
# New Pmate BackOffice

Projeto de Informática

Licenciatura em Engenharia Informática

Departamento de Eletrónica, Telecomunicações e Informática

Universidade de Aveiro

21 June 2021

**Team:**

Pedro Souto, 93106

Diogo Moreira, 93127

Maria Inês Rocha, 93320

Luís Pereira, 93321

Fábio Andrade, 93406

# New Pmate Backoffice

Relatório de Projeto em Informática da Licenciatura em Engenharia Informática da Universidade de Aveiro, realizado por Diogo Moreira 93127, Fábio Andrade 93406, Luís Pereira 93321, Maria Inês Rocha 93320, e Pedro Souto 93106, sob a orientação de André Monteiro (ISCAA) e Joaquim Sousa Pinto (DETI), e com auxílio do Gonçalo Vieira (STIC).

**KEYWORDS**

Management Platform, Pmate BackOffice, Database Migration, External Authentication

**Abstract**

This project's objective is rebuilding an existing platform - Pmate's BackOffice, and remodelling its database. Pmate is a project initiated in 1989 that organizes numerous competitions every year for students all over the country and where any school is welcome to take part in, given that it is interested in providing this chance for their students to challenge themselves. This BackOffice is the management platform where administrators handle all the information regarding the exams, people and schools involved in the project. It was developed around 20 years ago and due to careless maintenance, is not suited for today's needs anymore - having features that are deprecated, a confusing UI and a need for a few new features. Similarly, the database in which the data that this platform manages is stored in, not only was poorly modelled without any respect for the relation model of SQL, but also accumulated a lot of unnecessary tables and attributes inside said tables. Given these circumstances, we will strive to mitigate all the problems described above and create a more user-friendly platform.

## Table of Contents

## List of Figures

# Introduction

**Context**

Pmate is a project that was initiated in 1989 with the promotion of the students' academic success in mind. Every year Pmate organizes competitions comprising exams covering numerous different fields such as: Physics, Mathematics, Languages, Science etc. These exams are based on multiple-choice questions and usually have a time limit of around 30 minutes. Students must strive to answer the questions correctly in the shortest time possible, since the score of each participant is not only calculated based on the number of correct answers, but also on the amount of time needed to answer them.

To manage Pmate the Pmate BackOffice platform was created with the purpose of being used by a small team of managers. The platform has many features, such as: searching for students and their respective tests; generating an html page with the results of a certain event; creating new tests or training games; registering schools in new events; among others. All of the information related to Pmate events is saved in a database that is connected to the Pmate BackOffice platform.

**Motivation and Goals**

The BackOffice platform has many issues. In its current state the platform does not fully support the team's needs. The current interface is hard to navigate and the team has to frequently access the database because there are some necessities the team has that are not covered by the current available features. Such is the case when it is necessary to make a user an administrator. That can't be done in the current back office, and so, direct access to the database is required.

The database that stores all the information regarding Pmate has many issues, such as missing references, un-encrypted passwords, and a non-relational model (No Foreign Keys or Primary Keys are being used).

This project consists of rebuilding the existing Pmate BackOffice platform and the remodelling of the database. Our goal is to improve the existing platform and make the job of the managers easier, make it faster and more reliable, while maintaining the main features and adding new ones.

**Organization**

In this report we will cover the following points:

- **State of the Art**: The existing work and gathering requirements.
- **Theoretical Analysis**: Presentation of the gathered requirements, personas and scenarios.
- **Implementation**: Details on the new platform and the new DB.
- **Results and Discussion**: Explanation of the obtained results and issues encountered.
- **Conclusion**: Present our conclusions on this project, and potential future work.

**Features implemented**

Some of the features that were implemented are: listing and creating new schools, training games and competitions, mass emailing by roles, generating html pages with the results of a competition, changing the roles of the users and many more.

# State of the Art

There are many examples of products with solutions similar to ours. After all, our product is just the management system behind the main application, and, bigger or smaller, depending on the necessities of each application, most have a separate website for their management that is to be accessed just by a small team of administrators. This being the case, the objective of our product is not to create an innovative application but to refactor both the management site and the database of an existing one.

**Existing Work**

The previous Pmate BackOffice platform was developed in ASP around 20 years ago with a SQL database where the data was stored. While relatively new at the time, most of the technologies used are now outdated and quite hard to work with. Considering the technological advances up to this day, and, consequently, more exigent standards of quality, the necessity to rebuild this system became more urgent. Thus, the need for this project with the rebuilding of both the database and the backoffice in mind.

While there may not have been many participants at the start, through the years Pmate's dimension has gotten bigger and bigger, which meant more data to store. This, in conjunction with not taking advantage of SQLs relational properties, due to the lack of primary and foreign keys, made the disorganization of the stored data bigger and harder to work with. Resulting in the need for this project.

# Theoretical Analysis

Our job consists of the refactoring of the existing Pmate BackOffice platform and remodelling the database. Removing unused features and adding new ones, while improving the data visualization with paginated results and search features were our main goals for the new platform. Regarding the database, our main goals were making it relational and simplifying the organization of the users by adding Roles to each user, while maintaining the data history.

