

**Project Report**

**Datenbanken und Web-Techniken**

**Summer Semester 2021**

**School Grading System Web Application**

**Mohammad Tanvir Islam**

**Matriculation # 676614**

**Master in Automotive Software Engineering**

Contents

[1. Introduction 3](#_Toc76298143)

[2. Project: School Grading System 3](#_Toc76298144)

[3. Technology Used 3](#_Toc76298145)

[4. System Requirement 4](#_Toc76298146)

[5. Designs 4](#_Toc76298147)

[4.1 Use Case 4](#_Toc76298148)

[4.1 Application Layer Pattern 4](#_Toc76298149)

[4.2 Entity Relationship Diagram (ERD): 5](#_Toc76298150)

[4.3 Stored Procedures 6](#_Toc76298151)

[6. References 9](#_Toc76298152)

# Introduction

To gain a better understanding of test results and student performance, a school would like to switch from a paper to a digital grading system. It should also encourage instructors and students to communicate with one another (and their parents).The new system should have a good user rights management system and a lot of options for users.

The major purpose will be to provide teachers with an easy and quick way to record the results of all of their students' tests, as well as to provide an overview of tests and subjects for both students and teachers, with some rudimentary export functionality. A rudimentary communication messaging system should also be developed. A detailed task description can be found on the following pages.

# Project: School Grading System

A backend capable of aggregating and processing data is one of the project's key requirements. The aggregated data is stored in a database during processing and can later be seen by the end-user via an interactive interface. The frontend uses a REST-based interface to ingest data from the data store. The situation is even better shown in the diagram below.

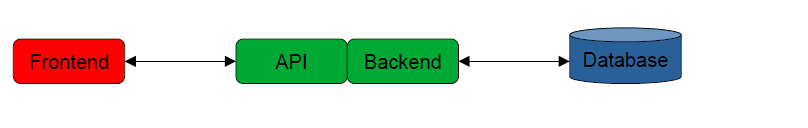


Figure-1: Project Structure

The program is made up of four parts:

1. A database for storing data.

2. A backend for processing data.

3. An API-based application programming interface for delivering data.

4. A frontend for showing data.

# Technology Used

* Oracle Database, Stored Procedures
* ASP.NET, WebApi 2
* Nodejs [14.17.0]
* Npm [7.17.0]
* JSON
* Angular [12.1.0]
* JavaScript
* JQuery
* Bootstrap: [5.0.2]
* HTML

# System Requirement

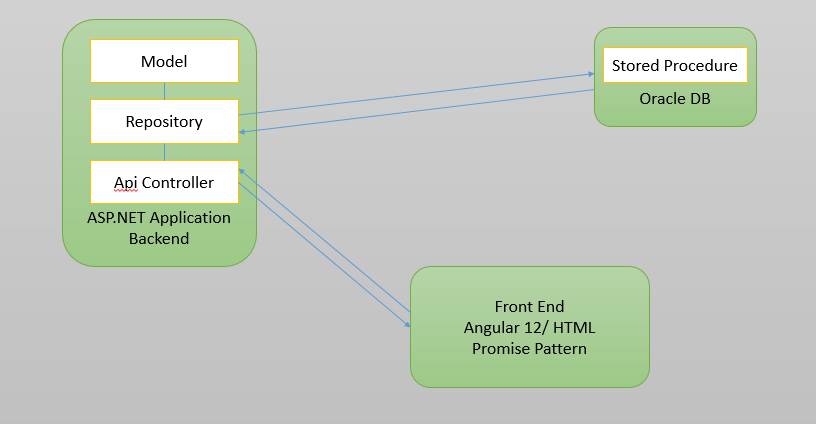
* Oracle 12c Database
* Minimum 1GB Ram
* Windows 8/10 64 Bit OS

# Designs

## 5.1 Use Case

A use case portrays how a client utilizes a framework to achieve a specific objective. A use case chart comprises of the system, the related use cases and actors and related to these to one another to picture: what is being depicted? (System), who is utilizing the framework? (Actors) and what would the actors like to accomplish? (Use cases), subsequently, use cases help guarantee that the right framework is created by catching the necessities from the client's perspective.

