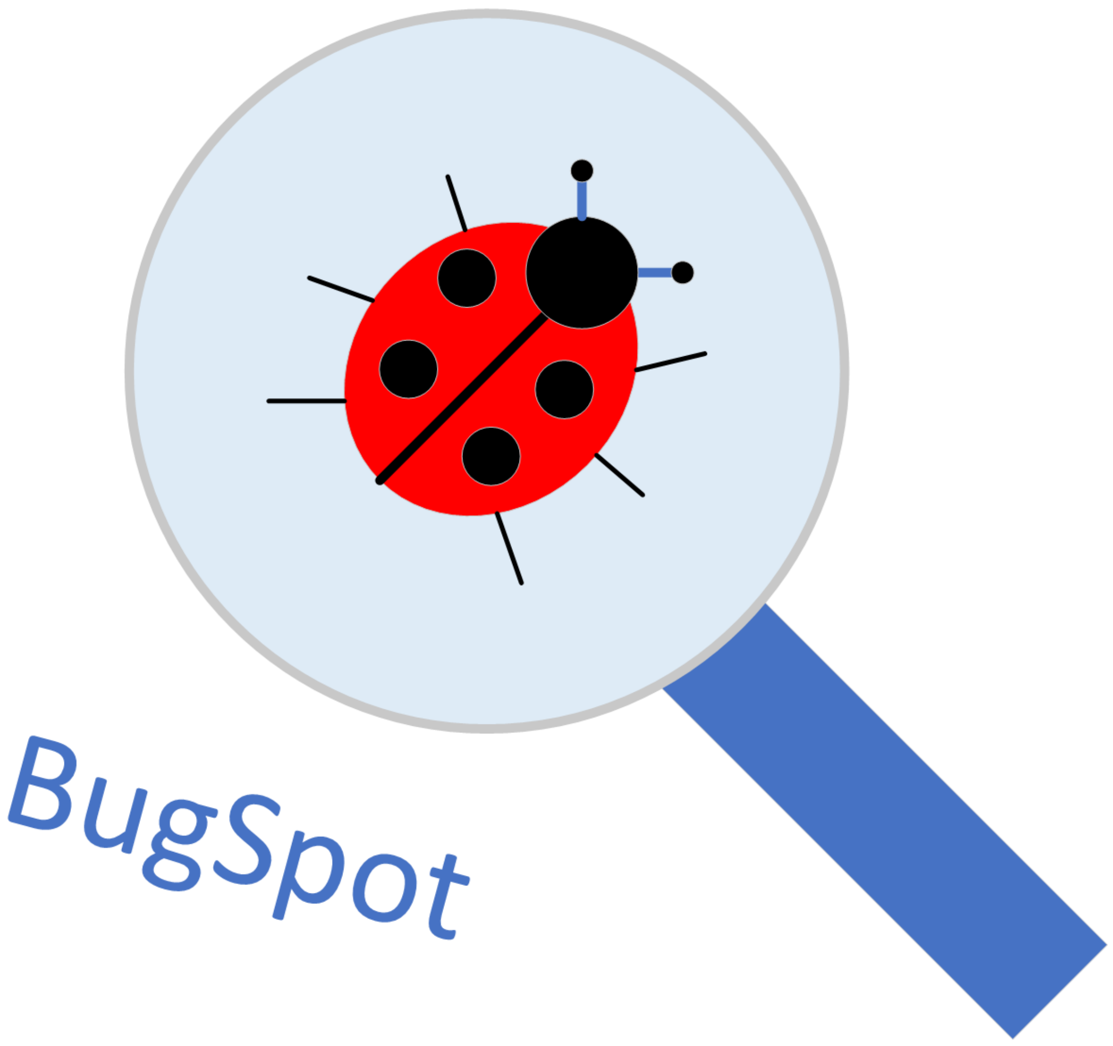
Software Requirements & Design



By USN Software AS

Table of Contents

1. Introduction

This document presents and overview, requirements and design for a bug tracking system which is targeted for developers needing feedback from their customers to make improvements to the product or service they are providing. In addition to users being able to report and send in bugs to the developers of a product or service, the developers should also be able to post and show their users new and exciting features they have been working on.

2. System Overview

This basic overview shown in Figure 2-1 is a simplified overview of the system and the different modules for easy understanding of how the system is structured. The figure shows the system tied together by the cloud, which is where all information and applications will be stored. Users will be able to access the bug reporting section where they can submit bug reports, and or request features. Bugs and features are separated with different forms. The developers marked with orange will access their own module when logging in with credentials. This module will have access to read and modify reports that’s been sent in. Mockups for graphical user interface can be seen in chapter 3.3 below.