One of the changes made to the BackOffice platform was the introduction of roles. The roles available currently are student, teacher and administrator. However, some new roles may be created dynamically if necessary. From these, the administrator role is the only one with access to the platform and all its features. This restriction was one of our project's main requirements considering the purpose of the BackOffice system.

**Requirements gathering**

Since our job was to refactor an old platform used by few people, we did not see a need for extensive requirements gathering along big groups of different people, our target user was homogenous. We based our requirements on the features of the old platform, new features requested by the advisors, while interacting with someone who is part of the STIC team, the team that works with the Pmate BackOffice platform daily, to better understand their issues with the current platform.

We knew from the start that we had to use a more modern technology, so our new BackOffice platform was developed using .NET Core MVC, a worldwide framework used by many large companies. It features a new user interface (UI) meant to clearly organize different features and improve usability. The new relational version of the old database, with all the old data still safely stored, was implemented with SQL.

**Functional Requirements**

These are the key features that already existed in the old platform:

- List past competitions.
- List the tests of a competition.
- Search for and recover the state of a test in a competition.
- Option to generate the HTML with the results of a competition.
- Register a school.
- List schools.
- Registration of schools in competitions.
- Associate a teacher to a school.
- Map view of the schools involved in Pmate.
- Create a new test or training game.
- Copy 'question models' from an older to a new test.

And there are also several new features implemented in the new platform:

- There are different roles with different privileges.
- Only possessors of the administrator role can access the BackOffice.
- Communication via email for a user or a target group of users based on their roles.
- Tab “Utilizadores” to list and manage the roles of the users: search users, select a user, and add or remove roles.
- External authentication with Google and Facebook.

**Non-Functional Requirements**

For the new platform there are some non-functional requirements that should be talked about:

Implementation

- The System should be developed in C# and HTML/CSS/JS using .NET Core MVC Framework.
- The System must be integrated with the new (re-modelled) SQL database.

Efficiency

- Loading data from the database to the interface should be fast.
- Updating the data from the database numerous tables should be fast.

Portability

- The system must be supported in various platforms.
- It should allow authentication from external services (i.e., Facebook, Google)

Security

- Users’ passwords must be encrypted.
- Two factor authentication should be possible.
- The platform shouldn't be accessible by normal users.

Usability

- The Navigation bar and Its menus must be simple and intuitive to use.
- The presentation of results must be paginated to make their reading easy and organized.

Legal

- When deleting an account, although some of users’ data may be kept for history registries purposes, any personal information is hidden (overwritten by anonymous tags)

**Personas**

- **Persona 1:**

Pedro Relvas is a 37 year old STIC technician that works with the Pmate BackOffice platform frequently. He is hard working, intelligent, organized and proactive. He knows the current platform very well, but recognizes its issues.

- **Persona 2:**

Leonor Santos is a 29 year old newly hired member of the STIC team, and has only recently started using the Pmate BackOffice platform. She works with schools participating in Pmate to handle any issues that might occur. She is dedicated, responsible and astute, but despite that she has been having problems understanding how to perform certain tasks on the platform.

**Scenarios**

- **Scenario 1:**

  Pedro (Persona 1) is creating exams and their respective question models. However, right now, the menus are not intuitive and to do that he has to search for information from the database and enter it manually. The newer version of the platform should facilitate this process.

- **Scenario 2:**

  Pedro (Persona 1) wants to send emails to all the participants or some specific group of users. The current platform is limited in this regard, so the renewed platform must implement a mass email feature.

- **Scenario 3:**

  Leonor (Persona 2) is contacted by schools whenever they want to register their students in some competition or when there's a teacher interested in some project. She's also responsible for generating the results of each competition.

  Currently, these tasks are made difficult by the disorganization of the platform. There are tabs with deprecated features and pages that are lists with lack of pagination and filters.

  As Leonor is new to the platform her job would be easier if the renewed platform were concise, intuitive, organized and with some helpful labels so that she doesn't spend too much time trying to accomplish simple tasks.

**Use Cases**

Here we can see some use cases that served as a guide through the development of the project. These have the purpose of showing the main tasks that should be available in our new platform, as well as the entities involved and the steps needed in each interaction.

- **Add/Remove user roles.**
    1. Go to the “Utilizadores” tab
    2. Find the user whose role you want to change (by searching or filtering, or otherwise)
    3. Click the pencil icon button on that user’s row.
    4. You will then be able to change that user’s role.

