|  |  |
| --- | --- |
| COMP-1687 Web Application Development | Madalin Cristian Preda 000937119 |

Table of Contents

[1 – PART A 2](#_Toc32913273)

[1.1 Group Work Introduction 2](#_Toc32913274)

[1.2 Entity Relation Diagram (ERD) 3](#_Toc32913275)

[1.3 Architecture Diagram 4](#_Toc32913276)

[1.4 Individual Report 5](#_Toc32913277)

[2 – PART B 5](#_Toc32913278)

# 1 – PART A

## Group Work Introduction

Group work done with Gabriel Ciortea-Pop, 000968052

For the group was not the first time working together, and pair programming was a developing strategy already adopted in the past. Both members have always been keen to work together when facing new challenges as it speeds up the learning process. When pair programming the quality of the code also increases, and the developed solution becomes more robust. Besides, pair programming enables both team members to have an excellent understanding of the code as it requires engagement from both the driver and navigator.

The group started the assignment by designing the entity relation diagram (ERD), which enabled the group to define the objects requiring abstract representation. After completing the initial ERD, it was decided not to use the university hosted SQL server as it would give less flexibility but to take advantage of the Microsoft Azure donation for students and to deploy a SQL Server instance on the cloud which would allow us to work from our devices from home or at university comfortably. To create an entity/model, the group decided to scaffold the database and use Entity Framework.

The use of Entity Framework extremely simplified the database management and enabled to quickly be able to develop a desktop application to perform create, read, update and delete (CRUD) operation for the employees. The entity framework scaffold was implemented as a separated project in the solution, and it enabled to require the entity/model on other projects in the same solution by adding a reference and the connection string the config file. This allowed updating the model in one point for all the projects each time the database was altered.

To handle the numerous windows required by the application, the group decided to take advantage of Winforms “User Controls” (UC) which enables to create views inside a single form dynamically. Each UC is controlled by a singleton pattern which restricts the application to only have a single instance of each user control at runtime. The advantage of the approach is to save memory and avoid a large number of user views instances. The approach has also enabled modulation of the front-end. The group has also decided to create two components: a new TextBox for password management, which inherits the properties of the standard TextBox but masks the input and enforces minimum and maximum chars. Also, a dynamic button was created, which was reused for different CRUD operations.

The group acknowledges that it is possible that the database and the design could further change, the trial and failure approach adopted is also a critical element in the learning process and what appeared correct in the initial analysis could turn out not to be ideal.

## Entity Relation Diagram (ERD)

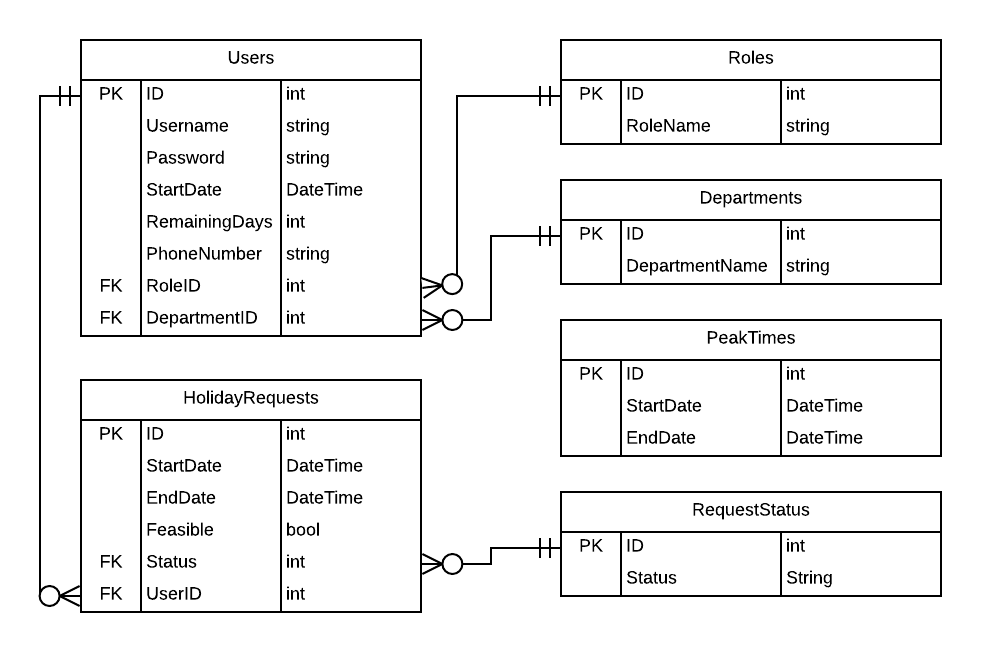


Figure 1 Initial Entity Relationship Diagram (ERD)

The above ERD diagram is the result of the group effort. The diagram was kept as simple as possible, for example users do not have a name or surname, but just their username which is a unique field. Additional details such as age, sex, address are not considered in the scope of this project. The main relationship is between Users and Holiday Requests, while the other linked table are simple look up table to manage a user role and department and the Request status for a Holiday Request.

PeakTimes is a table needed only to record the holiday requests peak time, it has no relationships and the purpose is to allow the admin users to edit the peak periods.

The group agrees that it is possible that the implementation could change while developing part B.