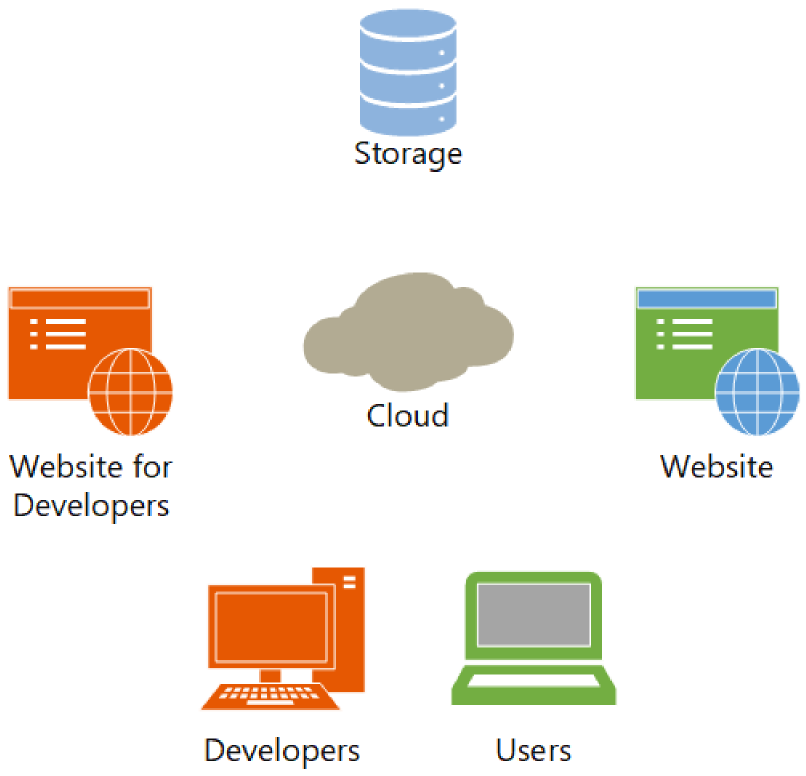


Figure 2-1: Basic system overview

2.1 Architecture

The bug tracker system is built up by three modules. In Figure 2-2 the modules are shown as web applications made using ASP.NET. To differentiate between access levels there has been added color coding to the system sketch, which indicates their level of permission. Administrators or Developers are marked with Orange. Orange access level is the highest level of access for normal intended use. Normal users, bug reporters are marked in Green. Green has access to reporting bugs, viewing status, and looking at new features.

Developers and Administrators will have their own module where they have full access over the reports submitted. Orange access level will have CRUD (Create Read, Update, Delete) access. All reports can individually be claimed or assigned to a developer to be investigated. This will give options for freedom and flexibility when it comes to tackling a bug report.

Users will be able to access two of the three modules, with green access level they will be able to submit reports in the bug reporting module while also being able to see the status of bugs in the Status module. The Status module will also be used to present new features and bug fixes. This sub module will also be available to those with green access level.

