that performs what our project does, as a result, we faced many challenges during building this project, below we will give a list of some of the large challenges we faced.

- Lack of SQL queries datasets, we collect datasets as much as we can and we try to perform crawling in order to increase our dataset much more as well as trying to synthesize data but these approaches failed.

- Our project modules were inspired by some products, but as mentioned before there is no product existing that performs the whole of the process so we try different approaches of previous work that are somehow related to what we want to implement but we add our improvements and changes to make it better suit the sequence of our project development. We tried to be as innovative as possible.

- We tried another approach that generates SQL from a well-known schema with data filled in it. It was an approach implemented in a paper so we tried to mimic it by gathering schemas with filled data but it failed as it highly depended on statistics gathered from the filled data so it was very limited.

- We also tried to implement SQL query generation by deep learning using the seq2seq model but it also failed due to a lack of large datasets.

- In the module on converting natural language to SQL queries, we also faced some challenges as the papers we read were not specifying in detail how to

perform such approaches, so we designed our own approach to seek our goal of generating the SQL query inspired by subjects such as compilers.

6.2. Gained Experience

We gained lots of experience while building this project, below we will list some of these experiences gained.

- Our research skills have improved as we searched a lot for approaches to implement in our modules.
- Problem-solving and innovation skills as we face many challenges and try to overcome them by changing approaches or finding other alternatives to solve them.
- We get hands-on experience in image processing, NLP, and AI.
- We learned how to automate the creation of API and Vue applications and also learned how to write bash scripts.

6.3. Conclusions

Our project is a large one with many different modules that support converting from an ERD image to a suggesting system for the given ERD, as we predict the most relevant queries that are most probably required in such systems, moreover we list those predicted queries to the developer before proceeding in the creation of the API and UI source code so that the developer can revise them and has the ability to edit or delete any query he/she want then after finishing proceed to the next step where we also support an automated generation of API and user interface with the ability to running data seeds so that data is automatically filled in the database for better testing the generated queries. We also support another feature about converting Natural language questions written in English to SQL queries with the ability also to add some changes to the created query.

there are some limitations in our project as we listed in detail above in section 4 within each module like:

- Shapes needed to be relatively closed.
- a limited number of data types.

- Writing the cardinalities near to the relations in order to be correctly detected.

- Regarding NL to SQL module there are multiple constraints as it is implemented by rule-based approach:

    - Converting NL to SQL query is done for one entity.
    - Order of attributes then the target entity is important.
    - Adding values within square brackets.
    - For ordering multiple attributes add “then” between them.
    - “Having” is done on one attribute and one condition.
    - Selecting attributes with aggregations will require to be separated with “then”, only if they are both together otherwise no constraint.

6.4. Future Work

There are many improvements we aim to do:

- Supporting UML Diagram.

- Give the user the ability to draw the diagrams inside the application by drag and drop feature for predefined shapes needed either for UML or ERD.

- Increasing the dataset of the search engine for better results and generalization.

- We may require to distribute our dataset in case of increasing it over multiple servers for further processing.

- Support generating the source code of the frontend side with other frameworks like React and Angular.

- Support generating the source code of the backend side with other frameworks like Node and Ruby on Rails.

References

[1] UrsulaGonzales-Barron, FrancisButler, “A comparison of seven thresholding techniques with the k-means clustering algorithm for measurement of bread-crumb features by digital image analysis”, Journal of Food Engineering, Volume 74, Issue 2, May 2006.

[2] Nidhal El Abbadi1, Lamis Al Saadi, “Automatic Detection and Recognize Different Shapes in an Image”, International Journal of Computer Science Issues, Vol. 10, Issue 6, No 1, November 2013.

[3] Adrian Rosebrock, “OpenCV Thresholding (cv2.threshold)”, 2021, https://pyimagesearch.com/2021/04/28/opencv-thresholding-cv2-threshold, last access: Apr 2022.

[4] Auto Threshold, “yen threshold”, 2019, https://imagej.net/plugins/auto-threshold, last access: May 2022.