## Architecture Diagram

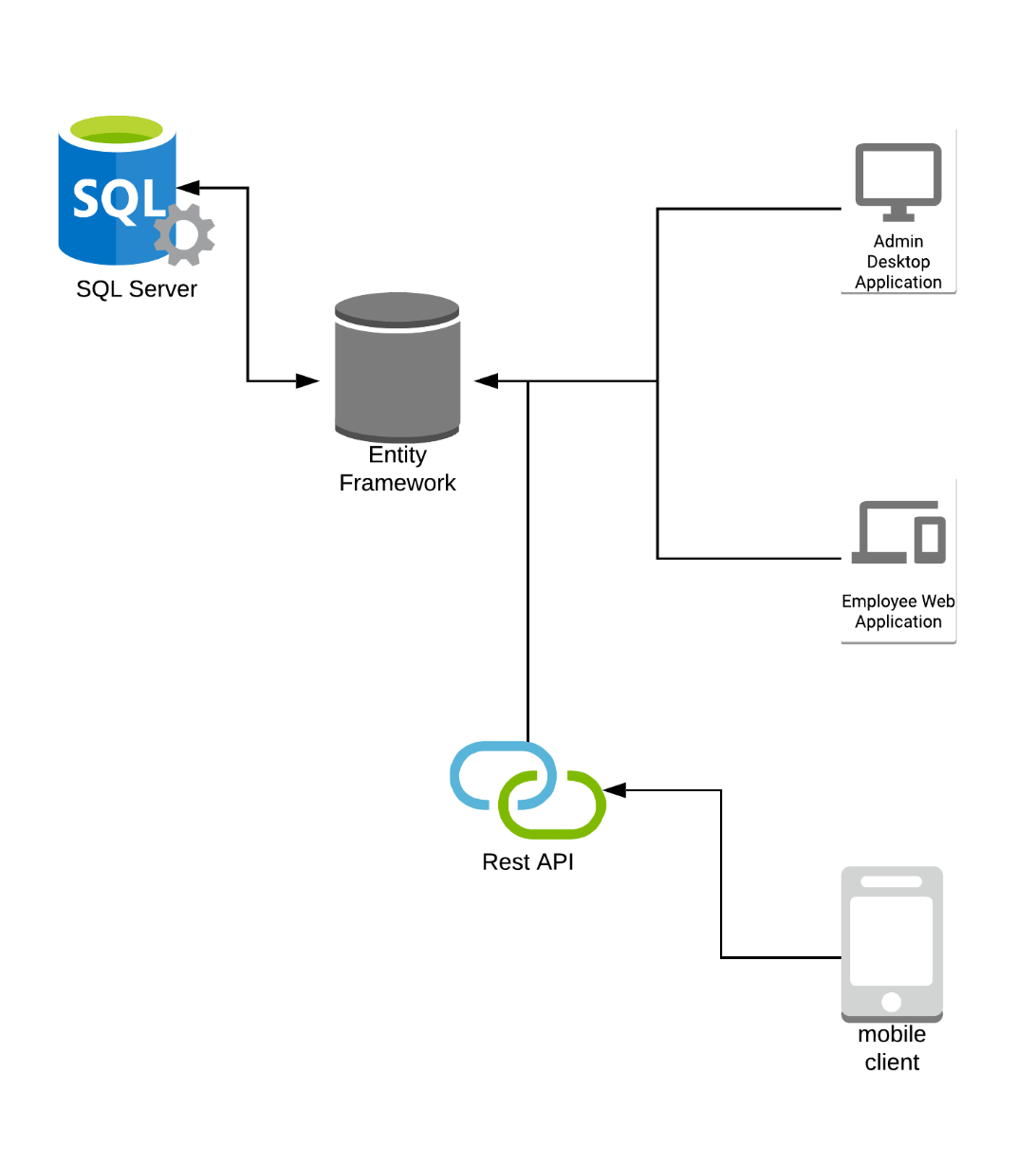


Figure 2 Initial envisioned architecture

The above diagram represents the architecture of the holiday booking system application. There is a SQL Server database stored on the cloud (MS Azure) and all operations are handled by the entity framework projects, which in fact operates as the middleman for all the other components in the diagrams. The Rest APIs will also provide a communication interface for the mobile client(s).

## Individual Report

Gabriel and I worked previously in many different projects since year one, thus each of knew how to each other’s strengths and weaknesses. I strongly believe that we had a great team dynamic and managed to provide a good solution to the required tasks. As we both previously worked with C# we concluded that we still have different skillsets, as Gabriel did projects with windows forms before he clearly did a better job than I could have done on putting the different User Interface(UI) components together, and myself, I tried to add a bit of a styling so that the fonts and colors are nicer.

We worked on everything together, database, backend and frontend, I assissted with the database design and building it. Additionally, I helped at sampling the solution structure together as we managed to scaffold the database into an ADO.NET Entity Model and linked it to both Manager Windows Form Login and Employees ASP.NET Forms web application. Additionally, I created the backend seed queries for populating the database with an admin and some default users, also worked on exception handling, and queries optimization as Gabriel had not use Entity Framework before to the extent I did.

Personally, I believe that our team dynamic did not disappoint, and we did good on pair programming, he was the main person typing and I was the one researching how to do things, supervise and assist him on writing code.

Since PART A did not seem very demanding we did not assign specific roles one to another, we did not discuss anything such as he was going to do the database and the frontend, and I would do the backend. However, if there were more requirements for this part, we would have been better off splitting the work accordingly, in different roles, not everyone working on everything. We mainly worked together on the day when the tutorial is scheduled and remained to carry on working after the tutorial has finished for few more hours.

# – PART B

## Screenshots

This are screenshots taken from the final version of the coursework.