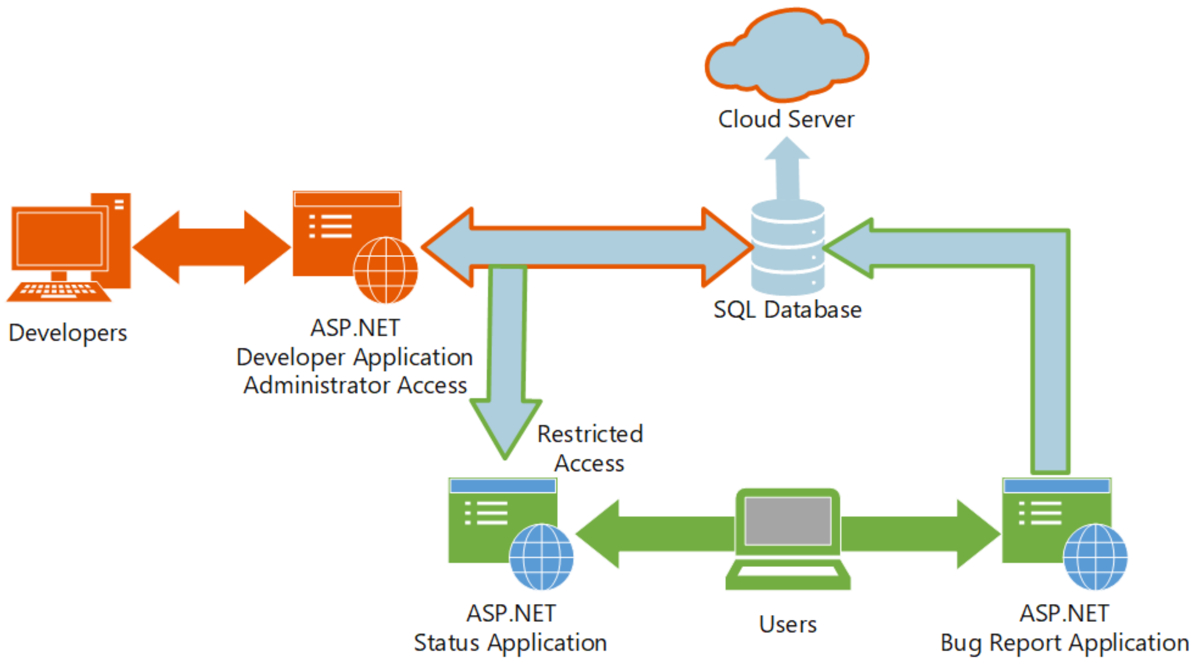


Figure 2-2: General architectural overview of the system

3. System Requirements

This chapter will highlight the requirements for the product, such as functional and non-functional requirements, user-interface specifications, user task flow, input/output and other data specifications, and interface specifications to other systems in greater detail.

3.1 Functional Requirements

Functional requirements define the function for the system or one of its components. For this project the functional requirements are:

- Data Storage
    - For storing of data, a database, consisting of multiple tables where the data itself is stored will be used. The structure of the tables is required to be made in such a way that certain data can be identified from other tables. Example for this project would be the ability to identify a reported bug status from the name of the bug via the use of foreign and primary keys.
- Data Collection
    - Collection of data is the main part of the user module. This will be done by filling in text fields with the relevant information such as name of the bug, what program/platform is the bug for, description on how the bug could be found or steps to replicate it, version of the program/platform. The information is submitted by clicking a button “Submit Bug”.
- Data Management
    - This module requires functions for manipulation of the database table content. Such functions are the ability to add new data to the tables in a similar fashion to the user module with the additional function of drop-bars where “Type” (bug or feature) and “Responsible Person” can be selected for input of data, the ability to delete data from multiple tables by selecting a bug or feature from a list and clicking the “Delete” button and the ability to edit already existing information by selecting a bug or feature from a list, clicking the “Edit” button which then fills out the input fields with selected information that can be changed manually and saved by clicking “Save” button.

- Viewing data
    - Certain data, such as the status of the reports sent in by a user, may be interesting for the user to view. A module that presents this data is therefore necessary.
- Search Functionality
    - The ability to search for bugs or features by their name, type, or date. This is implemented by entering search information in the search textbox and clicking the “Search” button, which will list the information that is relevant to the search input, whether it is name, type, responsible person, program/platform, or date.
- Sorting of viewed data
    - The listed data should have the ability to be sorted by name, date, type, responsible person, or program/platform version. This should be done by clicking on a bar at the top of the viewing list where the column names for name, date, type, responsible person, and program/platform version are. By clicking on, for example, name bar all listed data is sorted alphabetically in a descending order. When sorting by date or version the newest date and latest version is presented at the top of the list with older dates and versions being listed under in a descending order.
- User login
    - Login function is important as some users may want to see the status of their reports. The developers will need a way to manage the number of reports they are receiving; thus, they will also need additional functionality which should not be accessible by unauthorized individuals.

3.2 Non-functional Requirements

The non-functional requirements specify features on how the system should work, such as:

- Input errors
  - When submitting a bug or adding a new feature to the database certain fields must be filled out for proper storage of data. In case of the event where one or more of required input fields are not filled in or invalid input is attempted to be sent into the database must be detected and throw an error message telling the user to fill in the necessary information, in case of empty fields, or to insert data in a certain format, example would be attempting to insert date in format MM/DD/YY when the database only accepts DD/MM/YY.
- Authentication
  - Developers who will have access to the database and the ability to remove or change the data within, should have authorized login which creates a layer of security against misuse of the system.
- Web browser support
  - It is important that the website part of the system is compatible and functioning on different browsers. Some users may be using Google Chrome, others may prefer MS Edge, etc.

3.3 User-Interface Specifications

One of the most important requirements are the user interface with the system with a special emphasis on the graphical design, which should be simple to understand at first glance and intuitive to use. This part of document will present the suggested and simplified version of the user interface.

Upon entering the web application, the user is greeted with a Homepage where they can select various options, such as reporting a bug, requesting a feature or to log in to a higher level of access, see Figure 3-1.

