

**Operation Sundew**



November 14, 2019

Console Commander

https://www.consolecommander.net/

Contents

[1. Introduction 2](#_Toc25761607)

[1.1 Purpose 3](#_Toc25761608)

[1.2 Intended Audience 4](#_Toc25761609)

[1.3 Scope 5](#_Toc25761610)

[Benefits 5](#_Toc25761611)

[Objectives 5](#_Toc25761612)

[1.4 Definitions and Acronyms 6](#_Toc25761613)

[2. Overall Description 7](#_Toc25761614)

[2.1 User Needs 7](#_Toc25761615)

[2.2 Assumptions and Dependencies 8](#_Toc25761616)

[3. System Features and Requirements 9](#_Toc25761617)

[3.1 Functional Requirements 9](#_Toc25761618)

[3.2 Job Backend Operations summary 9](#_Toc25761619)

[3.2.1 Create User 9](#_Toc25761620)

[3.2.2 Update User 10](#_Toc25761621)

[3.2.3 Delete User 10](#_Toc25761622)

[3.2.4 Read User 11](#_Toc25761623)

[3.2.5 Read Users 11](#_Toc25761624)

[3.2.6 Search Users 12](#_Toc25761625)

[3.2.7 Verify Password 12](#_Toc25761626)

[3.2.6 Confirm Required Contact Info Provided 13](#_Toc25761627)

[3.2.7 Create Job 13](#_Toc25761628)

[3.2.8 Update Job 14](#_Toc25761629)

[3.2.9 Delete Job 14](#_Toc25761630)

[3.2.10 Delete All Job Problems 15](#_Toc25761631)

[3.2.11 Delete All Job Chats 15](#_Toc25761632)

[3.2.12 Read Job 16](#_Toc25761633)

[3.2.13 Read Jobs 16](#_Toc25761634)

[3.2.14 Search Jobs 16](#_Toc25761635)

[3.2.15 Auto Insert Job Start Date 17](#_Toc25761636)

[3.2.16 Confirm Required Job Info Provided 17](#_Toc25761637)

[3.2.17 Register User For Job 18](#_Toc25761638)

[3.2.18 Create Problem 18](#_Toc25761639)

[3.2.19 Update Problem 18](#_Toc25761640)

[3.2.20 Delete Problem 19](#_Toc25761641)

[3.2.21 Read Problem 19](#_Toc25761642)

[3.2.22 Read Problems 19](#_Toc25761643)

[3.2.23 Search Problems 19](#_Toc25761644)

[3.2.24 Confirm Required Info Is Not Empty 20](#_Toc25761645)

[3.2.25 Convert Problem Severity To Text 20](#_Toc25761646)

[3.2.26 Attach Zip Folder 20](#_Toc25761647)

[3.2.27 Attach Zip File 20](#_Toc25761648)

[3.2.28 Create Chat 20](#_Toc25761649)

[3.2.29 Publish Chat Time And Date 21](#_Toc25761650)

[3.2.30 Read Chat 21](#_Toc25761651)

[3.2.31 Update Chat 21](#_Toc25761652)

[3.2.32 Delete Chat 21](#_Toc25761653)

[3.2.33 Confirm Required Info Is Not Empty 21](#_Toc25761654)

[3.3 External Interface Requirements 22](#_Toc25761655)

[3.3 System Features 24](#_Toc25761656)

[3.4 Nonfunctional Requirements 24](#_Toc25761657)

[3.5 Entity Relationship Diagram 24](#_Toc25761658)

[3.6 UML Class Diagram 25](#_Toc25761659)

[3.7 Security 26](#_Toc25761660)

# 1. Introduction

We have a problem in which we are constantly using Microsoft Word to show what bugs we have and how we are attempting to solve them. We need something like Jira so that we can log bugs. We don’t have the budget for that.

My plan is to create Sundew a bug tracking software which can contain projects, the people assigned to the project and bugs for each project. So if I work with a team, we can use it to report bugs and solve them together as a group.

This will stop us from having to report bugs verbally and interrupting each other’s work or sending emails back and forth that contain duplicate descriptions about the bug.

As an optional requirement we can allow this system to email other parties upon logging of the bug including the sender as well. We will use an Agile based methodology that allows us to plan and complete our work in sprints. Not just anyone can be able to use this so we will need a login system.

## Purpose

To make life easier for developers to stop from emailing duplicate bug reports to each other. To have a system that show bugs that are resolved and not resolved and have an automated process of emailing developers the bug status until app can be pushed into production or made useful for software development teams.

Purpose is also to make this application reusable by having backend database as reusable as possible for other CRUD business scenarios.

This software will operate on a server giving teams 24hr access.

## Intended Audience

Just me and any developers who collaborate in this project.

## Scope

We are developing bug tracking web software.