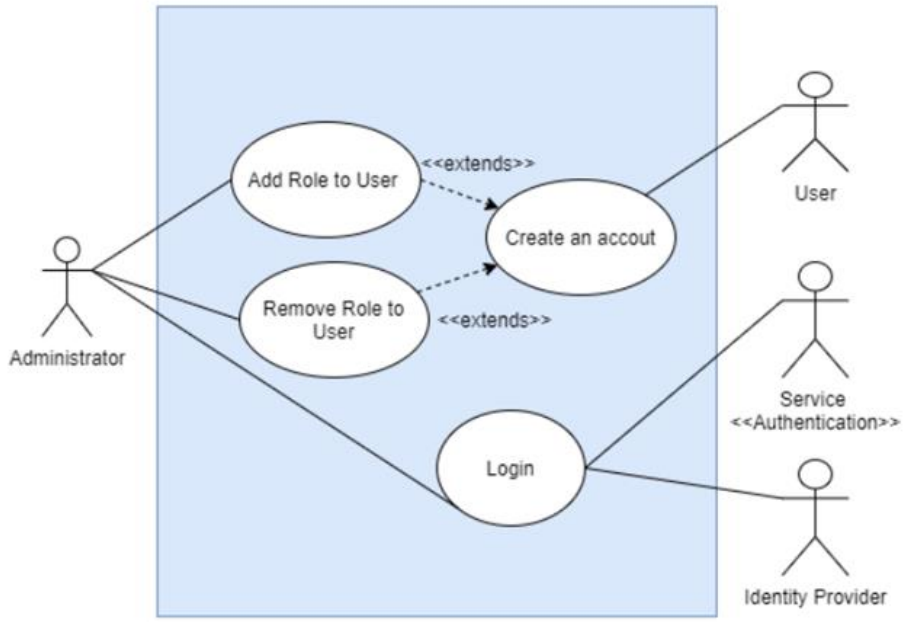



*Figure 1 - Use Case: Add/Remove user roles*

- **Send a mass email to a group of people.**
    1. Go to the “Mailing” tab.
    2. Choose a title and write an email message.
    3. Select the sender.
    4. Select who will receive it (other admins, teachers, students).
    5. Press ENVIAR.

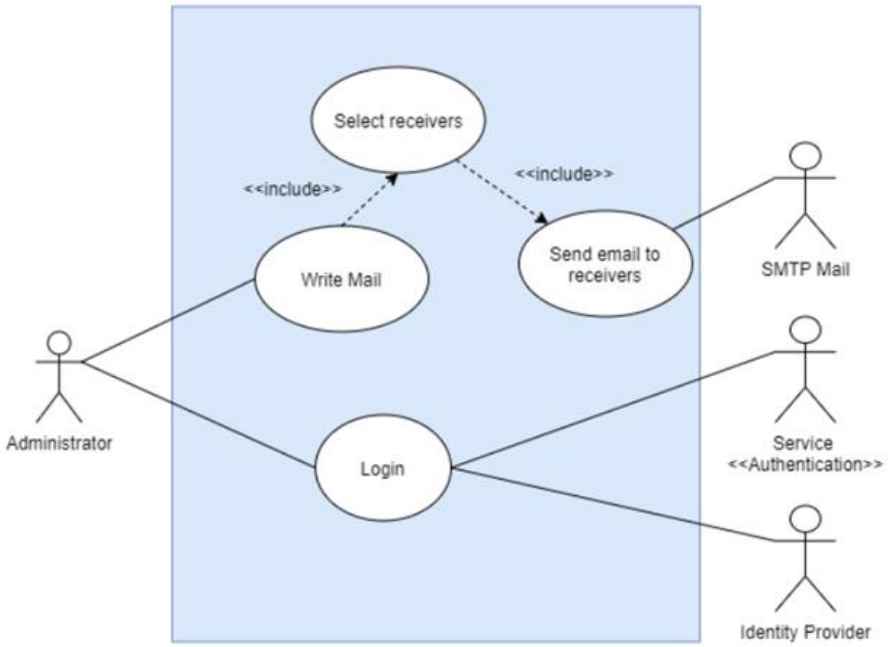



*Figure 2 - Use Case: Send a mass email*

- **Assign a School to a competition**
    1. Go to the COMPETIÇÕES tab.
    2. Choose the INSCREVER ESCOLAS option.
    3. Select the Competition.
    4. Select the schools you want to sign up, by checking them on the table.
    5. Press the INSCREVER button