## 5.2 Project Architecture

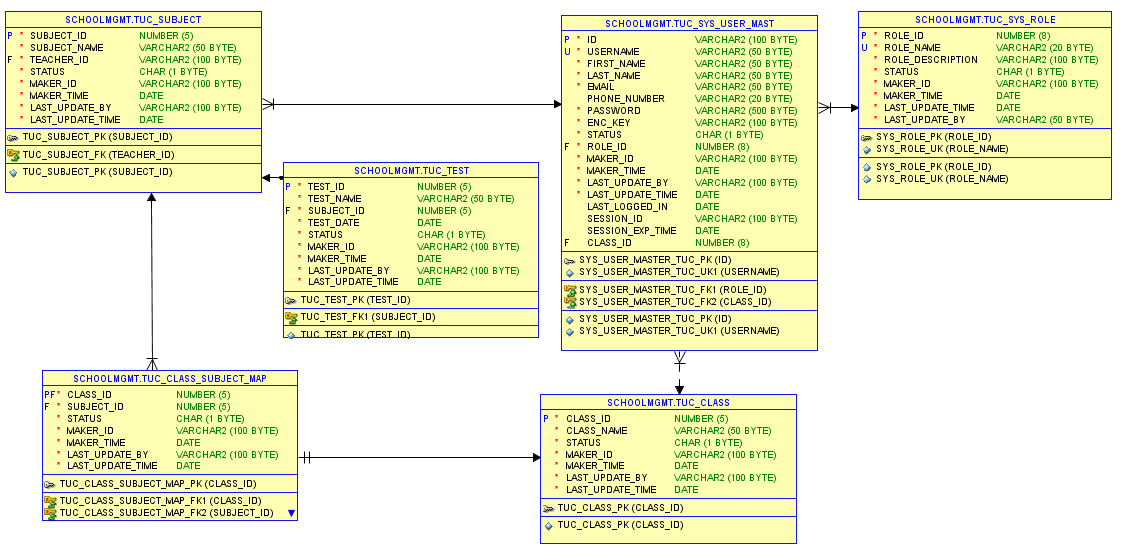
 Figure-2: Project Structure

The above diagram depicts the **M**odel, **R**epository and **C**ontroller to the AP.NET high level architecture. The controller is an Api Controller implemented with a Asp.Net WebApi 2, the model is implemented with Repository, which calls database stored procedure, and in stored procedure, all business logics are implemented.

This is the architectural overview of the Asp.Net WebApi2 Repository pattern. We will go through each component in more detail in the subsequent chapters.

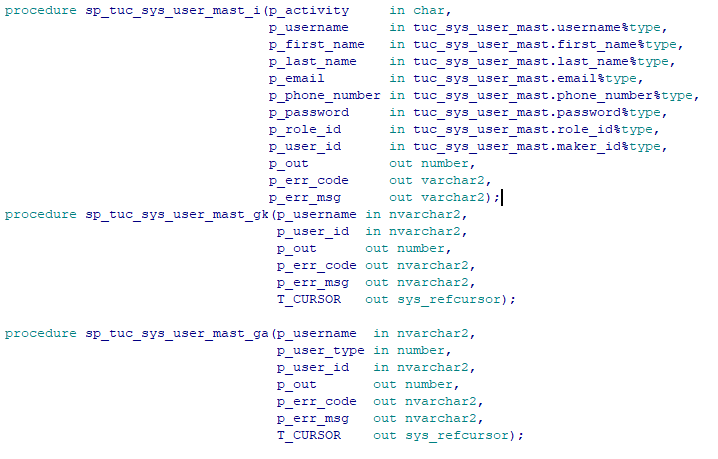
## 4.2 Entity Relationship Diagram (ERD):

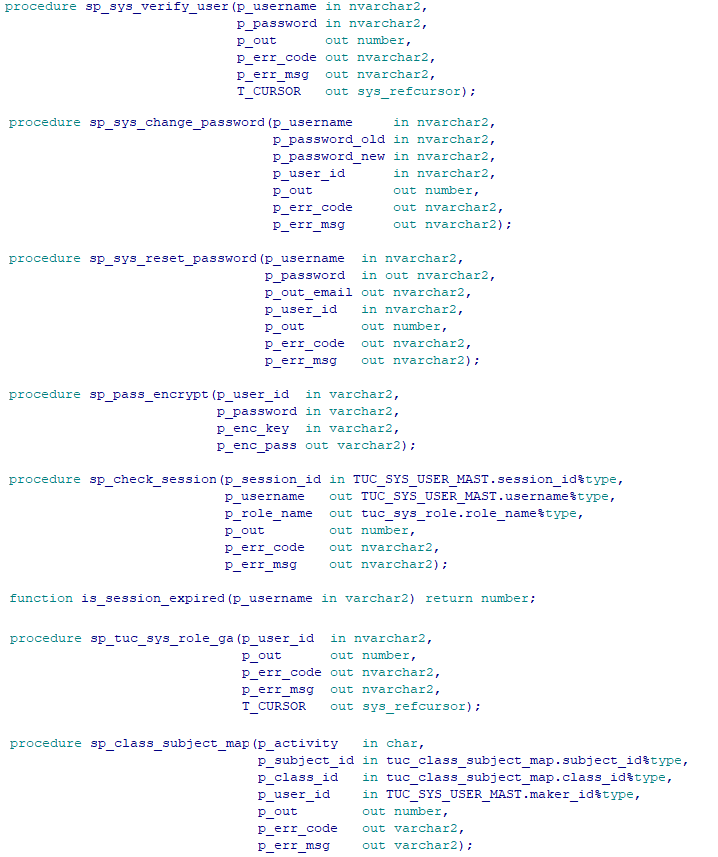
The Entity Relationship Diagram (ERD) or Entity Relationship Diagram (ERD) is a type of structural diagram used in database schema. The Entity Relation model is based on the concept of major actual elements inside the system scope and their relationships.