The software will have the following types of information

### Benefits

* Will use open source software that can run on any hosting environment.
* Team leader will be the only one who can create read, update and delete projects in the system. This will stop the team from creating duplicate projects.
* Increased productivity and less interruption.
* Centralized system that teams can tap into especially while remote.
* Allow team to track reported bugs, and comment on each one and be notified via email and application notifications of future progress including the status and severity of the bug.
* Allows team members to prioritize bugs in severity.

Objectives

Objectives are in this <Agile-Product-Backlog.xlsx>

|  |  |
| --- | --- |
| **Objectives** | **Complete?** |
| To allow team leader to create projects and users only. |  |
| Used a third party authentication and authorization service to allow users to login. |  |
| To have title about the problem that tells a story in several different information fields, including the date/time/, ID, description, steps to replicate, who’s assigned to it. |  |
| To allow team to assign members to bugs including themselves. |  |
| Can login and out of system. |  |
| Can display information about bug. |  |
| Can write information about bug to database. |  |
| Can have backend information as generic as possible to be reused for other applications. |  |

[Risk table here](Risk%20Matrix.docx)

## Definitions and Acronyms

**Sundew:** The project which is named after a carnivorous plant.

**ASP.NET Core**: Active Server Pages. Open source web language that helps build dynamic web sites, applications and services

**Technical Debt**: The cost of additional rework, caused by choosing an easy (limited) solution now instead of using a better approach that would take longer due to complexity.

**Developer**: A person who create a program.

**Automation Tester**: Someone who automates tests.

**Quality Assurance**: Someone who ensures the application is performing to client specifications.

**Product Owner**: Someone who drives product from business perspective. There are a customer intermediately.

**Scrum Master**: The Project Manager.

**Bootstrap**: CSS framework directed at responsive, mobile-first front-end web development.

**Entity Framework Core**: Open source ORM that allows .NET developers to eliminate the need for most of the data-access code.

**Auth 0**: A universal authentication & authorization platform for web, mobile and legacy applications.

**SQL Server:** A relational database management system.

**ORM:** Relational Object Mapper.

**MVC:** Model View Controller.

**Design Pattern:** A highly reusable framework solution to a commonly occurring problem.

**DGV:** Data Grid View. A digital interactive grid that contains data. Similar to a spreadsheet.

**CBO**: Combo Box.

# 2. Overall Description

This will be a whole new product. We are going to attempt to back the backend as generic as possible in it’s naming structure so it will become more reusable. For example instead of calling a table tblBug we will call it tblProblem as every customer usually wants software to solve a business problem and not a bug in their system. We will also attempt to do the same with the front end, whenever possible.

## 2.1 User Needs

We have 2 actors. Admin and user. The admin can be a tech leader and the user can be a developer under that tech lead.

Admin will be able to create users and mark whether user is also a leader or not. If they are then they will get access to the user index.

User and Admin can both login to the Main menu and Admin can access users while User can only read their own account information.

They will both receive problems and job they have all been assigned to and can be notified by email and or text about them been created, updated, or deleted. They input additional info and update the problem creator in the chat log. The person who opened the bug is the one who can close it.

Chats

Job Problems

My

Account

User

Admin

User Index

Main Menu

Bug tracking system

User

Account

Problems

Jobs

## 2.2 Assumptions and Dependencies

* I have never done this before.
* I assume anyone can create an issue that is a part of the project. I don’t feel I have to make separate fields to indicate what type of employee the user is.
* I don’t feel the project needs to have a set of permissions for different types of employees. The only real permission should be where the user who created the problem should be the same one who can close it and approve the status of whether the problem was fixed.
* I don’t feel we need to specify different types of users such as testers and developers. The people using this system are more concerned about the what, when, where, why and most importantly how. How do we solve the problem?

# 3. System Features and Requirements

## 3.1 Functional Requirements

The software will have login for each user. Each user will automatically have a registration date. There should be an option to indicate if the user is a leader of that job. If user is a leader then only they will be able to generate projects. This will stop users from creating duplicate projects. Each project will allow user to create, read update and delete many bugs.

### 3.2 Job Backend Operations summary

**Users**: CreateUser, UpdateUser, DeleteUser, ReadUser, ReadUsers, SearchUsers, VerifyPassword, ConfirmRequiredContactInfoProvided.

**Jobs**: CreateJob, UpdateJob, DeleteJob + DeleteAllJobProblems + DeleteAllJobChats, ReadJob, ReadJobs, SearchJobs, AutoInsertJobStartDate, ConfirmRequiredJobInfoProvided.

**Registrations**: RegisterUserForJob.

**Problems**: CreateProblem, UpdateProblem, DeleteJob + DeleteAllJobProblems + DeleteAllJobChats, ReadProblem, ReadProblems, SearchProblems, ConfirmRequiredInfoIsNotEmpty, ConvertProblemSeverityToText, AttachZipFolder, AttachZipFile.