[5] Tao Yu and Rui Zhang and He Yang Er and Suyi Li and Eric Xue and Bo Pang and Xi Victoria Lin and Yi Chern Tan and Tianze Shi and Zihan Li and Youxuan Jiang and Michihiro Yasunaga and Sungrok Shim and Tao Chen and Alexander Fabbri and Zifan Li and Luyao Chen and Yuwen Zhang and Shreya Dixit and Vincent Zhang and Caiming Xiong and Richard Socher and Walter Lasecki and Dragomir Radev, “CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases”, Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, 2019.

[6] A. M. Mir and E. Latoskinas and G. Gousios, “ManyTypes4Py: A Benchmark Python Dataset for Machine Learning-Based Type Inference”, IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), 2021.

[7] Tao Yu and Rui Zhang and Kai Yang and Michihiro Yasunaga and Dongxu Wang and Zifan Li and James Ma and Irene Li and Qingning Yao and Shanelle Roman and Zilin Zhang and Dragomir Radev, “Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task”, Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018.

[8] Kaggle, “Students' Academic Performance Dataset”, https://www.kaggle.com/datasets/aljarah/xAPI-Edu-Data, last access: May 2022.

[9] Metatext BETA, “Advising Dataset”, 2018, https://metatext.io/datasets/advising, last access: Apr 2022.

[10]Yuyu Luo and Jiawei Tang and Guoliang Li, “nvBench: A Large-Scale Synthesized Dataset for Cross-Domain Natural Language to Visualization Task”, 2021.

[11] Harvard, “Geography Datasets”, 2010, https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/SPHS5E, last access: Apr 2022.

[12] IMDB, “IMDb Datasets”, https://www.imdb.com/interfaces, last access: Jul 2022.

[13] Ahmed Elgohary and Saghar Hosseini and Ahmed Hassan Awadalla, ”Speak to your Parser: Interactive Text-to-SQL with Natural Language Feedback”, Association for Computational Linguistics, 2020.

[14] yelp, “Yelp Dataset”, 2021, https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset, last access: May 2022.

[15] Tao Yu and Rui Zhang and Michihiro Yasunaga and Yi Chern Tan and Xi Victoria Lin and Suyi Li and Heyang Er and Irene Li and Bo Pang and Tao Chen and Emily Ji and Shreya Dixit and David Proctor and Sungrok Shim and Jonathan Kraft and Vincent Zhang and Caiming Xiong and Richard Socher and Dragomir Radev and, “SParC: Cross-Domain Semantic Parsing in Context”.

[16] GENSIM, “Word2vec embedding”, 2022, https://radimrehurek.com/gensim/models/word2vec.html, last access: May 2022

Appendix A: Development Platforms and Tools

In this appendix we will add the software tools used, a simple guide to use our website as well as a feasibility study for our product.

A.1 Software Tools

A.1.1 Programming Languages

- Main development language is python.
- Javascript used in UI.

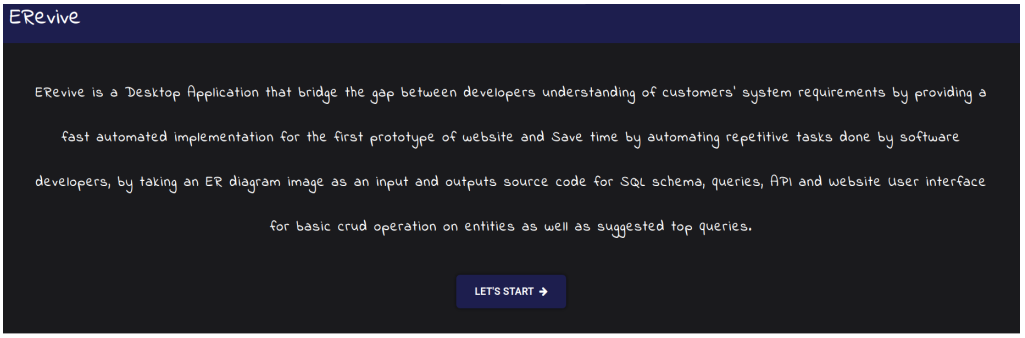
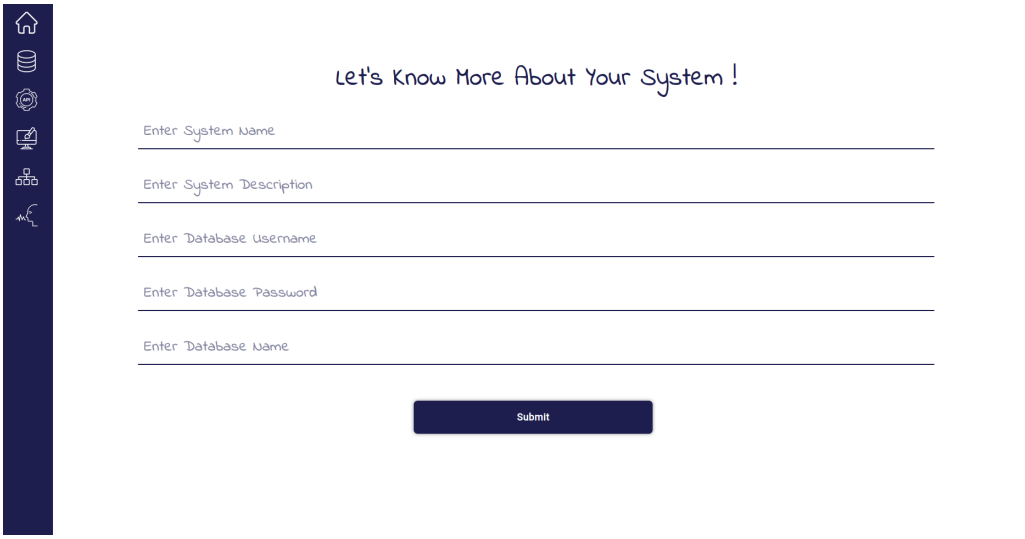
A.1.2 Libraries and Frameworks