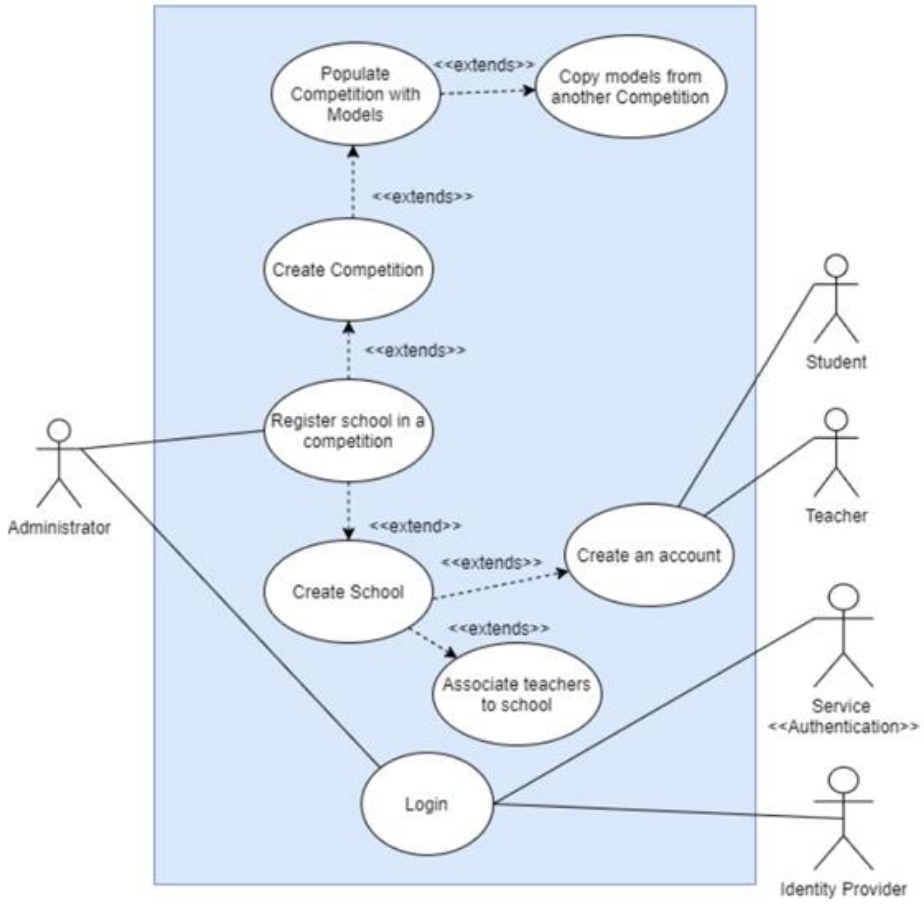



*Figure 3 - Use Case: Assign a School to a Competition*

- **Check the results of a competition.**
    1. Go to the "Competições" tab.
    2. Choose the “Listar Competições” option.
    3. Search or filter the competition you want.
    4. Click the plus icon button on that competition’s row.
    5. If the results are not generated yet, click the “Gerar Resultados" button to generate the results and view them.
    6. If they are already generated, click the “Ver Resultados” button to view the results.

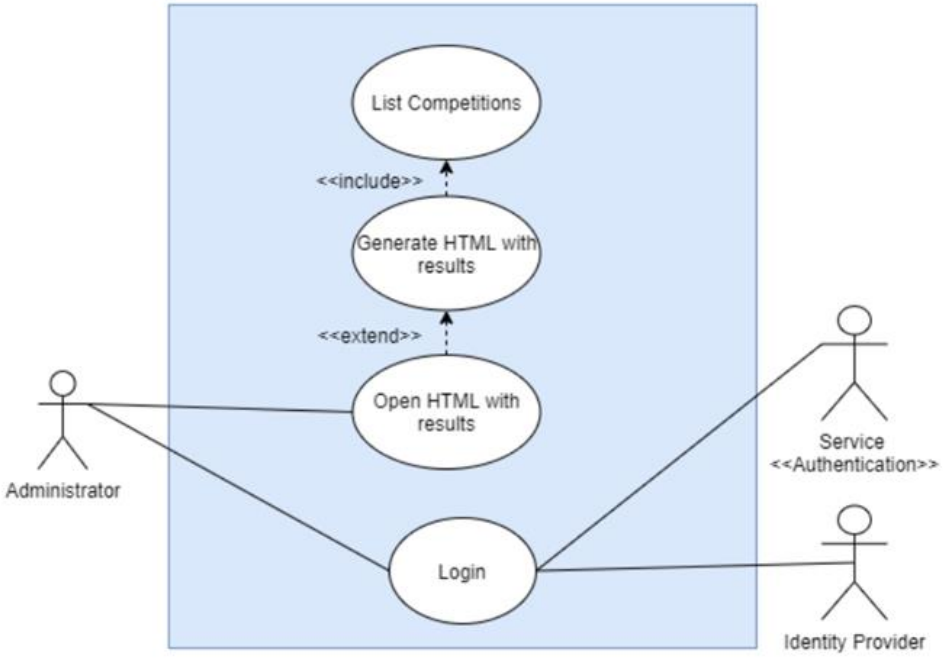



*Figure 4 - Use Case: Check the results of a Competition*

- **Create a Training Game**

1. Go to the “Treinos” tab.
2. Choose the “Criar Treino” option.
3. Fill out the form specifying the model for the questions, and other important information.
4. Submit.