**Chats**: CreateChat + PublishChatTimeAndDate, ReadChat, UpdateChat, DeleteChat, ConfirmRequiredInfoRIsNotEmpty,

### 3.2.1 Create User

|  |  |
| --- | --- |
| **Operation Name** | CreateUser |
| **Trigger** | The user who has the boolean value of UserIsLeader set to true, will get to access the User Index and create the user. They will click a “Create user” button to do that. |
| **Precondition** | The Admin has accessed the User Index page linked from the Main Menu and has selected the “New user account” and being taken to the User Account page. |
| **Basic Path** | 1. Admin selects a create button. 2. Execute method Confirm\_required\_contact\_info\_provided   Else user admin will be told which fields are empty and what to save. |
| **Post condition** | Record in list will be persisted to database and be assigned a Primary key. User will be notified via automated message saying.  “John Doe has added you to your Sundew bug tracking network.  Your username is: JaneDoe117  Your Password is: {#48jmKaPo982\*#!  ” |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### 3.2.2 Update User

|  |  |
| --- | --- |
| **Operation Name** | UpdateUser |
| **Trigger** | The user who has the boolean value of UserIsLeader set to true, will get to access the User Index and update the user. They will select from a Data Grid View that contains the users, click “Edit” and get all the fields appearing of the user they will “update.” By clicking update button. |
| **Precondition** | The Admin is on the User Account page linked from the User Index page linked from the main menu. |
| **Basic Path** | 1. If the admin has edited any of the rendered user data 2. Update button will be enabled. 3. Click Update. 4. Execute method Confirm\_required\_contact\_info\_provided |
| **Postcondition** | If in step 4 the required data was provided than any database cell will be replaced with the new data in a new data list.  User will be notified via automated message saying for example.  “John Doe has updated your information in your Sundew bug tracking network.  Your username is now: JaneDoe117  And your mobile number is now: 0218 287 143” |
| **Exception Paths** | The attempt may be abandoned at any time. Also if the user didn’t fill out the required fields they will be denied update. |
| **Alternative** |  |

3.2.3 Delete User

|  |  |
| --- | --- |
| **Operation Name** | DeleteUser |
| **Trigger** | The user who has the boolean value of UserIsLeader set to true, will get to access the User Index and delete the user. |
| **Precondition** | The Admin is on the User Index Page. |
| **Basic Path** | 1. Admin will select from a Data Grid View that contains the user they want to delete. 2. They will get a box asking them if they are sure. If yes, delete. Else, don’t. 3. Selected record to list will be deleted.   User will be notified via automated message saying for example.  “John Doe has deleted your account from the Sundew bug tracking network. |
| **Postcondition** | The empty list will be deleted where ID is equal to selected record. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### 3.2.4 Read User

|  |  |
| --- | --- |
| **Operation Name** | ReadUser |
| **Trigger** | The user who has the boolean value of UserIsLeader set to true, will get to access the User Index and see the user where ID is equal to the one they selected in a Data Grid View that contains the users.  They will click “Edit” and get all the fields appearing of the user they will “update.” By clicking update button. |
| **Precondition** | The Admin is on the User Index page linked from the Main Menu page. |
| **Basic Path** | 1. Admin selects the user from DataGridView. 2. Admin clicks “Edit”. 3. Selected user is put in a list and transferred to the User Account Page. |
| **Postcondition** | Where user ID list is equal to one in the database, the record is rendered to the User Account page fields. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** | The standard user who is logged in can access their account and only read their details that the Admin user has entered for them in the My Account page. |

3.2.5 Read Users

|  |  |
| --- | --- |
| **Operation Name** | ReadUsers |
| **Trigger** | The user who has the boolean value of UserIsLeader set to true, will get to access the User Index and see all users in a Data Grid View. |
| **Precondition** | The Admin is on the User Index page linked from the Main Menu. |
| **Basic Path** | If the admin user is in the User Index.   1. The web GUI will auto send response to database to see all the users from Users table. 2. Database will return all records. |
| **Postcondition** | Data Grid View will be populated with a bound list of Users from database and dispose resources. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### Search Users

|  |  |
| --- | --- |
| **Operation Name** | SearchUsers |
| **Trigger** | The user who has the boolean value of UserIsLeader set to true, will get to access the User Index. |
| **Precondition** | The Admin is on the User Index page linked from the Main Menu. |
| **Basic Path** | If the admin user is in the User Index.   1. The web GUI will auto send response to database to see all the users from Users table. 2. Database will return all records. |
| **Postcondition** | Data Grid View will be populated with a bound list of Users from database and dispose resources. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### Verify Password

|  |  |
| --- | --- |
| **Operation Name** | VerifyPassword |
| **Trigger** | The user who has the boolean value of UserIsLeader set to true, will get to access the User Index and will have the password verified by a third party service. |
| **Precondition** | The Admin is on the User Account page linked from the User Index. |
| **Basic Path** | 1. Admin user enters password twice and username. 2. Third Party service verifies the password twice. 3. Checks passwords match. |
| **Postcondition** | If passwords match then created or updated user will have a successful |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### Confirm Required Contact Info Provided