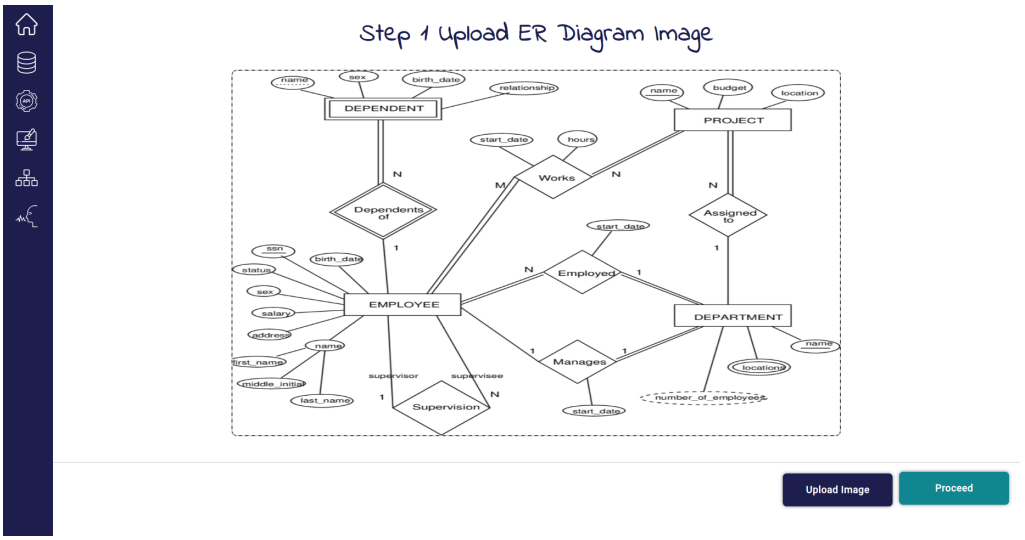
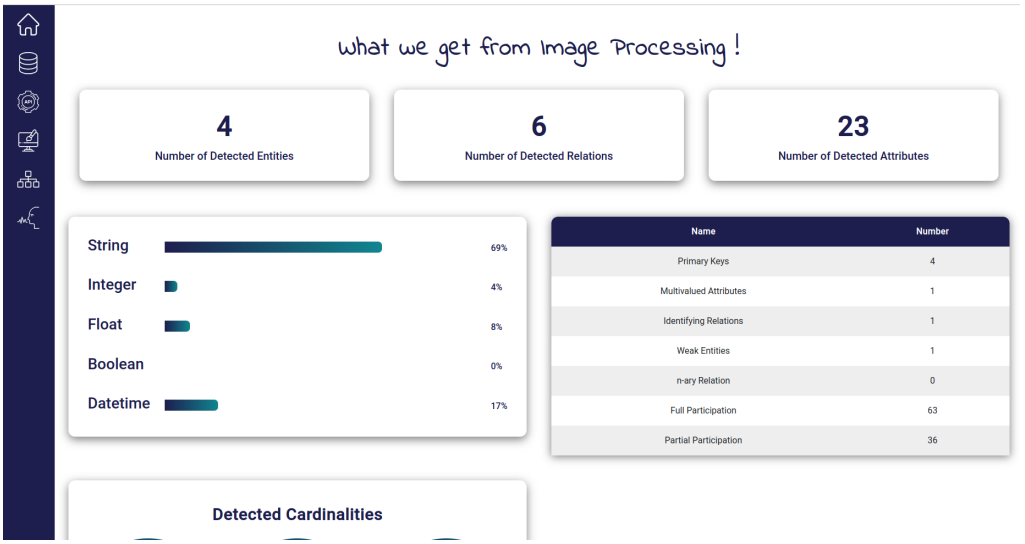
- Numpy supports large, multidimensional arrays and matrices.
- Vue is a front-end framework.
- Flask is a server framework.
- Opencv is an optimized image library.
- NLTK is a natural language processing library used in lemmatization, pos tagging, and word2vec model only.