### Implementation from Part A

#### Diagrams

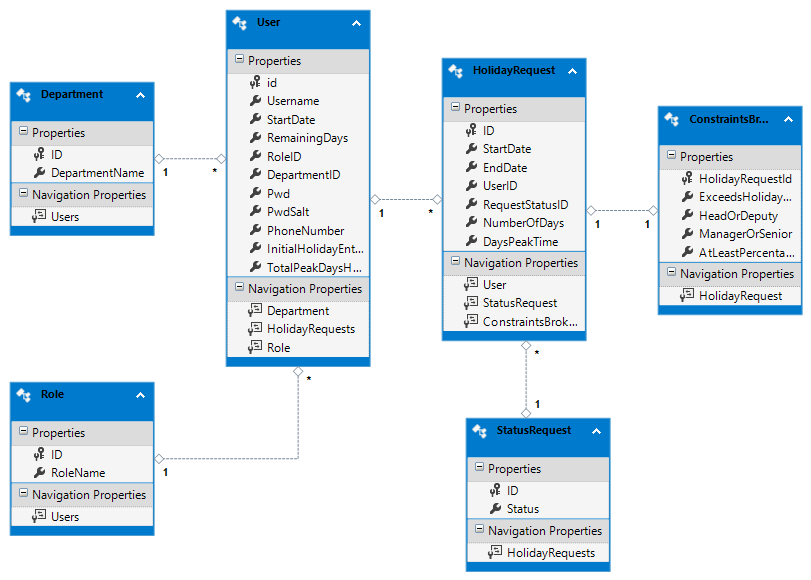


Figure 3 Entity Framework generated model of database

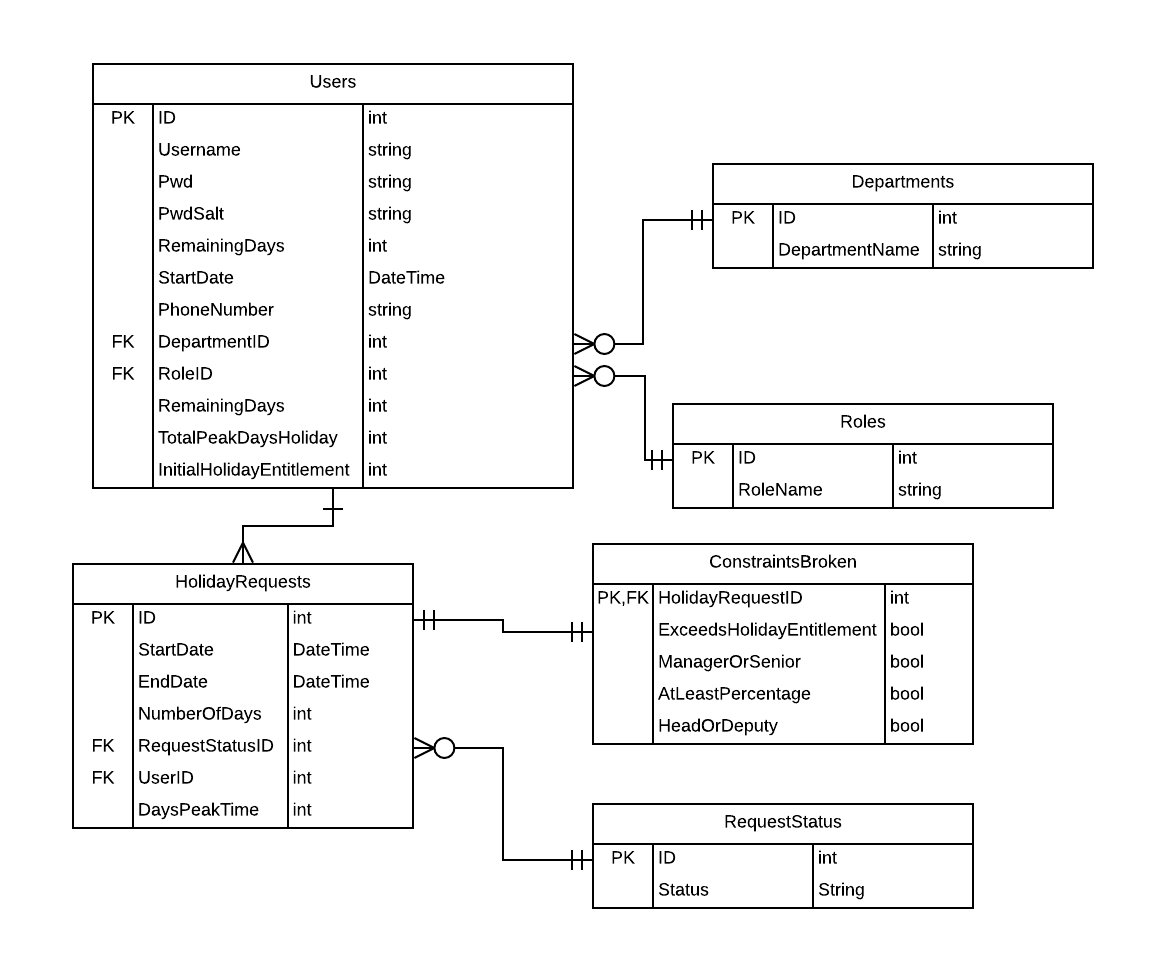


Figure 4 Final Entity Relationship Diagram

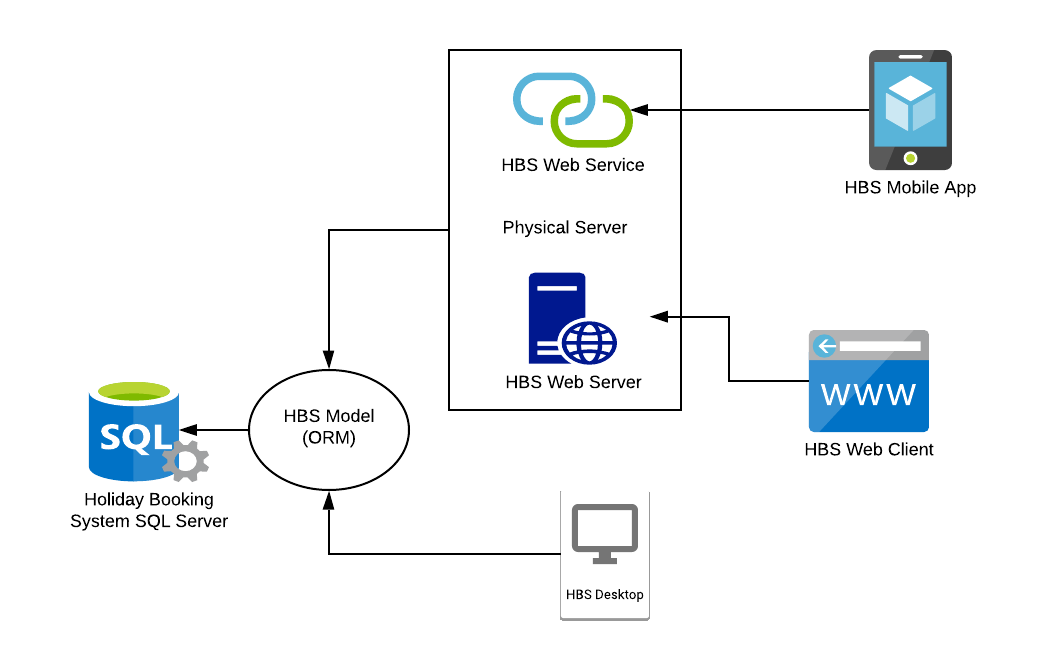


Figure 5 Final Architecture Diagram

#### Login

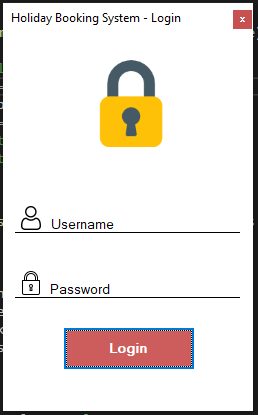


Figure 6 Desktop Login Form

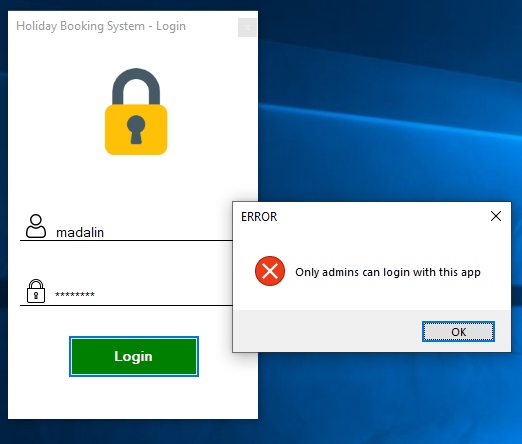


Figure 7 Attempting to sign in with employee account

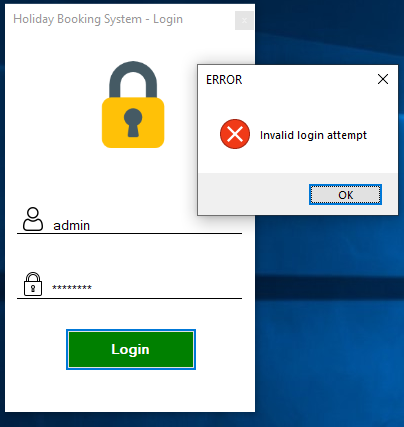


Figure 8 Typing wrong credentials on login