|  |  |
| --- | --- |
| **Operation Name** | Confirm\_required\_contact\_info\_provided |
| **Trigger** | Admin is creating or updating user. Admin selects existing user to update user from DGV or creates new one. |
| **Precondition** | The Admin is on the User Account page linked from the User Index. |
| **Basic Path** | 1. If the user email contains “@” AND user admin has filled out the required fields:   • First Name  • Last Name  • Login Name  • Login Password   1. Allow admin to perform desired CRUD operation, with these details. |
| **Postcondition** | Step 2 is executed if step 1 is true. |
| **Exception Paths** | The attempt may be abandoned at any time. In addition if step 1 is not true then we execute a statement saying “user details invalid” and identify the invalid fields. |
| **Alternative** |  |

### Create Job

|  |  |
| --- | --- |
| **Operation Name** | Create\_Job |
| **Trigger** | Admin has clicked the “Create New Job” button. |
| **Precondition** | Admin has accessed the Jobs page from the Main Menu page. |
| **Basic Path** | 1. Admin presented with a blank form to enter new Job info and a DGV containing all other jobs. 2. Admin has filled out required fields. 3. Admin has selected users from DGV to be assigned to job. 4. Admin hits “Create New Job” 5. System executes Confirm\_required\_job\_info\_provided. |
| **Postcondition** | Job is created in database. All users who are registered to it are informed about it:  “John Doe has given you access to your new job Brisbane Hospital Medication Database” |
| **Exception Paths** | The attempt may be abandoned at any time. In addition if step 5 has invalid information, we execute a statement saying “no job title” and identify the invalid field. |
| **Alternative** |  |

### Update Job

|  |  |
| --- | --- |
| **Operation Name** | Update\_Job |
| **Trigger** | Admin has selected an existing job from the DGV and the data in one of the web form fields has changed. |
| **Precondition** | The Admin has accessed the Jobs page from the main menu page. |
| **Basic Path** | 1. Admin select job record from DGV. 2. Job record populate web form fields on same page. 3. Admin edits selected fields. 4. Update button enables. 5. Admin clicks “Update job”. 6. Auto triggers Confirm\_required\_job\_info\_provided. 7. Binds record to list. |
| **Postcondition** | Replaces record in database with new one. Refreshes the DGV to include updated record.  All users who are registered to it are informed about it via email:  “John Doe has updated job Brisbane Hospital Medication Database” |
| **Exception Paths** | If the job is not already in the database, the operation is abandoned. In addition, the Admin may abandon the operation at any time. |
| **Alternative** |  |

### Delete Job

|  |  |
| --- | --- |
| **Operation Name** | Delete\_Job |
| **Trigger** | Admin has selected an existing job from the DGV and the data has populated the ones in the web form fields. |
| **Precondition** | The Admin has accessed the Jobs page from the main menu page. |
| **Basic Path** | 1. Admin select job record from DGV. 2. Job record populate web form fields on same page.   3. Delete current job button appears on form.  4. Admin clicks button “Delete current job” |
| **Postcondition** | IF user clicks button, message box appears asking user if they are sure. If they click yes then record is deleted from database. Record in fields clears. DGV refreshes to process the changes made.  All users who are registered to it are informed about it via email:  “John Doe has deleted job Brisbane Hospital Medication Database” |
| **Exception Paths** | If the job is not already in the database, the operation is abandoned. In addition, the Admin may abandon the operation at any time. |
| **Alternative** |  |

### Delete All Job Problems

|  |  |
| --- | --- |
| **Operation Name** | Delete\_All\_Job\_Problems |
| **Trigger** | Admin has deleted a job from the Jobs page and the problems from it are erased. |
| **Precondition** | Admin must be on the Jobs page and have selected a job from DGV. |
| **Basic Path** | 1. Admin selects Job record from the DGV. 2. Admin deletes current job. |
| **Postcondition** | Auto delete all problems associated with the job so they don’t become orphaned records. |
| **Exception Paths** | If the job is not already in the database, the operation is abandoned. In addition, the Admin may abandon the operation at any time. If there are no problems then the operation is abandoned. |
| **Alternative** |  |

### Delete All Job Chats

|  |  |
| --- | --- |
| **Operation Name** | Delete\_All\_Job\_Chats |
| **Trigger** | Admin has deleted a job from the Jobs page and the chat from it are erased. |
| **Precondition** | Admin must be on the Jobs page and have selected a job from DGV. |
| **Basic Path** | 1. Admin selects Job record from the DGV. 2. Admin deletes current record. 3. Foreach problem in job.   Go to post condition. |
| **Postcondition** | Delete all Chats in the problem. |
| **Exception Paths** | If the job is not already in the database, the operation is abandoned. In addition, the Admin may abandon the operation at any time. If there are no chats then the operation is abandoned. |
| **Alternative** |  |