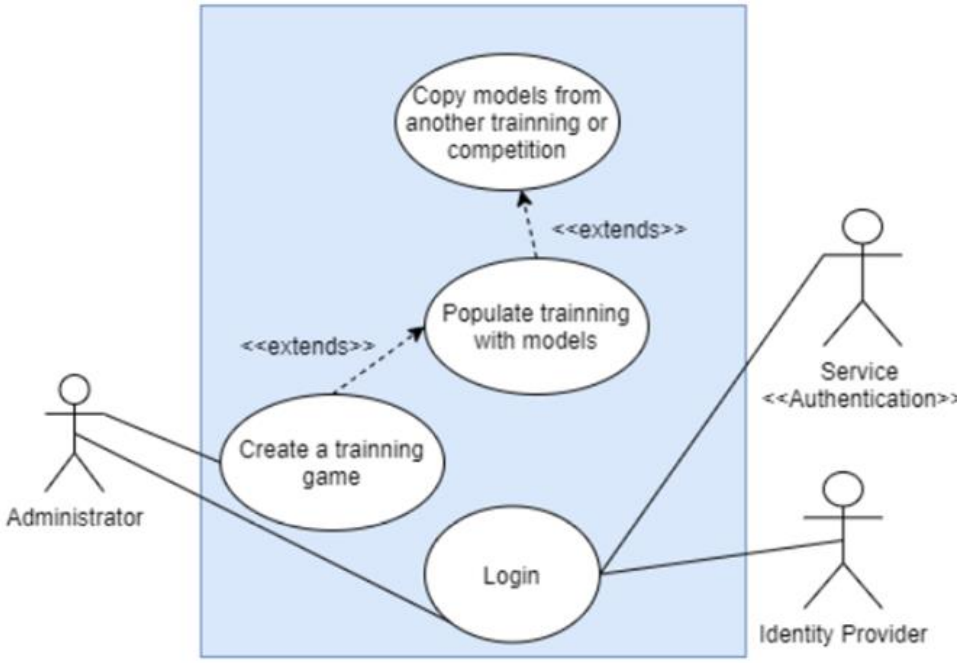



*Figure 5 - Use Case: Create a training game*

# Implementation

In the first phase of the development of this project we decided to assign different jobs to each of the team members: Pedro Souto as the team leader, Fábio Andrade as the DevOps Master, Diogo Moreira as the Lead Developer, Maria Inês Rocha as the Architect, and Luís Pereira as the Product Owner.

We also decided to opt for an Agile methodology, distributing the tasks to each member of the team and with a defined deadline. This was done weekly to ensure that all of the objectives proposed by the teachers were met in each iteration.

In the first iteration, after doing the gathering of the requirements in meetings with our advisors and further looking into the existing platform, we defined the architecture of the project and stipulated which external systems were going to interact with our system in different situations.

The creation and migration to a new database was influenced not only due to the necessity of establishing relations between the existing tables but also because of the existence of new ones generated automatically with .NET Core for the users management and their respective roles.

Furthermore, as we can see in the diagram represented in Figure 6, the email system was implemented using Google's email server and the authentication was done with the help of services like Facebook, Google and idp.ua.

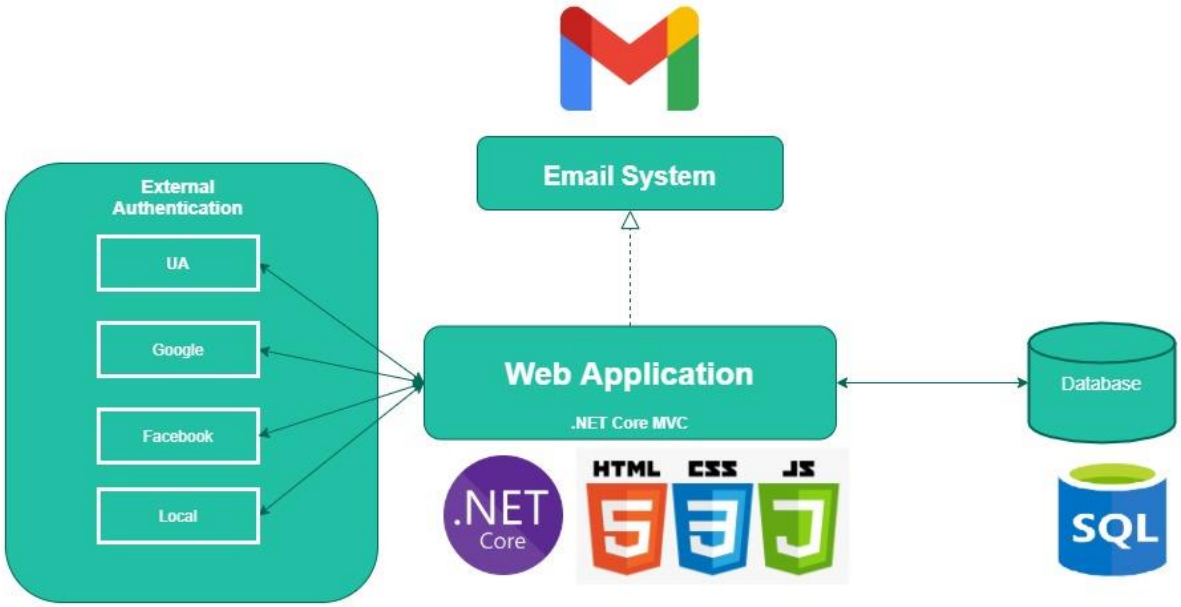



*Figure 6 - Architecture Diagram*

**Web Application**

The web application was developed using the .NET Core framework and is based on the Model-View-Controller architecture.

To generate the C# models for the entities in the database the command "Scaffold-DbContext" was used - we found this command when searching for a quick way to generate models while maintaining relationships between tables.

After a few tweaks to the models, we used the Entity Framework to create Controllers and Views with default implementations of CRUD (create, read, update, delete) operations for the relevant entities (Exams, Training games, Schools, Competitions). From this point, we adapted the views to only show relevant attributes for the 'create' and 'edit' forms.

We implemented a simple pagination for the lists of entities and added filtering options for each of them. We also added the option to delete an entity from the list page itself with a confirmation dialog to ensure the deletion is intentional and not a mistake.

While the views generated by Entity Framework were enough for the basic features, some features required custom views. We created functions in the controllers to assign Teachers to Projects, to list a certain Competition's Exams and the Sub-exams of larger Exams, to change the Role of a User, and add/remove Question-Models to Tests or copy them from another.