#### Add Employee

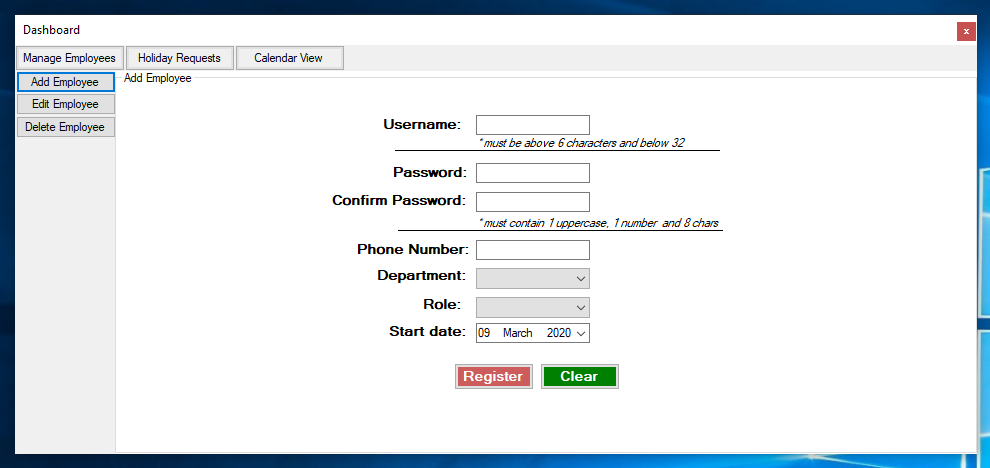


Figure 9 Desktop App Dashboard with Add Employee Form

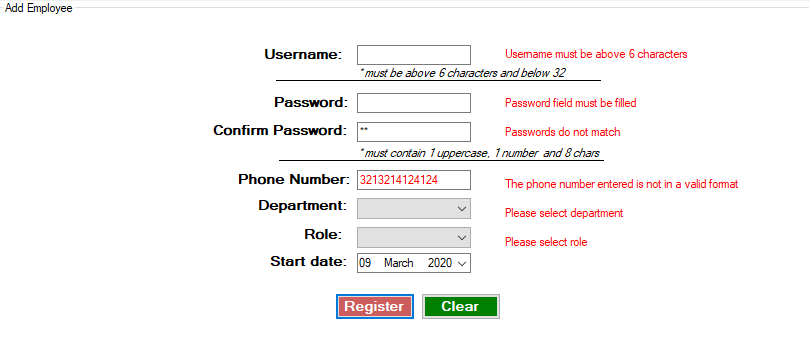


Figure 10 Create Employee Form Inputs Validation

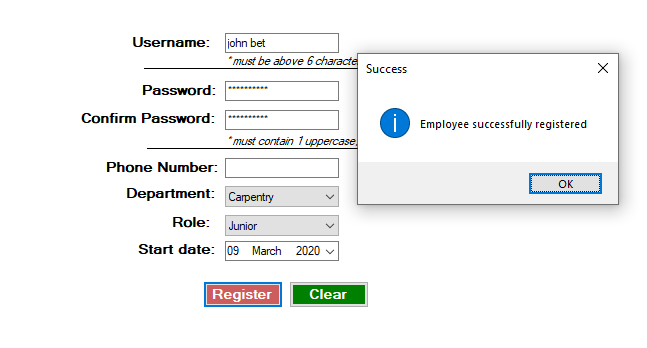


Figure 11 Successful employee registration

#### Edit Employee

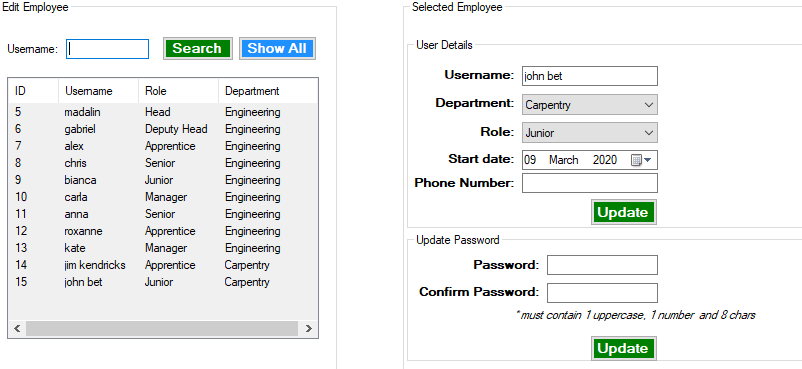


Figure 12 Edit employee details

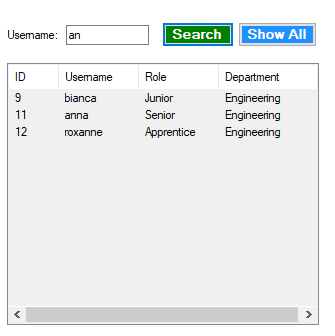


Figure 13 Search employee by Regular Expression match

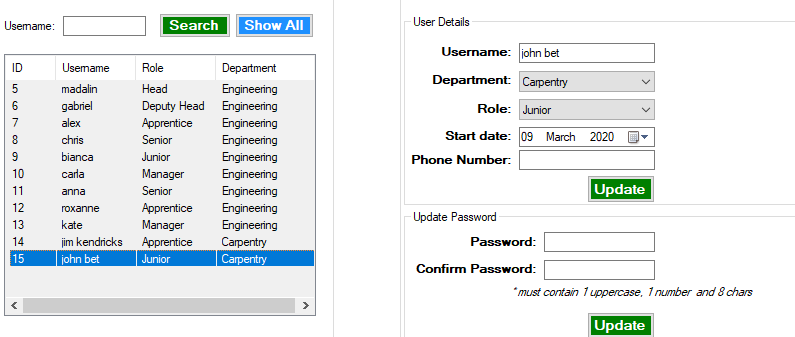


Figure 14 Editing selected employee details