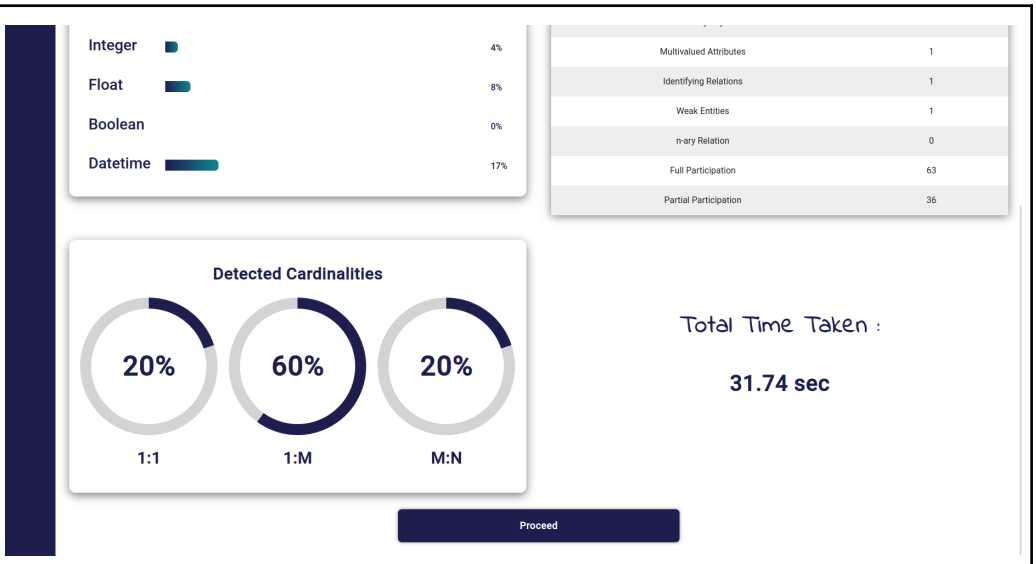
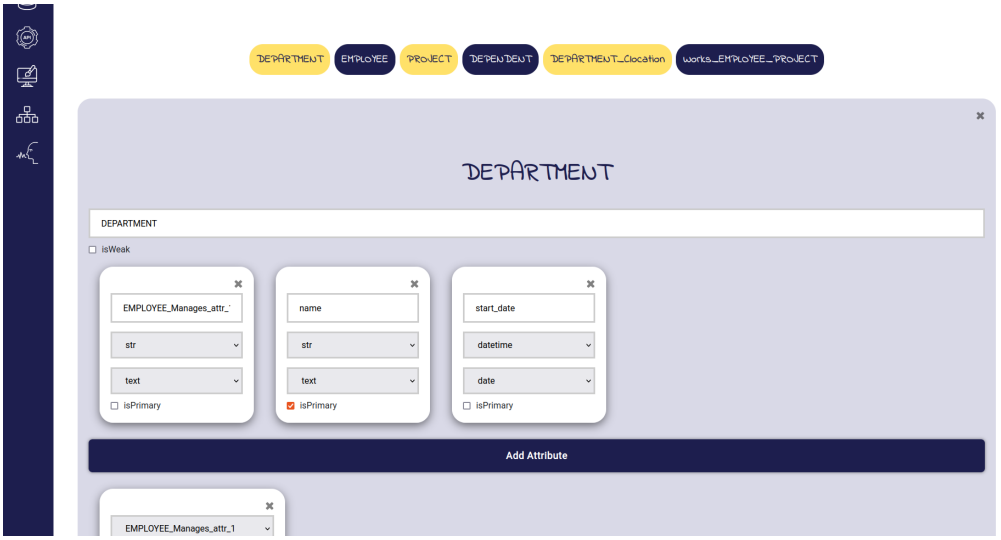
A.1.3 Tools and Platforms