To call these functions from the views we used Ajax in conjunction with jQuery and javascript.

We implemented an option to generate, delete and download an HTML with the results of each exam. This feature uses the results page html of the old platform as a template. Whenever a new html is to be generated, all the html before the results table and all the html after the table is loaded into 2 variables. Then the results are loaded from the database into a third variable, and when the process is finished, all 3 variables are written to the new html file.

Furthermore, we implemented a map with all the schools in the database correctly positioned. This provides a more user-friendly way to access a School's information. The map was implemented using a javascript library called 'leaflet'. The coordinates for each school are stored in the database. We put these coordinates in a dictionary where the key is the region of the school, and in the map view we convert the dictionary to json, create a leaflet marker for each school, and add these Markers to the markerClusterGroup, which groups the markers into clusters making it easier to navigate the map. Upon clicking a marker the school's information is revealed.

**External Authentication**

We wanted to implement external authentication with Google, Facebook and idp.ua. Google and Facebook's way of doing this authentication is very similar. In both cases we had to add the NuGet Package and register it in the start-up file, register as developers so we could access the "developer's console", and had to create an application so that we could generate the ClientId and ClientSecret. After having this information, we used the .NET Core secret manager to save it in the application. In the actual program we coded the login buttons to have the following attributes: name='provider' and value='service', where 'service' is either Facebook or Google. When clicked these buttons redirect the user to the respective service's login.

**Email System**

The BackOffice allows administrators to send emails to any user on the database or to a specific group of users with the same role. For this feature we used Google's Simple Mail Transfer Protocol Server.

First, we define the credentials of the 'sender' email and some additional configurations in appsettings.json. Then we create an EmailModel class which validates the fields of the emails, an AuthMessageSender class which will use the SMTPClient class to send the email, and an EmailController to call AuthMessageSender. In order to send emails to all users of a certain role, we use .NET Core's userManager and loop through every user with that role and send them an email.

**Deployment**

The current BackOffice web application is running on an IIS server and the current database is running on an SQL server. However, after consulting with our advisors, we realized it wasn't feasible to deploy our new application on the same servers. So, we host both the new web application and the new database in a Virtual Machine:

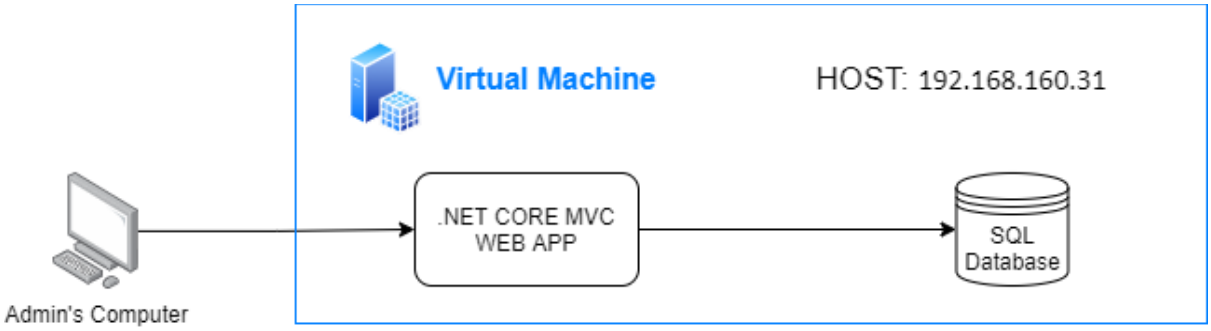



*Figure 7 - Deployment Diagram*

**Database Re-modelling and Migrations**

The old database was made in SQL, however, it didn't follow one of its main concepts. It didn't use SQL's Primary and Foreign Keys nor any other constraints that assure the database's consistency. Since it was not using the relational properties of SQL properly, it resulted in lack of control over the data, which translates to disorganization and unnecessary duplicated data. There were also many unnecessary attributes and even tables not being used that nobody bothered "cleaning".

We were initially provided with 50 tables. To solve all the problems found in them, we needed to plan the new database model very carefully, as we can see in the diagram in Figure 9 **(Appendix).** To define the new database skeleton, we had to study all the tables that were available, interpret the possible relation between them and decide the columns (attributes) to maintain in each one. Whenever we had doubts regarding information that was missing, we contacted our advisors to provide it. Finally, after exploring and migrating the relevant attributes from the old tables (that escalated from 50 to a total of 57) to the new ones, we ended up with 32 tables in our model, since the rest of them were deemed unnecessary.

The migrations took a long time since there was a lot of information that was duplicated or unused for no specific reason. Most of the migration scripts were trying to solve similar issues like: avoiding ID references to registries that were already deleted by using JOIN conditions, avoiding entirely duplicated rows that would violate primary key constraints by the use of the 'distinct' clause, and sometimes even JOIN clauses to the same table that was being migrated. This last condition was necessary because there were registries that were exactly the same except for differences of mere seconds in their timestamps. This meant that the distinct clause wouldn't be enough to prevent a primary key violation. Hence the necessity to join the table we were migrating, to a version of itself that was grouped by the primary key attributes and selected only the latest date for their timestamp.