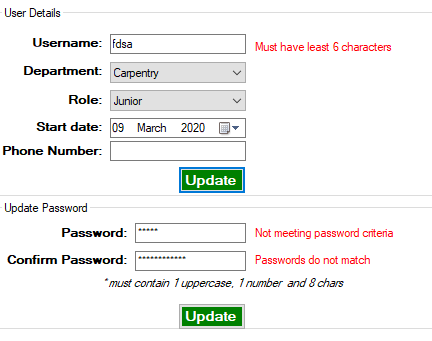


Figure 15 Input validation for editing employee details

#### Delete Employee

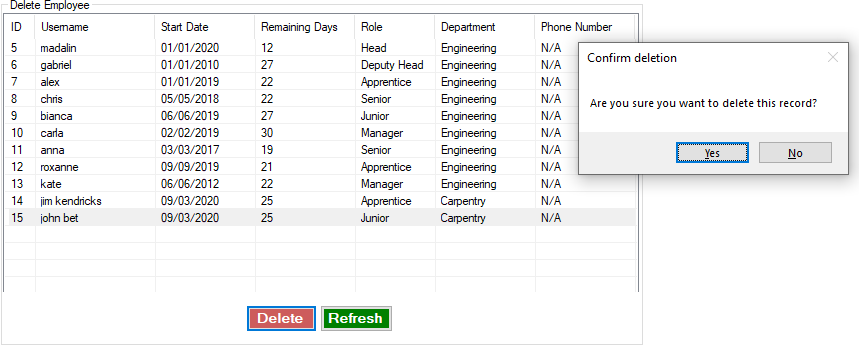


Figure 16 Delete Employee asking for confirmation

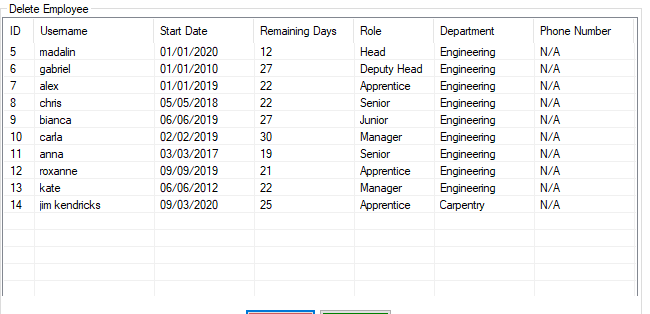


Figure 17 Once employee deleted the list refreshes

### Functionality A

#### Outstanding Requests

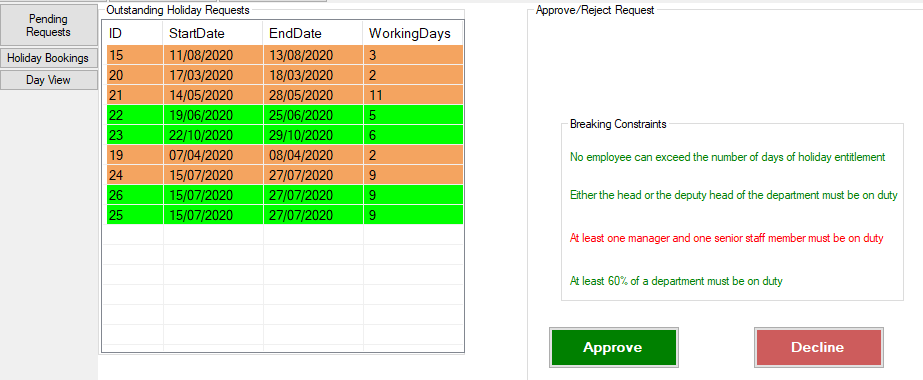


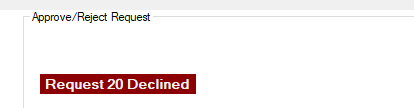
Figure 18 Outstanding Holiday Requests (includes the functionality with prioritization and constraints checking)