### Read Job

|  |  |
| --- | --- |
| **Operation Name** | ReadJob |
| **Trigger** | Admin or user has selected a record from DGV that matches their registration id and has populated the web form fields with information from that selected record. |
| **Precondition** | User must be on the Jobs page to see this information |
| **Basic Path** | 1. User selects record from DGV. 2. Request goes to database to get a record by JobID that the user has selected. |
| **Postcondition** | Record data populates web control fields. |
| **Exception Paths** | The user may abandon the operation at any time. |
| **Alternative** |  |

### Read Jobs

|  |  |
| --- | --- |
| **Operation Name** | ReadJobs |
| **Trigger** | Admin/user auto request jobs that they are registered to when page loads. |
| **Precondition** | User navigates to Job page. |
| **Basic Path** | 1. User loads Job page 2. If user is registered to current job. 3. Load records that user is registered to into DGV. |
| **Postcondition** | Record data populates DGV. |
| **Exception Paths** | The user may abandon the operation at any time. |
| **Alternative** |  |

### Search Jobs

|  |  |
| --- | --- |
| **Operation Name** | SearchJobs |
| **Trigger** | The user who is registered to the jobs will be able to see the jobs that match the keyword they have. |
| **Precondition** | The Admin is on the Job page linked from the Main Menu. |
| **Basic Path** | 1. User types text into the search box that matches text and hits the search button. 2. Refresh DGV. 3. Foreach record that the user is registered to. 4. Search job record for matching text. 5. If record text matches text in textbox. 6. Put record in list. |
| **Postcondition** | Show record that matches keywords in DGV. Clear up system resources. |
| **Exception Paths** | The attempt may be abandoned at any time. If no records match the searched text, tell user “Sorry no records match.” |
| **Alternative** |  |

### Auto Insert Job Start Date

|  |  |
| --- | --- |
| **Operation Name** | AutoInsertJobStartDate |
| **Trigger** | Admin has clicked the “Create New Job” button and has been updated with the current date and time it was created. |
| **Precondition** | Admin has accessed the Jobs page from the main menu page. |
| **Basic Path** | 1. Admin has filled job title. 2. IF Start date is empty. 3. Get current time and date from database. 4. Format data as “yyyy/mm/dd hh:mm:ss”. |
| **Postcondition** | Insert date and time into database field and clean up resources. |
| **Exception Paths** | The attempt may be abandoned at any time.  If no time is coming out of database data insertion will be abandoned. In addition tell user. “There was a problem auto inserting the job creation time and date. Please contact your administrator” |
| **Alternative** |  |

### Confirm Required Job Info Provided

|  |  |
| --- | --- |
| **Operation Name** | Confirm\_required\_job\_info\_provided |
| **Trigger** | The Admin has clicked the “Update Job” button or the “Create New Job” button. |
| **Precondition** | The Admin is presented with a blank web form containing the Job details for them to fill and a DGV with all the other job on it which will bind the selected DGV row to the job web form fields and populate them. |
| **Basic Path** | 1. Admin has hit “Update Job” or has hit “Create new Job” 2. If Admin has filled out the required field:   • Job Title   1. Allow admin to perform desired CRUD operation, with these details. 2. Else tell Admin to enter a title name. |
| **Postcondition** | If successful admin has performed CRUD op, else message them why op is not successful. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### Register User For Job

|  |  |
| --- | --- |
| **Operation Name** | RegisterUserForJob |
| **Trigger** | Job has been created by admin and has assigned an existing user or multiple users to it. |
| **Precondition** | Admin is in the Assign Users web form page. The webpage will have 2 list boxes on it, one for users who are no assigned to project and users who are.  Clicking the “>” which means (assign user to job). It will do just that.  Clicking the “<” which means (remove user from job). It will do just that.  It is accessible from the Jobs webpage via the “Assign users to job button” and only possible to navigate to once the Admin has selected a record from the DGV. |
| **Basic Path** | 1. IF Admin select user from listbox of non-assigned users. 2. Assign selected user from listbox to listbox of assigned job users. 3. Click “Register Users”. 4. For each user in listbox of assigned users 5. Assign JobID to user. 6. If Start date is empty. 7. Get current time and date from database. 8. Format data as “yyyy/mm/dd hh:mm:ss”. |
| **Postcondition** | Save Registration record to database. |
| **Exception Paths** | The attempt may be abandoned at any time. If User have been deleted in system, output “user no longer exists”. |
| **Alternative** |  |