Examples of the 3 most frequent problems described above are present in the script in Figure 8.

```
375  --------------------------  User-Escola
376  BEGIN TRANSACTION;
377  SET IDENTITY_INSERT pmate.UserEscola OFF
378
379  INSERT INTO pmate.UserEscola(IdUser, IdEscola,AnoLetivo, idProjeto, IdAnoEscolar, data_)
380  SELECT distinct [pmate-Equamat2000].dbo.tbluserescola.RefIdUser, RefIdEscola, RefAnoLectivo,Projecto,AnoEscolaridade,MaxDate
381  FROM [pmate-Equamat2000].dbo.tbluserescola
382  JOIN dbo.AspNetUsers ON CAST(RefIdUser AS nvarchar) = Id
383  JOIN pmate.Escola ON RefIdEscola= pmate.Escola.id
384  JOIN pmate.Projeto ON Projecto= pmate.Projeto.id
385  JOIN pmate.AnoEscolar ON AnoEscolaridade=pmate.AnoEscolar.id
386  JOIN pmate.AnoLetivo ON RefAnoLectivo=pmate.AnoLetivo.AnoLetivo
387  JOIN (
388      select RefIdUser, max(Data) as MaxDate
389      from  [pmate-Equamat2000].dbo.tbluserescola
390      group by RefIdUser
391  ) tm ON [pmate-Equamat2000].dbo.tbluserescola.RefIdUser = tm.RefIdUser and [pmate-Equamat2000].dbo.tbluserescola.Data = tm.MaxDate
392
393  COMMIT;
```

*Figure 8 - SQL Migrations*

**Testing**

As the BackOffice is a web application, we use Selenium, a browser-based testing framework, to create tests and make sure the menus and navigation are working as intended.

To test whether or not our new application was intuitive, efficient, and effective, we provided our new platform to the members of the STIC team alongside a questionnaire that they would fill out as they try to perform different tasks on the platform. The questionnaire asked how easy or hard they thought the task was to complete, how intuitive the platform was, and included a section for feedback on each task. The results of these questionnaires help gauge which features need refinement.

# Results and discussion

We were able to implement the key features we set out.

One of the biggest hurdles we faced was in restructuring and migrating data from the old DB to the new one. ...

TODO:

Organize the results of your work using tables and graphs, if possible.Accompany results with a meaningful discussion.Discuss possible sources of error and how accurate results are.

# Conclusion

TODO:

It should be a concise description of the project including its purpose and most important results providing specific quantitative information.

Explain what your project has achieved as well as the benefits and the shortcomings of your solution.

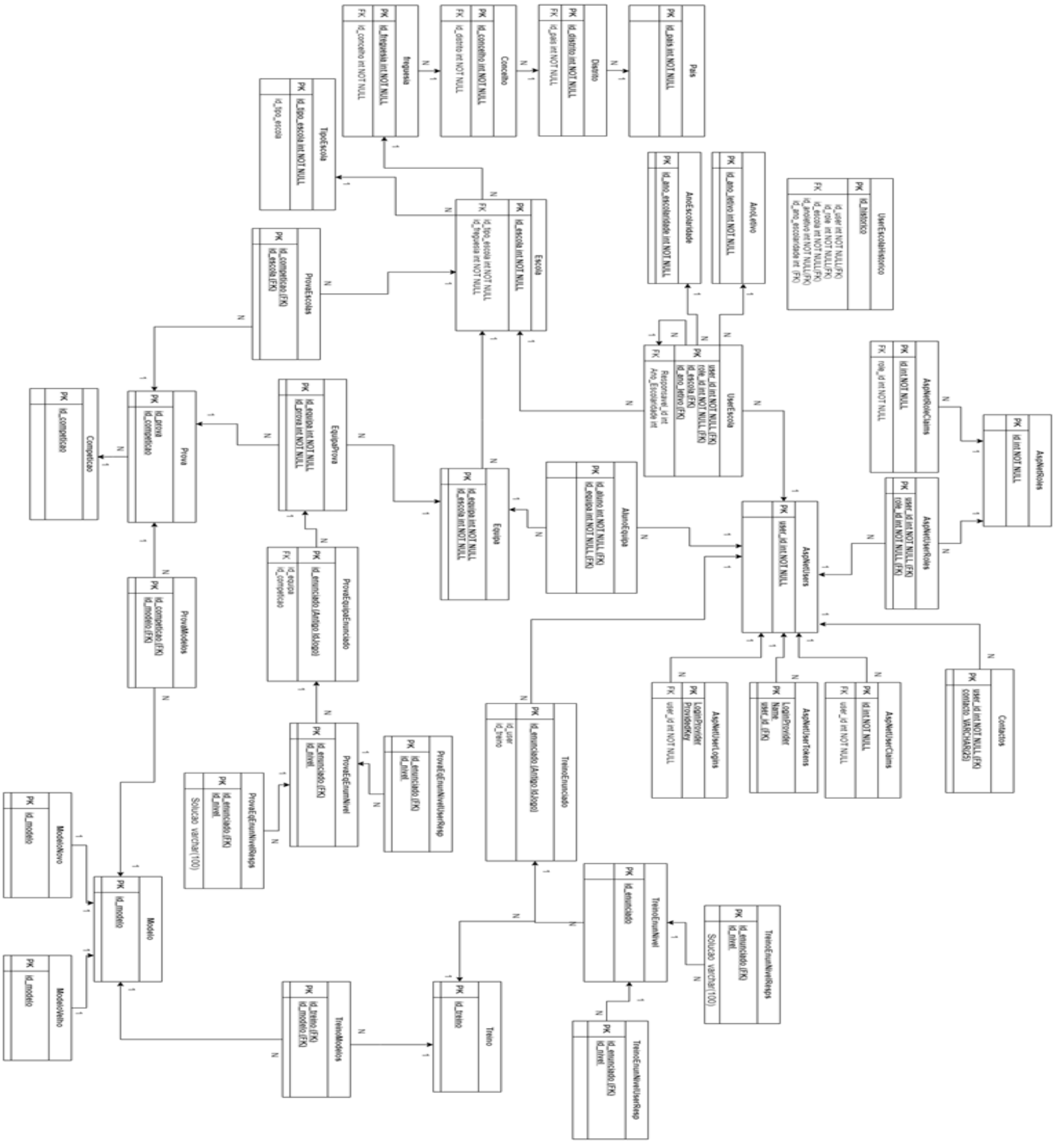
The reader should be able to read this chapter on its own.