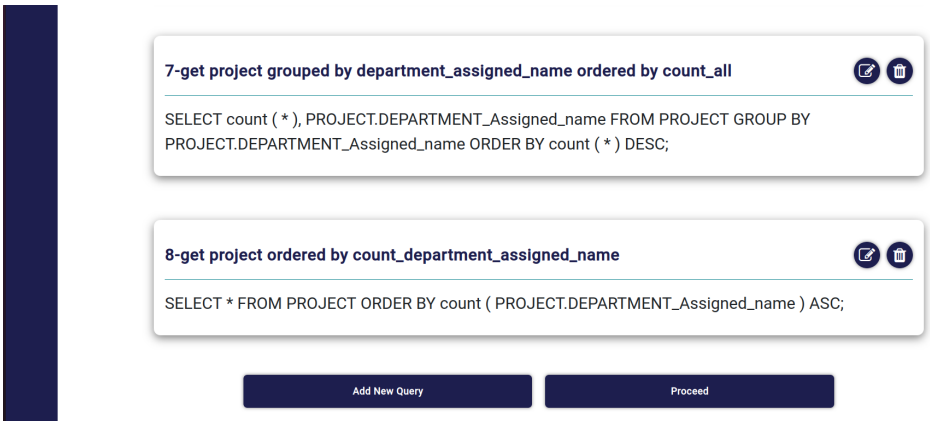
- Git is a version control system.
- GitHub: Repository hosting service.
- Visual studio code is a coding editor.

Appendix B: User Guide

On the home page press “Let’s start”	 *Figure B.1: The home page*
user should add system information and my SQL credentials	 *Figure B.2: System information page*

Upload Image and press proceed button	
	Figure B.3: Upload image page
Detected systems data statistics	
	Figure B.4: Image processing output page