### Create Problem

|  |  |
| --- | --- |
| **Operation Name** | CreateProblem |
| **Trigger** | User/Admin has filled all the required information fields to generate a problem and has been assigned a job id for the Job Problem to relate to. |
| **Precondition** | User has clicked the “Create problem” button in the Jobs page and is now in the Job Problem page which is linked from the Jobs page. |
| **Basic Path** | 1. User enters the following required fields.  * ProblemTitle * ProblemDescription * ProblemSeverity  1. Auto assign Job ID from previous Page to Problem. 2. Register User to problem with Registration ID. |
| **Postcondition** | Insert record into database. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### Update Problem

|  |  |
| --- | --- |
| **Operation Name** | UpdateProblem |
| **Trigger** | User/Admin will select a problem from a Data Grid View that contains the problems for the project and click “Edit” and get all the fields appearing of the problem they will update want to update by editing them and clicking update button. |
| **Precondition** | User is on the Job Problem form accessible by selecting DGV record from the Problems list and clicking the “Edit” button. |
| **Basic Path** | 1. Record data populate web form fields on same page. 2. User edits selected fields. 3. Update button enables. 4. Admin clicks “Update job”. 5. Auto triggers Confirm\_required\_job\_info\_provided. 6. Binds record to list. |
| **Postcondition** | Updated record is inserted into database. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### Delete Problem

|  |  |
| --- | --- |
| **Operation Name** | DeleteProblem |
| **Trigger** | User selects record from DGV, then clicks the Delete Record button. |
| **Precondition** | User has accessed the Problems page. |
| **Basic Path** | 1. Admin select job record from DGV. 2. Delete Record button enables. 3. User clicks the button “Delete current job”. |
| **Postcondition** | Selected record is deleted from database. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### Read Problem

|  |  |
| --- | --- |
| **Operation Name** | ReadProblem |
| **Trigger** | Admin or user has selected a record from DGV that matches their registration id and has populated the web form fields with information from that selected record. |
| **Precondition** | User must be on the Job Problem page which is accessible via the selected DGV problem record and clicking “Edit” via the Problems menu. |
| **Basic Path** | 1. User request loads Job Problem page 2. Request goes to database to get a record by ProblemID that the user has selected. |
| **Postcondition** | Load Problem data to all web form fields. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### Read Problems

|  |  |
| --- | --- |
| **Operation Name** | ReadProblems |
| **Trigger** | The user who is registered to a job or many jobs will see all problems which are to do with those jobs that match the keyword they have. |
| **Precondition** | The Admin is on the Job page linked from the Main Menu. |
| **Basic Path** | 1. Foreach problem record that the user is registered to. 2. Get record where the current user has a matching JobID. |
| **Postcondition** | Show all problem records that matches the JobIDs that the current user is registered to. Clear up system resources. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### Search Problems

|  |  |
| --- | --- |
| **Operation Name** | SearchProblems |
| **Trigger** | The user who is registered to the job’s problems will be able to see the problems that match the keyword they have. |
| **Precondition** | The user or admin is on the Problems page linked from the Main Menu. |
| **Basic Path** | 1. User types text into the search box that matches text in the list problems and hits the search button. 2. Refresh DGV. 3. Foreach job that the user is registered to. 4. (Nested) Foreach problem that matches that JobID. 5. Search problem record for matching text. 6. If record text matches text in textbox. 7. Put record in list. |
| **Postcondition** | Show record that matches keywords in DGV. Clear up system resources. |
| **Exception Paths** | The attempt may be abandoned at any time. If no records match the searched text, tell user “Sorry no records match.” |
| **Alternative** |  |

### Confirm Required Problem Info Provided

|  |  |
| --- | --- |
| **Operation Name** | Confirm\_required\_problem\_info\_provided |
| **Trigger** | The user has clicked the “Update problem” button or the “Create New problem” button. |
| **Precondition** | The Admin is presented with a blank web form containing the problem details for them to fill and a DGV with all the other problem on it which will bind the selected DGV row to the job web form fields and populate them. |
| **Basic Path** | 1. Admin has hit “Update Job” or has hit “Create new Job” 2. If Admin has filled out the required fields (as seen in ERD) 3. Allow admin to perform desired CRUD operation, with these details. 4. Else tell Admin to fill required fields. |
| **Postcondition** | If successful admin has performed CRUD op, else message them why op is not successful. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### Convert Problem Severity to Text

|  |  |
| --- | --- |
| **Operation Name** | Convert\_problem\_severity\_to\_text |
| **Trigger** | User/Admin is inputting or reading the severity level of the problem by selecting one of the 5 levels from the Problem Severity drop down box: “Very low”,“low”,“Medium”, “High”, “Very High” |
| **Precondition** | User is on the Job Problem page accessible via the Problems page and selecting a record from the DGV and clicking “Edit”. |
| **Basic Path** | 1. Render record data to fields. 2. Get Severity Level integer values 1 to 5. 3. Parse integer values to string elements in string array.    * 1 = Very Low.    * 2 = Low.    * 3 = Medium.    * 4 = High.    * 5 = Very high. 4. Foreach string of text in string array… |
| **Postcondition** | Display each string element in Drag drop box. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** | In the Problems page, the user will be able to see the string representations for each problem severity level in the DGV. |