Do not use specific terminology, abbreviations or acronyms and do not include figures and references to them. Suggest future works.

# References


1) https://www.yogihosting.com/aspnet-core-identity-roles/#testing
2) https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/sort-filter-page?view=aspnetcore-5.0
3) https://identity.ua.pt/oauth/
4) https://www.c-sharpcorner.com/article/calling-a-c-sharp-function-with-jquery-ajax-in-asp-net-mvc/
5) https://www.yogihosting.com/populate-cascading-dropdownlist-with-ajax/
6) https://forums.asp.net/t/2153487.aspx?Populating+a+select+box+depending+on+another+selecting+using+a+Model
7) http://www.macoratti.net/18/04/aspcoremvc_email1.htm
8) http://www.binaryintellect.net/articles/e30d07c6-6f57-43e7-a2ce-6d2d67ebf403.aspx
9) https://www.aspforums.net/Threads/545172/Pass-Send-multiple-Lists-from-Controller-to-View-in-ASPNet-Core-MVC/

# Appendix


Pais
id_pais int NOT NULL
Distrito
id_distrito int NOT NULL
id_pais int NOT NULL
Concelho
id_concelho int NOT NULL
id_distrito int NOT NULL
Freguesia
id_freguesia int NOT NULL
id_concelho int NOT NULL
TipoEscola
id_tipo_escola int NOT NULL
id_tipo_escola
Escola
id_escola int NOT NULL
id_tipo_escola int NOT NULL
id_freguesia int NOT NULL
AnoLetivo
id_ano_letivo int NOT NULL
AnoEscolaridade
id_ano_escolaridade int NOT NULL
UserEscolaHistorico
id_historico
id_user int NOT NULL(FK)
id_role int NOT NULL(FK)
id_escola int NOT NULL(FK)
id_anoletivo int NOT NULL(FK)
id_ano_escolaridade int (FK)
UserEscola
user_id int NOT NULL (FK)
role_id int NOT NULL (FK)
id_escola (FK)
id_ano_letivo (FK)
Responsavel_id int
Ano_EscolaridadeId int
AspNetRoles
id int NOT NULL
AspNetRoleClaims
id int NOT NULL
role_id int NOT NULL
AspNetUserRoles
user_id int NOT NULL (FK)
role_id int NOT NULL (FK)
AspNetUsers
user_id int NOT NULL
Contactos
user_id int NOT NULL (FK)
contacto VARCHAR(25)
AspNetUserClaims
id int NOT NULL
user_id int NOT NULL
AspNetUserTokens
LoginProvider
Name
user_id (FK)
AspNetUserLogins
LoginProvider
ProviderKey
user_id int NOT NULL
AlunoEquipa
id_aluno int NOT NULL (FK)
id_equipa int NOT NULL (FK)
Equipa
id_equipa int NOT NULL
id_escola int NOT NULL
EquipaProva
id_equipa int NOT NULL
id_prova int NOT NULL
ProveEscolas
id_competicao (FK)
id_escola (FK)
Competicao
id_competicao
Prova
id_prova
id_competicao
ProvaModelos
id_competicao (FK)
id_modelo (FK)
ProvaEquipaEnunciado
id_enunciado (Antigo Id Jogo)
id_equipa
id_competicao
ProvaEqEnumNivel
id_enunciado (FK)
id_nivel
ProvaEqEnumNivelResp
id_enunciado (FK)
id_nivel
Solucao varchar(100)
ProvaEqEnumNivelUserResp
id_enunciado (FK)
id_nivel
TreinoEnunciado
id_enunciado (Antigo Id Jogo)
id_user
id_treino
TreinoEnumNivel
id_enunciado
TreinoEnumNivelResp
id_enunciado (FK)
id_nivel
Solucao varchar(100)
TreinoEnumNivelUserResp
id_enunciado (FK)
id_nivel
Treino
id_treino
TreinoModelos
id_treino (FK)
id_modelo (FK)
Modelo
id_modelo
ModeloNovo
id_modelo
ModeloVelho
id_modelo