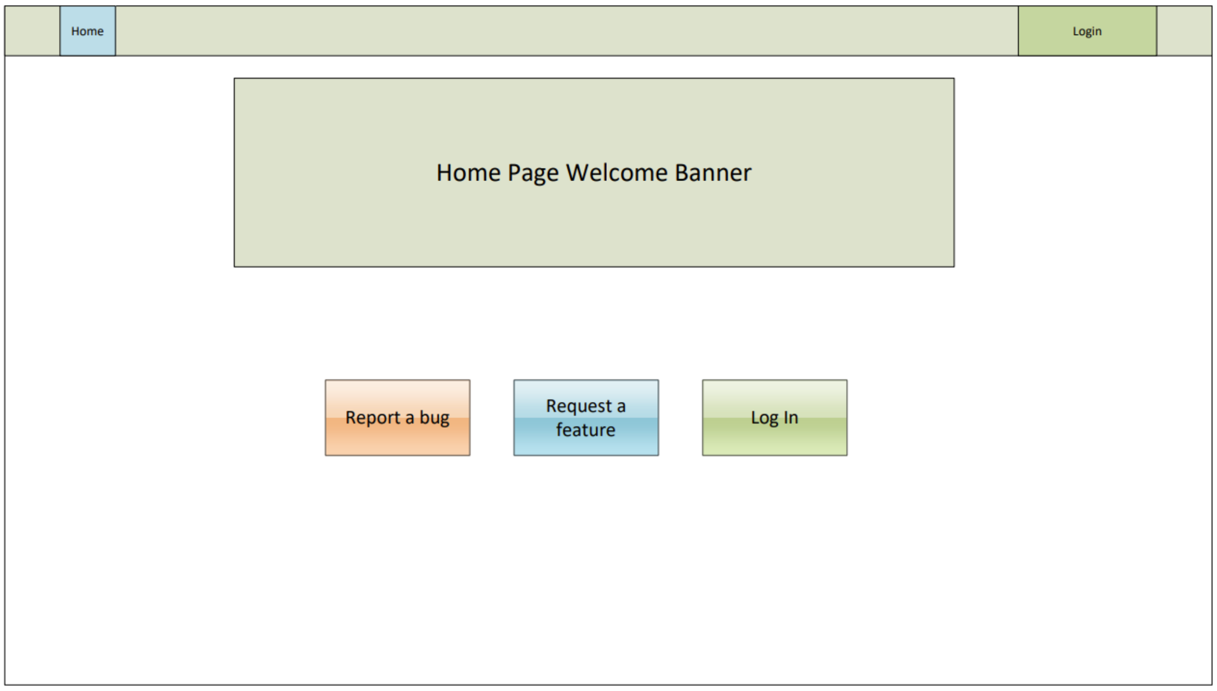


Figure 3-1: Home page graphical user interface (GUI) design

When the user goes to the bug report page, they are given a simple form to fill in, see Figure 3-2, with the relevant information and submitting it with the "Submit" button which then takes them back to the Home-page where a message thanks the user for submitting a bug or feature instead of the normal Home page welcome banner. The forms for both bug reports and feature requests are to be the same with only slight differences in the user description text.

The basic user is limited to the bug report and feature request pages. A developer who this system is designed for can manage the reported information in the system database by logging into their existing account as shown in Figure 3-4. There is also an option to register a new user to the system which prompts a new page for simple registration of username and password, see Figure 3-5. When a developer is logged in, they enter the developer page where additional functionality for management of data is presented as shown in Figure 3-3.