### Attach Zip Folder

|  |  |
| --- | --- |
| **Operation Name** | Attach\_zip\_folder |
| **Trigger** | User/Admin clicks Add Folder button. The browser fires up the local file explorer whether on Windows, Mac, Linux. This will allow user to add folder to Problem. |
| **Precondition** | User must create a problem and access the Job Problem page via the “Create problem” button. |
| **Basic Path** | 1. User clicks button Add folder button. 2. System fires up the file explorer on the desired OS. 3. User selects a folder. 4. If total attachment is less than 2mb. |
| **Postcondition** | Upload the folder to the Problem record as a binary. |
| **Exception Paths** | The attempt may be abandoned at any time or the folder is too big to fit the file threshold. |
| **Alternative** | User selects an existing problem from the Problems DGV and clicks the Edit button to access the Job Problem page and upload a folder to the currently selected problem. |

### Attach Zip File

|  |  |
| --- | --- |
| **Operation Name** | Attach\_zip\_file |
| **Trigger** | User/Admin clicks Add Folder button. The browser fires up the local file explorer whether on Windows, Mac, Linux. This will allow user to add a file to Problem. |
| **Precondition** | User must create a problem and access the Job Problem page via the “Create problem” button. |
| **Basic Path** | 1. User clicks button Add File button. 2. System fires up the file explorer on the desired OS. 3. User selects a file. 4. If total attachment is less than 2mb. |
| **Postcondition** | Upload the file to the Problem record as a binary. |
| **Exception Paths** | The attempt may be abandoned at any time or the file is too big to fit the file threshold. |
| **Alternative** | User selects an existing problem from the Problems DGV and clicks the Edit button to access the Job Problem page and upload a file to the currently selected problem. |

### Create Chat

|  |  |
| --- | --- |
| **Operation Name** | CreateChat |
| **Trigger** | A user publishes a chat to do with the selected problem. |
| **Precondition** | User has selected the “problem” from the DGV on Problems page and has hit the “View Chat log” button and is now in the Job Problem page which has all the chats including the information fields for the problem. |
| **Basic Path** | 1. User enters the following required fields.  * ChatDescription * ChatPublishDate  1. System calls method Publish\_chat\_time\_and\_date. 2. User hits “Submit”. 3. System auto assigns RegistrationID of current user. 4. System selects UserFirstName and UserLastName from database where UserID are equal to RegistrationID. |
| **Postcondition** | Insert chat log record into database and select and display UserFirstName and UserLastName in the chat item. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### Publish Chat Time And Date

|  |  |
| --- | --- |
| **Operation Name** | Publish\_chat\_time\_and\_date |
| **Trigger** | User has hit the submit button, putting system in the process of submitting a chat item by calling this method. |
| **Precondition** | User is on the Job Problem page. |
| **Basic Path** | 1. Get date and time. 2. If ChatPublishDate data is empty. 3. Get current time and date from database. 4. Format data as “yyyy/mm/dd hh:mm:ss”. |
| **Postcondition** | Return the date and time in specified format. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### Read Chats

|  |  |
| --- | --- |
| **Operation Name** | Read\_chats |
| **Trigger** | User/Admin can see all the chats for the selected problem in Chat DGV. |
| **Precondition** | User/Admin has fired up the Job Problem page accessible by clicking the “View Chat log” button. |
| **Basic Path** | 1. Select all chat records from database 2. Make string array. 3. Foreach chat record in the string array 4. Get UserFirstName and UserLastName as chat title. 5. Get Time and date of chat. 6. Get ChatDescription. |
| **Postcondition** | Display the selected fields to user. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### Update Chat

|  |  |
| --- | --- |
| **Operation Name** | Update\_chat |
| **Trigger** | Chat gets to be edited based on the user/admin who has the registration ID and has clicked the “Edit” button next to their chat entry. |
| **Precondition** | User/Admin has fired up the Job Problem page accessible by clicking the “View Chat log” button. |
| **Basic Path** | 1. User clicks edit. 2. Fields in DGV are now accessible. 3. User edits a field. 4. System enables “Update”. 5. User clicks “Update”. 6. System selects record. 7. IF any field is not equal to what is in the database. |
| **Postcondition** | Update record in database. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### Delete Chat

|  |  |
| --- | --- |
| **Operation Name** | Delete\_chat |
| **Trigger** | Chat gets to be edited based on the user/admin who has the registration ID and has clicked the “delete” button next to their chat entry. |
| **Precondition** | User/Admin has fired up the Job Problem page accessible by clicking the “View Chat log” button. |
| **Basic Path** | 1. User clicks the selects their chat log and clicks the delete button. 2. Clear all text from the fields. 3. IF all fields are blank and are not equal to selected record id that is in the database. |
| **Postcondition** | Delete the record. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

### Confirm Required Info Is Not Empty

|  |  |
| --- | --- |
| **Operation Name** | Confirm\_required\_info\_is\_not\_empty |
| **Trigger** | The Admin has clicked the “Update” button or the “Create chat” button. |
| **Precondition** | User/Admin has fired up the Job Problem page accessible by clicking the “View Chat log” button. |
| **Basic Path** | 1. User has hit “Update”. 2. Foreach item in record. 3. If record cell is empty. |
| **Postcondition** | Put asterisk symbol next to field to tell user field is required. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** |  |

## 3.3 External Interface Requirements

* **User:**

Frontend software will be ASP.NET Core MVC with responsive Bootstrap UI and Entity Framework.