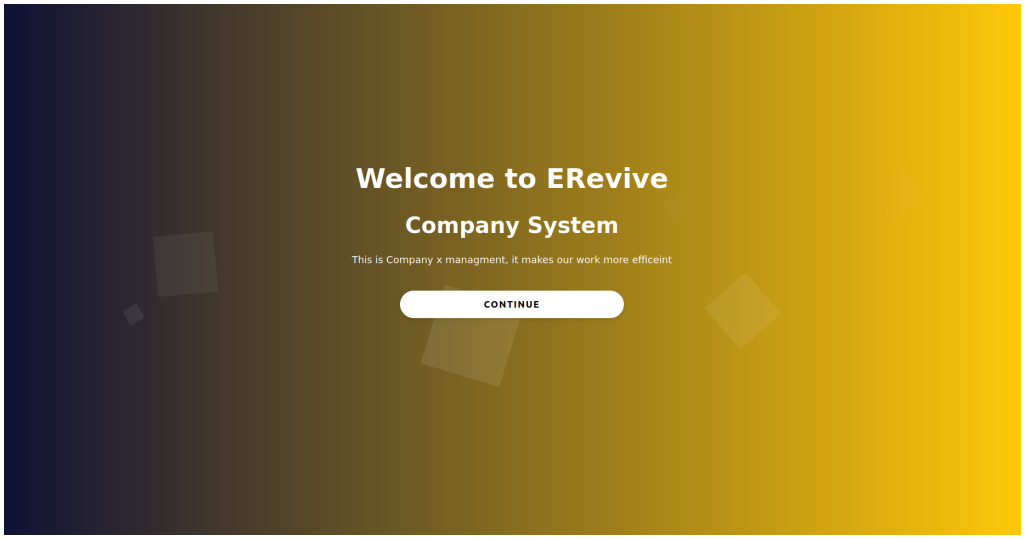
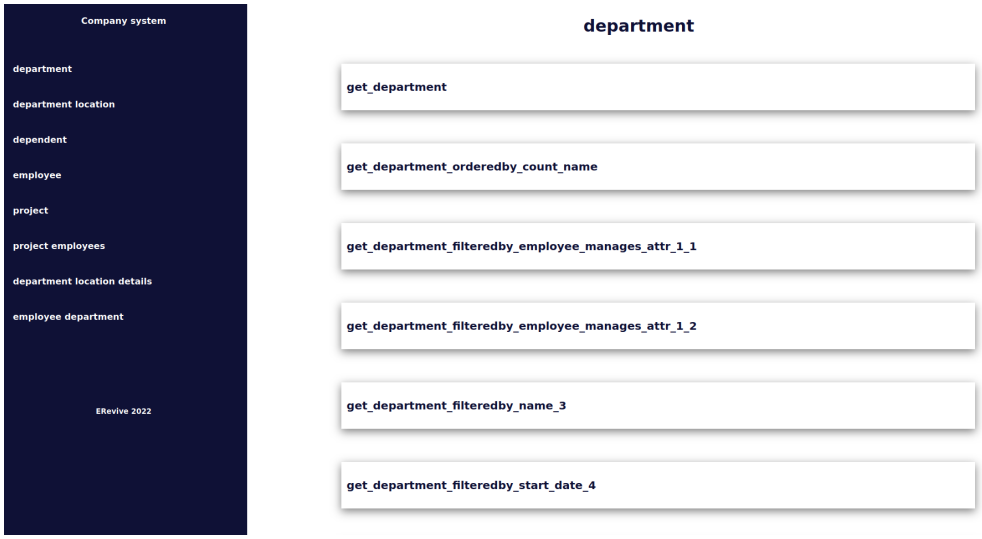
Press Proceed to start system validation	
	Figure B.5: Continue image processing output page
Users can change any names and add new entities, attributes, and Foreign keys or change existing Foreign keys. Press the Save Changes button to start generating queries	
	Figure B.6: Validation page

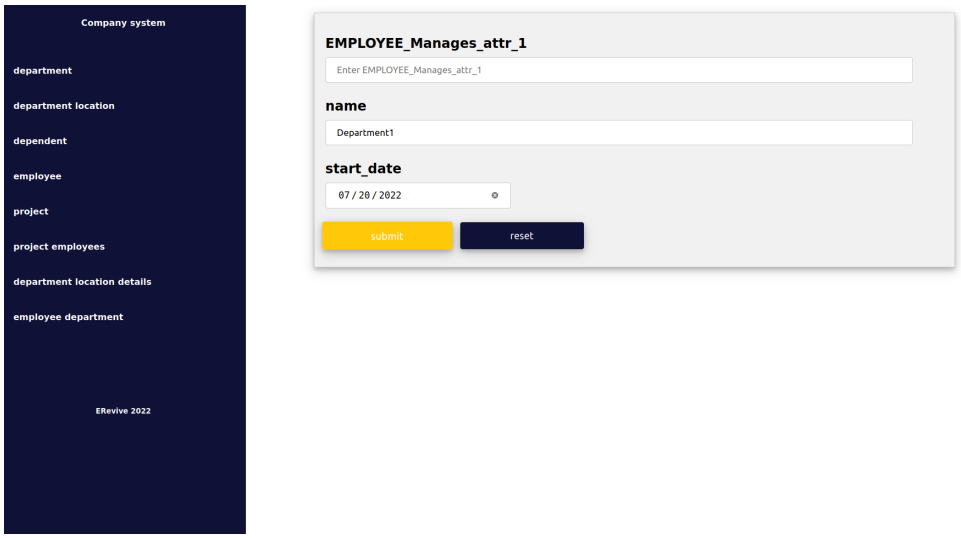
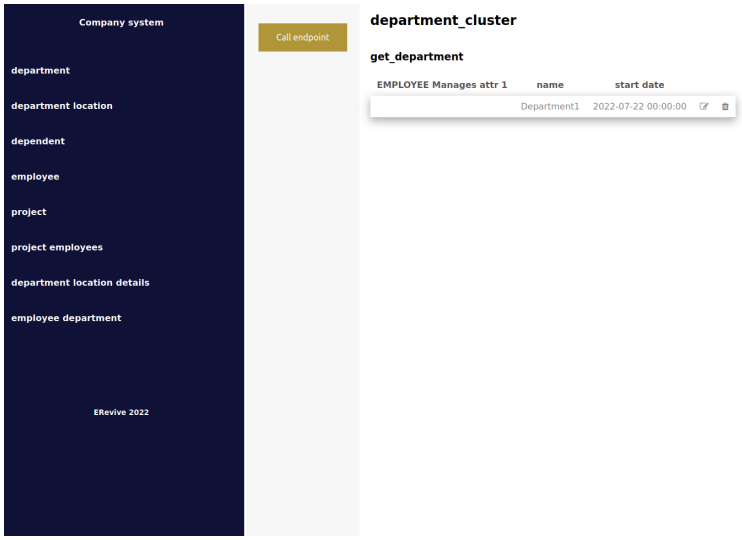
Clusters of different entity joins are displayed. users can view queries in this cluster by clicking on the cluster name	
	Figure B.7: Clusters page
User can view, edit, delete and add SQL suggested queries	
	Figure B.8: Queries for certain cluster page
User editing Query and saving changes	
	Figure B.9: Editing and deleting certain query within a cluster