Figure 19 Declining a request

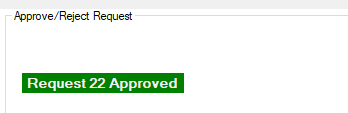


Figure 20 Approving a request

#### All holiday bookings and filter by employee

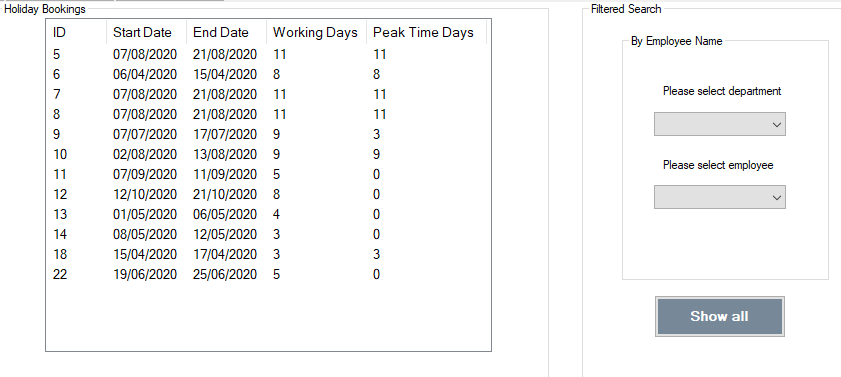


Figure 21 All holiday bookings

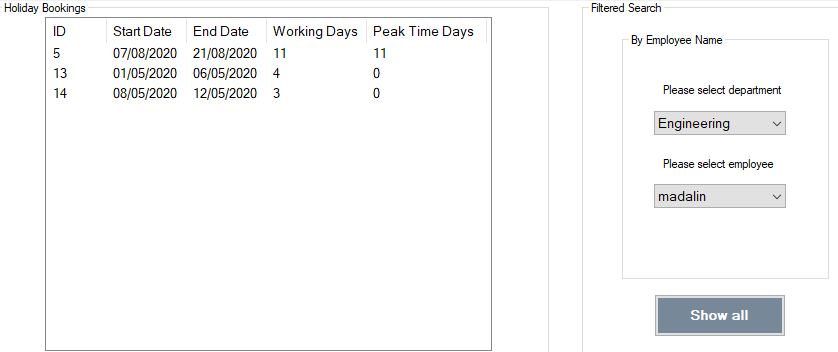


Figure 22 Holiday Requests filtered by employee

#### Employees working and on leave on a selected date

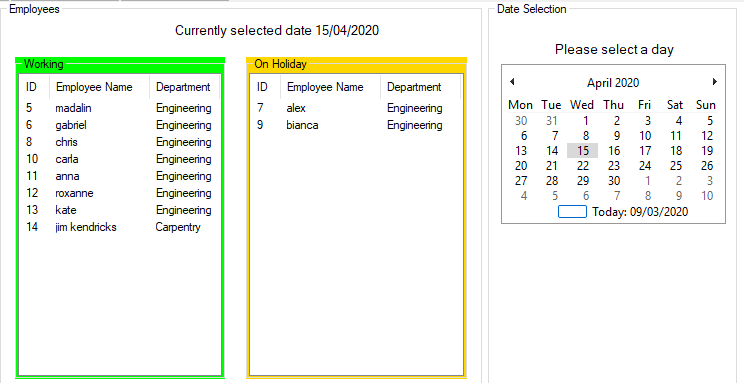


Figure 23 Employees working and employees on holiday

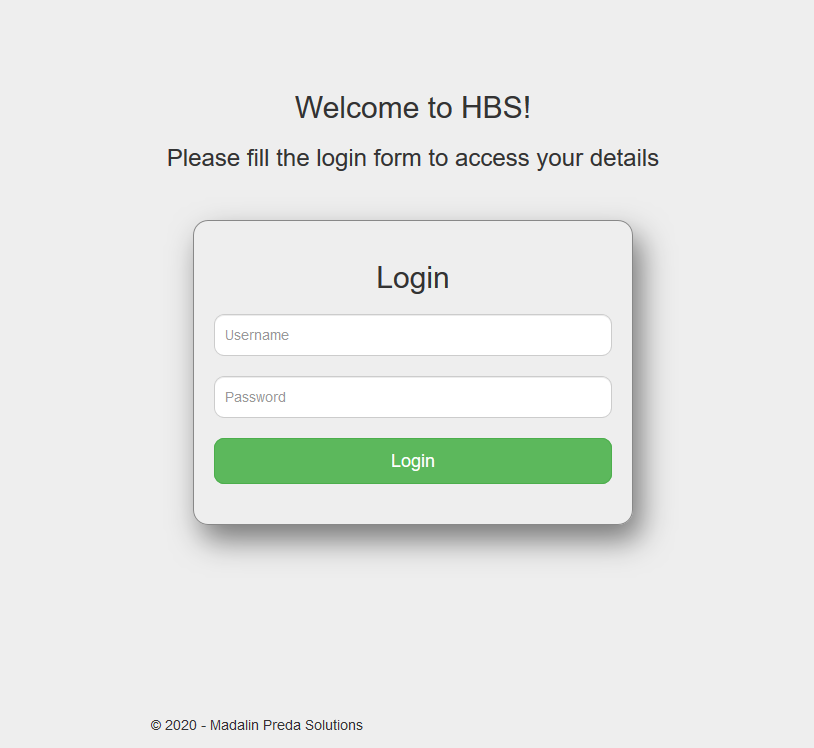
### Functionality B

#### Phone Text Box Component



Figure 24 Text becomes red if invalid input and only numbers allowed

#### ASP.NET Web forms Login



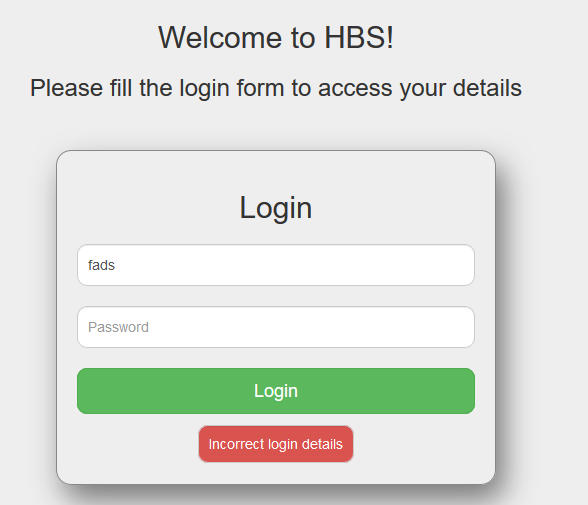