Backend software will be SQL Server database.

* **Hardware:**

Any operating system can run the front end. If used on mac then Backend will need to use a Docker image. We will target this towards many browsers that supports JavaScript, ASP, CSS3 and Bootstrap.

Backend can also run on all systems but needs to be inside a Docker image if run on Mac.

* **Software:**

|  |  |
| --- | --- |
| **Software used** | **Description** |
| Operating System | This will be able to run and maintained on any operating system. |
| ASP.NET Core MVC | We will use this design pattern to help build a lightweight web application that will render Web Form Controls at runtime and process client-server request. |
| Bootstrap | We will use a Bootstrap template because it is responsive to all devices we may view from. |
| Entity Framework Core. | We will use EFC for the business logic to simplify data access between ASP and SQL Server. |
| Database | To Create, Read, Update and delete records we have chosen SQL Server. |
| Auth0 | A third party authentication and authorization service we will use to manage users access. |

* **Communications:**

This project will support all types of web browsers. We are using ASP.NET Core Web forms to allow for login, creation of users, projects, and for bugs in those projects.

## 3.3 System Features

This system will store information on:

A set of **Users** with their: ID, first name, last name, status on if they lead, email, phone, address, postcode, country, mobile, state, Login name, Password.

A set of **Registrations** with their: ID, User ID, Job ID, Register date.

A set of **Jobs** with their: ID, Job ID, Title, Description, Start Date, Deadline, Completion status.

A set of **Problems** with their: ID, Job ID, Registration ID, Title, Description, Start date, Attachments, Severity level, Completion status.

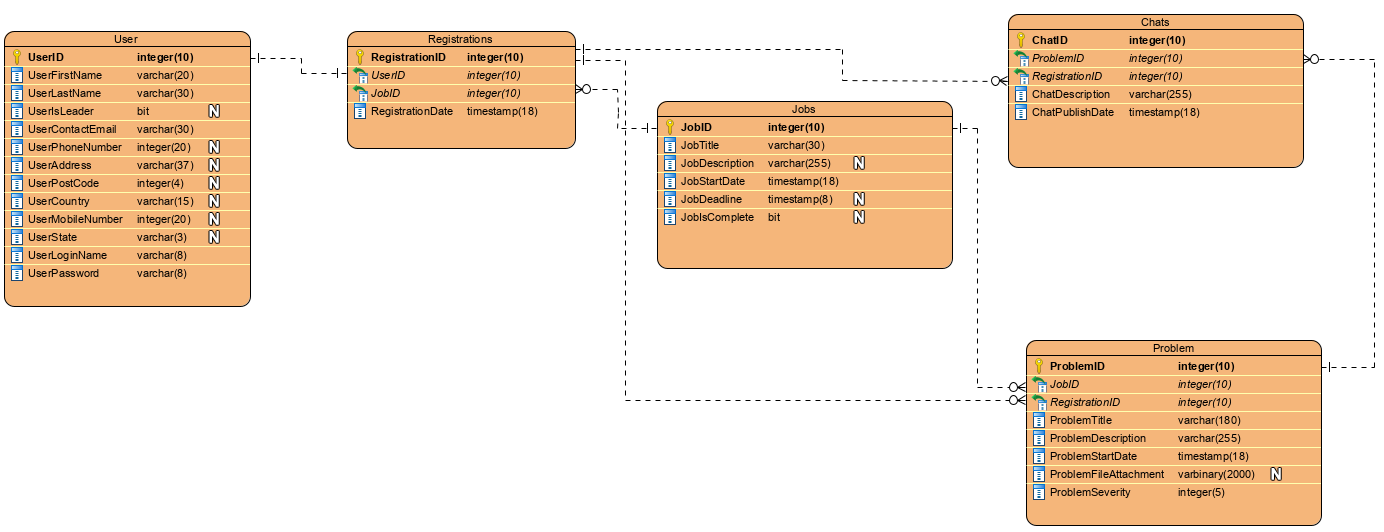
A set of **Chats**: ID, Problem ID, Description, Published date, Registration ID.

### 

## 3.4 Nonfunctional Requirements

The steps involved to perform the implementation of bug track database are as listed below.

3.5 Entity Relationship Diagram



* 1. UML Class Diagram



## 3.7 Security

We are using a third party service called Auth0 for handling the security of the frontend connection to our database. We will use it to manage universal authentication & authorization on the web so the user will be going through a login system with Auth0 acting as the intermediary security layer.

The only way data can be created updated and deleted is for users to be created in the system. By default we will have a “Lead developer” user who can decide who has to be created in the database in order to use the application. They will be defined by the IsUserLeader boolean value which will be true.