

**Operation Sundew**



November 14, 2019

Console Commander

https://www.consolecommander.net/

Contents

[1. Introduction 1](#_Toc25244293)

[1.1 Purpose 2](#_Toc25244294)

[1.2 Intended Audience 3](#_Toc25244295)

[1.3 Scope 4](#_Toc25244296)

[Benefits 4](#_Toc25244297)

[Objectives 4](#_Toc25244298)

[1.4 Definitions and Acronyms 6](#_Toc25244299)

[2. Overall Description 7](#_Toc25244300)

[2.1 User Needs 7](#_Toc25244301)

[2.2 Assumptions and Dependencies 8](#_Toc25244302)

[3. System Features and Requirements 9](#_Toc25244303)

[3.1 Functional Requirements 9](#_Toc25244304)

[3.2 External Interface Requirements 9](#_Toc25244305)

[3.3 System Features 11](#_Toc25244306)

[JobBackend Operations 11](#_Toc25244307)

[Entity Relationship Diagram 12](#_Toc25244308)

[UML Class Diagram 13](#_Toc25244309)

[3.4 Nonfunctional Requirements 14](#_Toc25244310)

# 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.

**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 database management system.

**ORM:** Relational Object Mapper.

**MVC:** Model View Controller.

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

# 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 screen.

User and Admin can both login to the Main menu and have access to users.

They will 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.

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, ConfirmRequiredInfoIsNotEmpty, NotifyUserOfCreation.

**Jobs**: CreateJob, UpdateJob, DeleteJob + DeleteAllProblems + DeleteAllChats, ReadJob, ReadJobs, SearchJobs, AutoInsertStartDate, ConfirmRequiredInfoIsNotEmpty NotifyAssignedUserOfJobCreation, NotifyAssignedUserOfJobUpdate, NotifyAssignedUserOfJobDeletion.

**Registrations**: RegisterUserForJob, AssignUserID, AssignJobID, AutoInsertStartDate.

**Problems**: CreateProblem, UpdateProblem, DeleteJob + DeleteAllProblems + DeleteAllChats, ReadProblem, ReadProblems, SearchProblems, ConfirmRequiredInfoIsNotEmpty, NotifyAssignedUserOfProblemCreation, NotifyAssignedUserOfProblemUpdate, NotifyAssignedUserOfProblemDeletion, ConvertProblemSeverityToText, AttachZipFolder, AttachZipFile.

**Chats**: CreateChat + PublishChatTimeAndDate, ReadChat, UpdateChat, DeleteChat, ConfirmRequiredInfoRIsNotEmpty, NotifyAssignedUserOfChatCreation, NotifyAssignedUserOfChatUpdate, NotifyAssignedUserOfChatDeletion.

### 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** | If the user admin has filled out the required fields:  • First Name  • Last Name  • Login Name  • Login Password  Then the user selects a create button.  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. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Alternative** | None. |

### 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. If the user has filled out the required fields:   • First Name  • Last Name  • Login Name  • Login Password   1. Click Update. |
| **Postcondition** | Any updated database cell will be replaced with the new data in a new data list. |
| **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** | None. |

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. |
| **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** | None. |

### 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** | None. |

* + 1. 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** | None. |

### 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** | None. |

### Confirm Required Contact Info Provided

|  |  |
| --- | --- |
| **Operation Name** |  |
| **Trigger** |  |
| **Precondition** |  |
| **Basic Path** |  |
| **Postcondition** |  |
| **Exception Paths** |  |
| **Alternative** |  |

### 3.2.8 Confirm Required Info Is Not Empty

|  |  |
| --- | --- |
| **Operation Name** |  |
| **Trigger** |  |
| **Precondition** |  |
| **Basic Path** |  |
| **Postcondition** |  |
| **Exception Paths** |  |
| **Alternative** |  |

### Notify user of creation

|  |  |
| --- | --- |
| **Operation Name** |  |
| **Trigger** |  |
| **Precondition** |  |
| **Basic Path** |  |
| **Postcondition** |  |
| **Exception Paths** |  |
| **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.

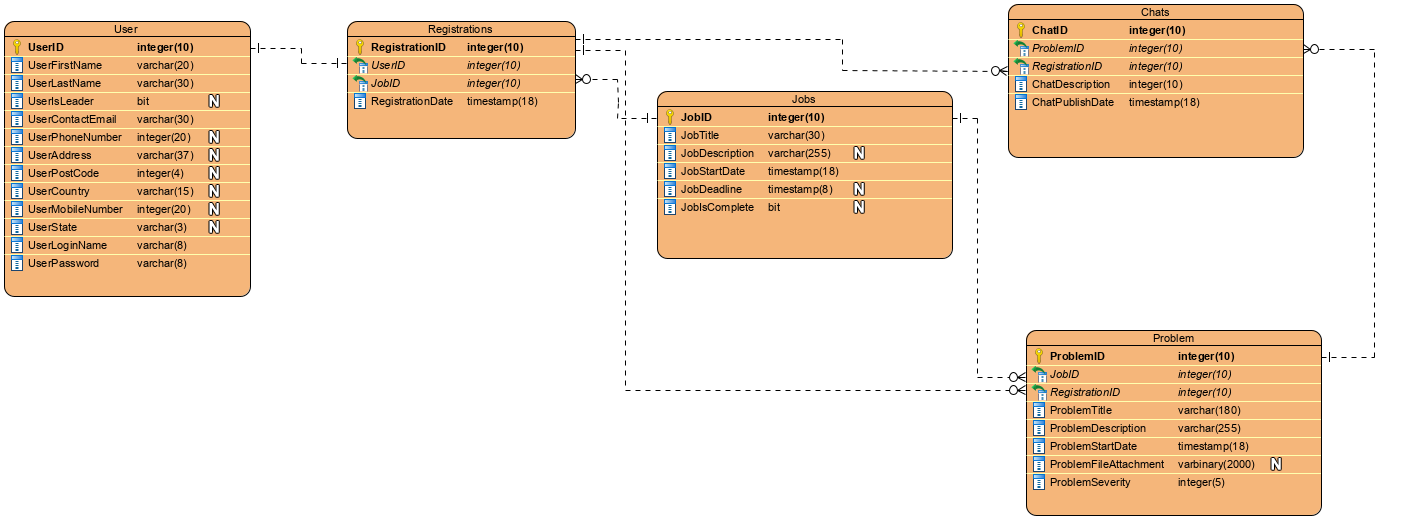
A set of **Chats**: ID, Problem ID, Description, Published date, Registration ID.

### 

## 3.4 Nonfunctional Requirements

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

3.5 Entity Relationship Diagram



3.6 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.