Figure 25 Failed login

#### Submit holiday request

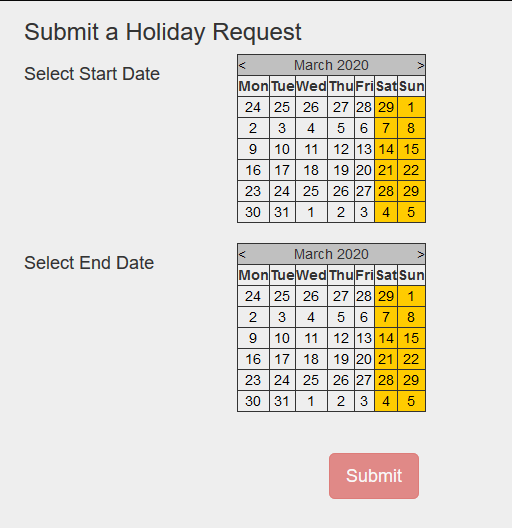


Figure 26 Submit Holiday Request

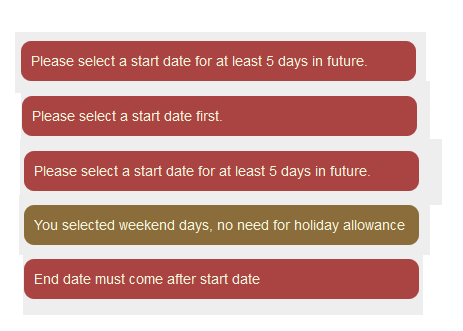


Figure 27 Types of error messages

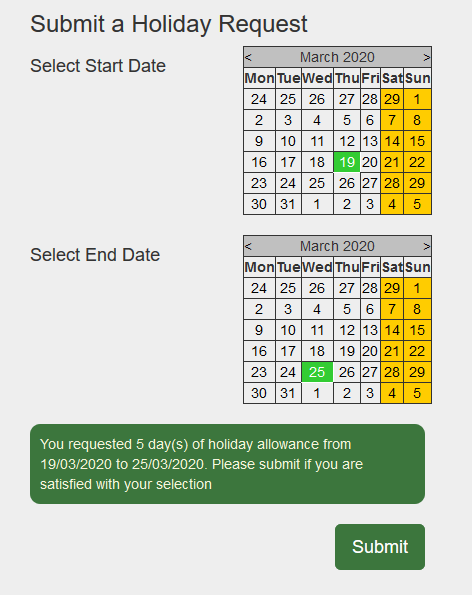


Figure 28 Valid holiday request submission

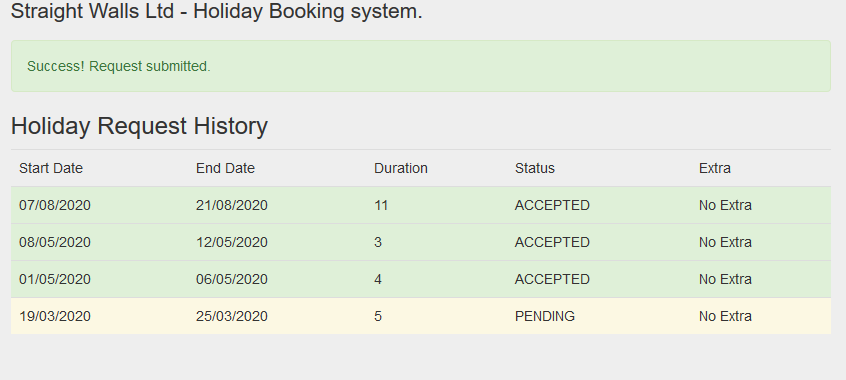


Figure 29 Confirmation request submitted

#### List of employee holiday requests and their status

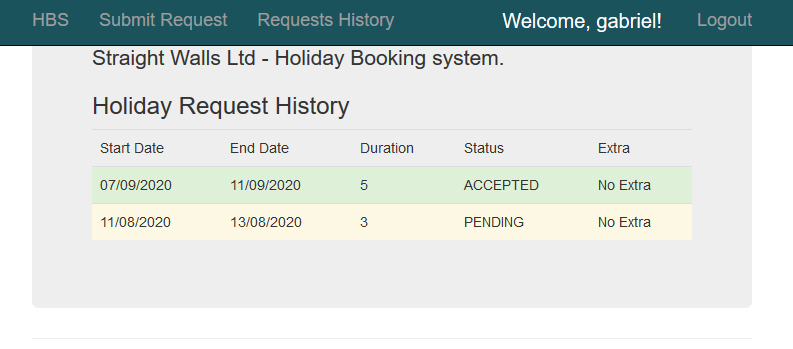


Figure 30 List of employee's holiday requests

### Functionality C

#### Constraint Checking Errors Display

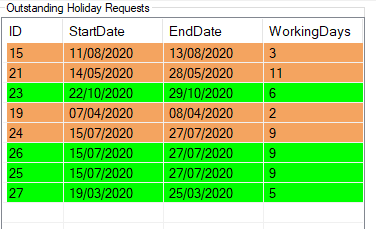


Figure 31 Requests breaking constraints (red), valid requests (green)