User can Add Natural Language statements and have it converted to SQL query	
	Figure B.10: NL to SQL page
User can go back to the home page and add another ER diagram	
	Figure B.11: Last Page

SideBar Icons Guide

1. The first icon will move to the home page.
2. The second icon will move to the database seeds generation page.
3. The third icon will move to the API documentation page.
4. The Fourth icon will move to the generated UI application.
5. The fifth icon will move to the schema page.
6. The sixth icon will move to the NL to SQL page.

Generated website for the user	
	Figure B.12: Welcome page in the generated UI
Cards that navigate to pages each page consume API endpoint of a SQL query	
	Figure B.13: Showing queries for the certain cluster in the generated UI

Add new Object form	
	Figure B.14: Post and put the form in the generated UI
View objects in one or multiple entities. Users can click on edit or delete icons to edit or delete this object.	
	Figure B.15: Get the dashboard in the generated UI

Appendix C: Feasibility Study

In this section we will show the feasibility study of our project that is in form of the website allowing users to enter an image of the ERD system and then generating the required system with the most relevant queries and also providing source code for API and UI that are generated based on the given system while supporting the feature of converting from NL to SQL query.

C.1. Product Description

Our product is a website that takes as an input ERD image and performs some image processing techniques to extract information from the image which will be used in the generation of the initial schema, then we will allow the user to make some changes to this schema and to validate it in order to make sure that it is correct without any errors for further processing as we will take some keywords from the final schema then move to the search engine which will return a list of relevant queries that will be ranked from the most to the least relevant to the system, then we give the developer some flexibility in editing and deleting the generated queries before moving to the API and UI generation, moreover the developer has the ability to write some question in the NL to SQL module and his question will be automatically converted to the corresponding SQL query. The developer also has the ability to run the generated queries within the generated UI so that he/she can deal with the database which can be also dynamically generated through running the database seeds.

C.2. Technical Feasibility

For image processing and schema generation modules, knowledge of image processing techniques, established machine learning techniques for classification of shapes and datatypes, and algorithmic knowledge of techniques to extract information from image and consider it as a graph is sufficient to emphasize the feasibility of this module.

For the Query Suggestion module having multiple proposed approaches in similar fields such as code suggestion, search engines, recommendation engines, and SQL query suggestions on a known data schema is sufficient to indicate the feasibility of this module.

C.3 Product Marketplace

ERevive targets software developers and software companies that build websites that require dealing with SQL databases, to fasten this process and suggest proper queries that are necessary for the given systems.

C.4 Financial Feasibility

Cost for development is negligible, requiring only laptops for each team member.

C.5 Legal Feasibility

The development of the project depended on publicly available software tools and libraries to develop the code. Also, the datasets are distributed under licenses that permit the use of the creators' work publicly and our code will be publicly available and not intended for commercial use.