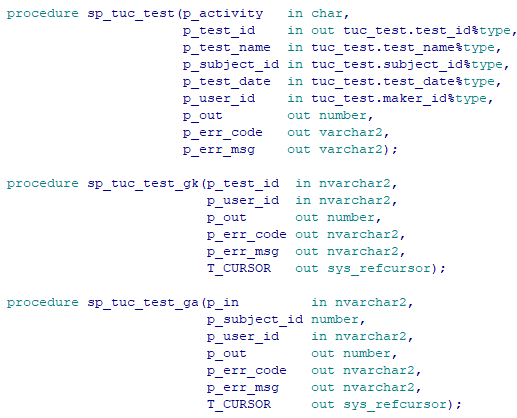


## 4.3 Stored Procedures

The PL/SQL language of the Oracle database consists of stored procedures used to create applications in databases. IT specialists write and test code by utilizing stored programs in the Oracle database, which are then compiled into stored methods. The fundamental structure of the PL/SQL block, including declarative, executable and exception handling components, is followed by a stored procedure in Oracle.







# References

1. Richter D. (2021. May 31). “DBW Project” [PDF File]. Retrieved from <https://www.tu-chemnitz.de/informatik/DVS/lehre/DBW/Projekt/DBW%20Project.pdf>
2. Richter D. (2021. June 25). “Questions and Answers concerning project task in Datenbanken und Web-Techniken 2021” [Text File]. Retrieved from <https://www.tu-chemnitz.de/informatik/DVS/lehre/DBW/Projekt/DBW%20Project%20QA.txt>
3. OracleDatabase. (2021, June 28). Retrieved from <https://en.wikipedia.org/wiki/Oracle_Database>
4. What is NodeJS? Executive Summary. (2021, June 28). Retrieved from <https://nodejs.org/en/about/>
5. What is angular? (2021, June 28). Retrieved from <https://angular.io/guide/what-is-angular>
6. What is RESTful ?. (2021, June 28). Retrieved from <https://restfulapi.net/>
7. What is ASP.NET WebApi2? (2021, June 28). Retrieved from <https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api>
8. jQuery. (2021, June 28). Retrieved from <https://jquery.com/>
9. What is Stored Procedures?. (2021, June 28). Retrieved from <https://searchoracle.techtarget.com/definition/stored-procedure>
10. ASP.NET-CORS. (2021, June 28). Retrieved from <https://www.c-sharpcorner.com/article/enable-cors-in-asp-net-webapi-2/>
11. Requests: HTTP for Humans™. (2021, June 28). Retrieved from <https://requests.readthedocs.io/en/master/>
12. Bootstrap (front-end framework). (2021, June 28). Retrieved from <https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)>
13. Font Awesome (front-end framework). (2021, June 28). Retrieved from <https://en.wikipedia.org/wiki/Font_Awesome>
14. DataTables | Table plug-in for jQuery (2021, June 28). Retrieved from <https://datatables.net/>

# Appendix

**API Documentation**

|  |  |
| --- | --- |
| **RESOURCES** | **END POINTS** |
| Login | POST api/Login |
| User | POST api/User/Register |
| POST api/User/UpdateUser |
| GET api/User/GetUserList?userType={userType} |
| GET api/User/Get?id={id} |
| Class | GET api/Class/GetClassList |
| GET api/Class/GetClassInfo?id={id} |
| POST api/Class/ModifyClassInfo |
| POST api/Class/AddNewClass |
| DELETE api/Class/DeleteClass?id={id} |
| Subject | GET api/Subject/GetSubjectList |
| GET api/Subject/GetSubjectInfo?id={id} |
| POST api/Subject/ModifySubjectInfo |
| POST api/Subject/AddNewSubject |
| DELETE api/Subject/DeleteSubject?id={id} |
| POST api/Subject/ClassSubjectMap |
| DELETE api/Subject/ClassSubjectMapRemove?classId={classId}&subjectId={subjectId} |
| Test | GET api/Test/GetTestList?subjectId={subjectId} |
| GET api/Test/GetTestInfo?testId={testId} |
| POST api/Test/ModifyTestInfo |
| POST api/Test/AddNewTest |
| DELETE api/Test/DeleteTest?id={id} |