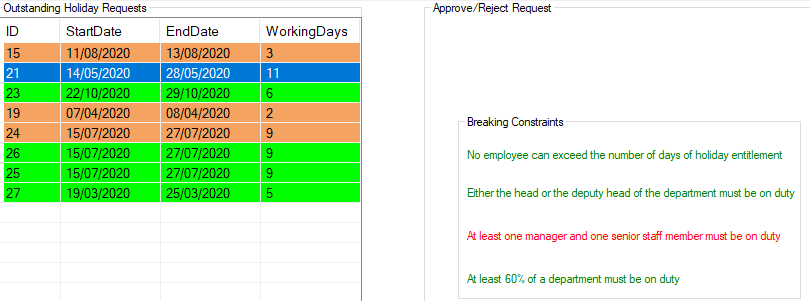


Figure 32 Constraints Broken Highlights

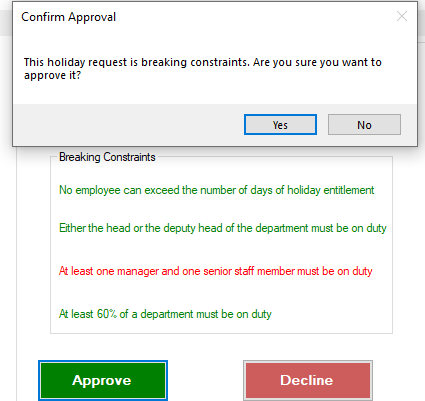


Figure 33 Approving holidays that break constraints

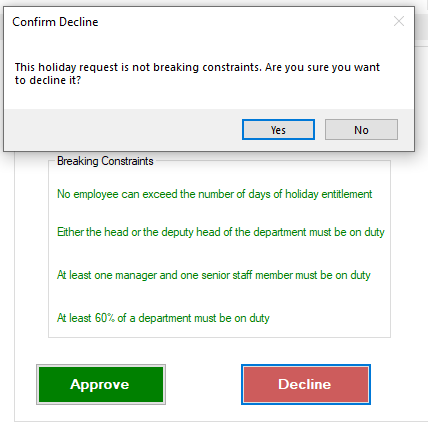


Figure 34 Declining holidays that do not break constraints

### Functionality D

#### Web Service

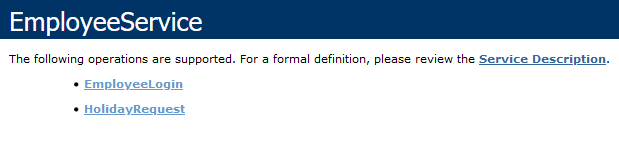


Figure 35 SOAP Web Service exposed methods

#### Mobile App Employee Login

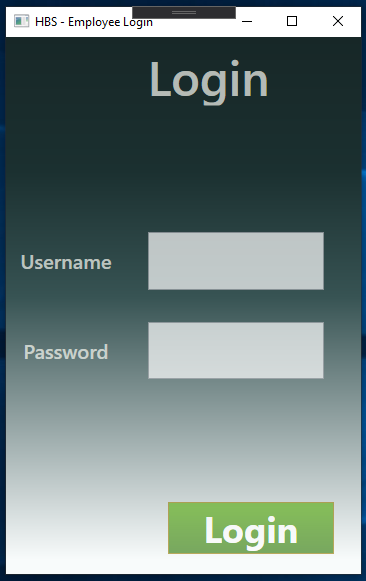


Figure 36 Login form employee app

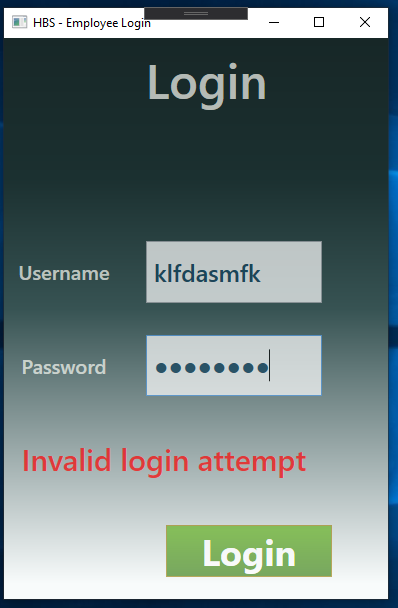


Figure 37 Attempt to login with wrong credentials

#### Employee app make holiday request

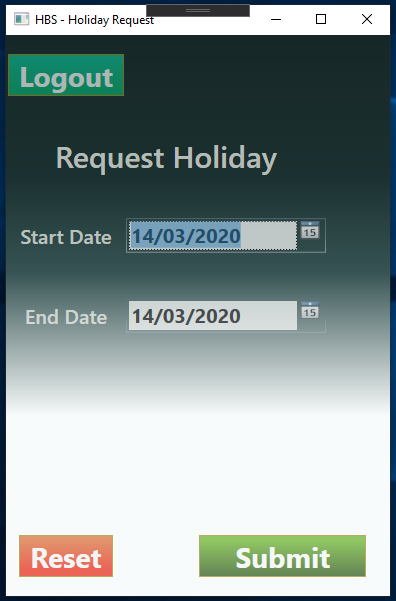


Figure 38 Submit holiday request

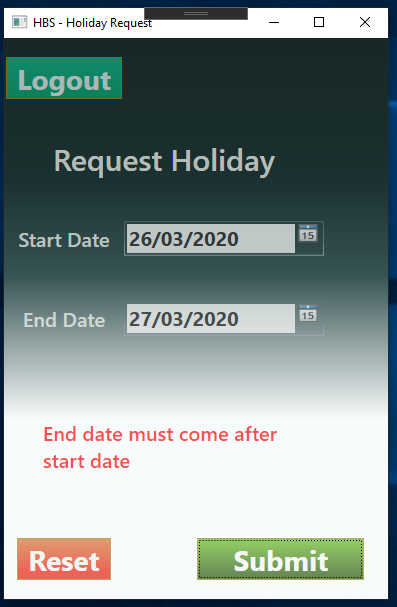


Figure 39 Making an invalid request

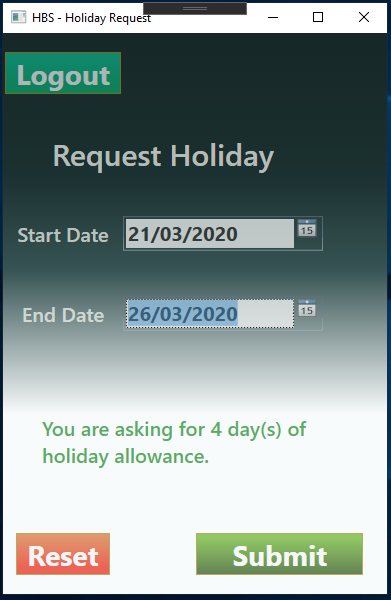


Figure 40 App checks how many days user asked for

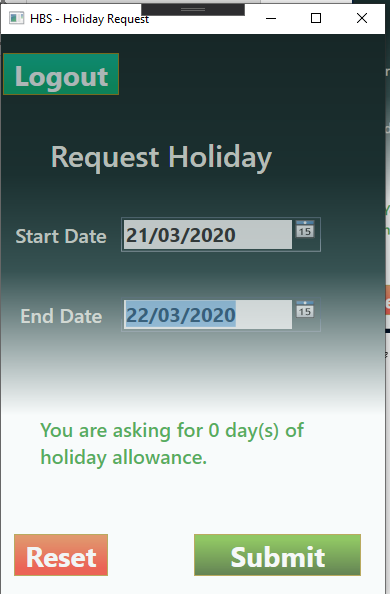


Figure 41 App lets user know if they select weekend days

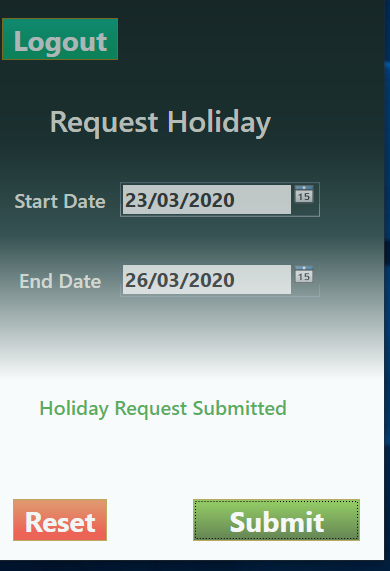


Figure 42 Feedback on request submission

### Functionality E

#### Prioritization of Holiday Requests

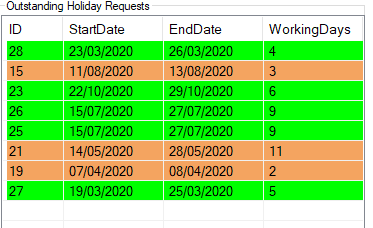


Figure 43 Rows are ordered by priority

#### Suggestions of Holiday

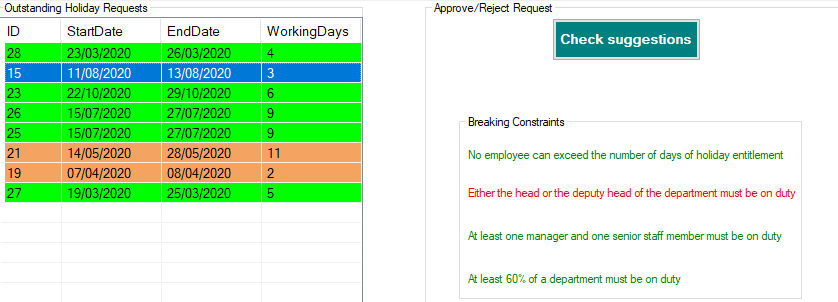


Figure 44 Suggestions button shows if request falls during peak times

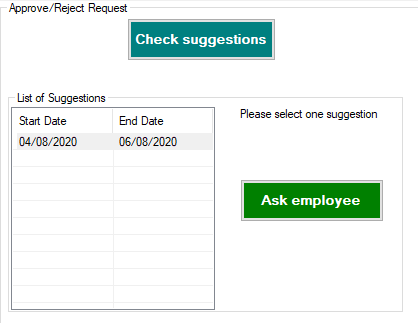


Figure 45 If suggestions available a list is returned

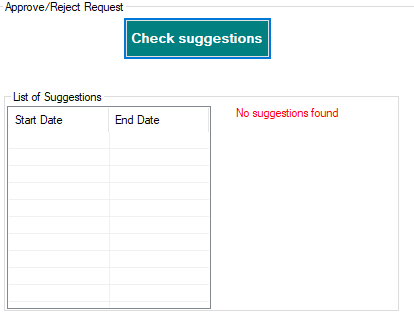


Figure 46 If no suggestions available