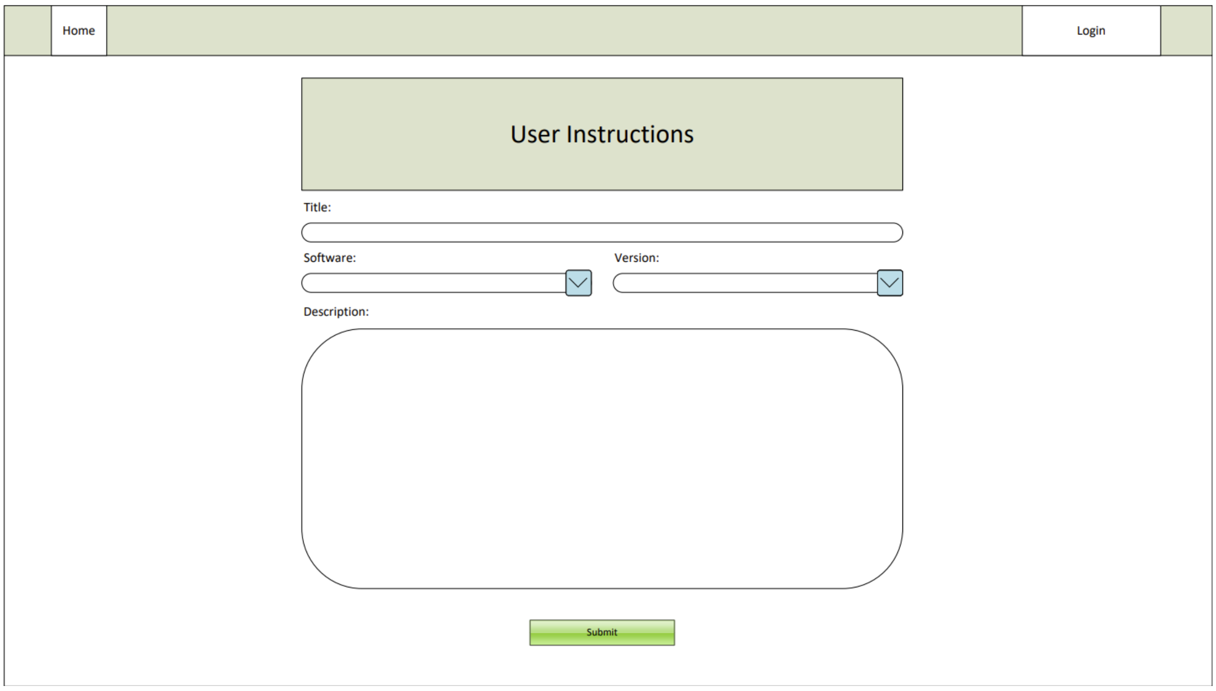


Figure 3-2: Report & Feature Request form page

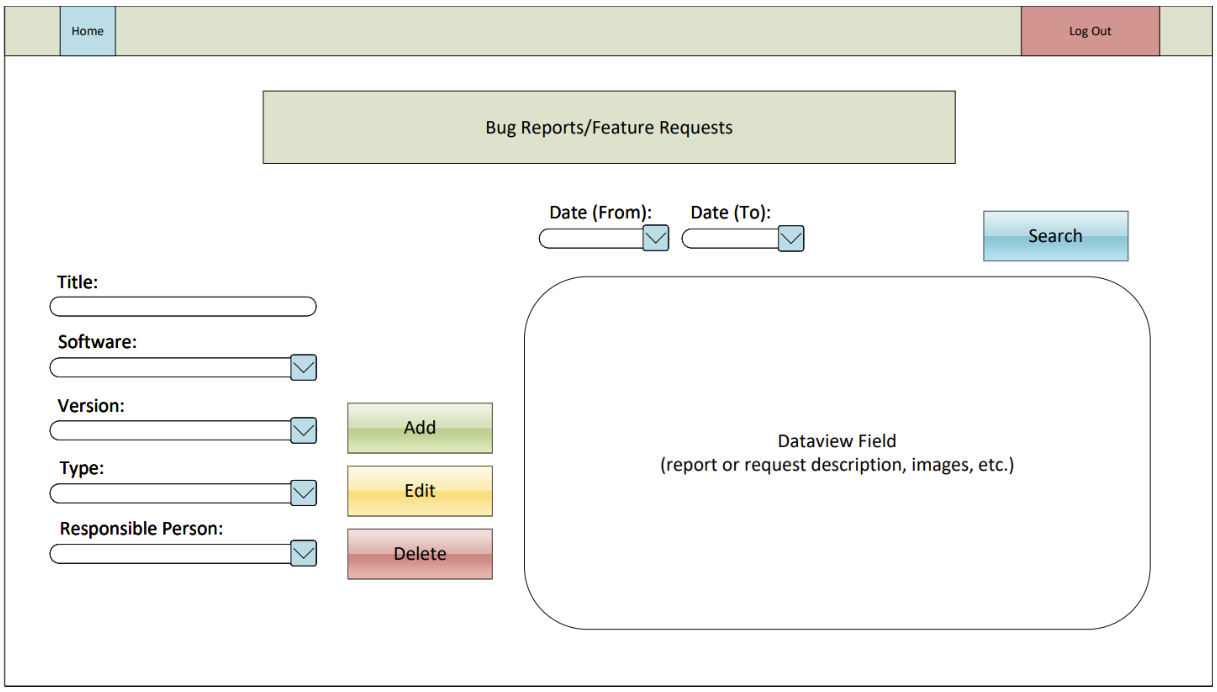


Figure 3-3: Developer system page

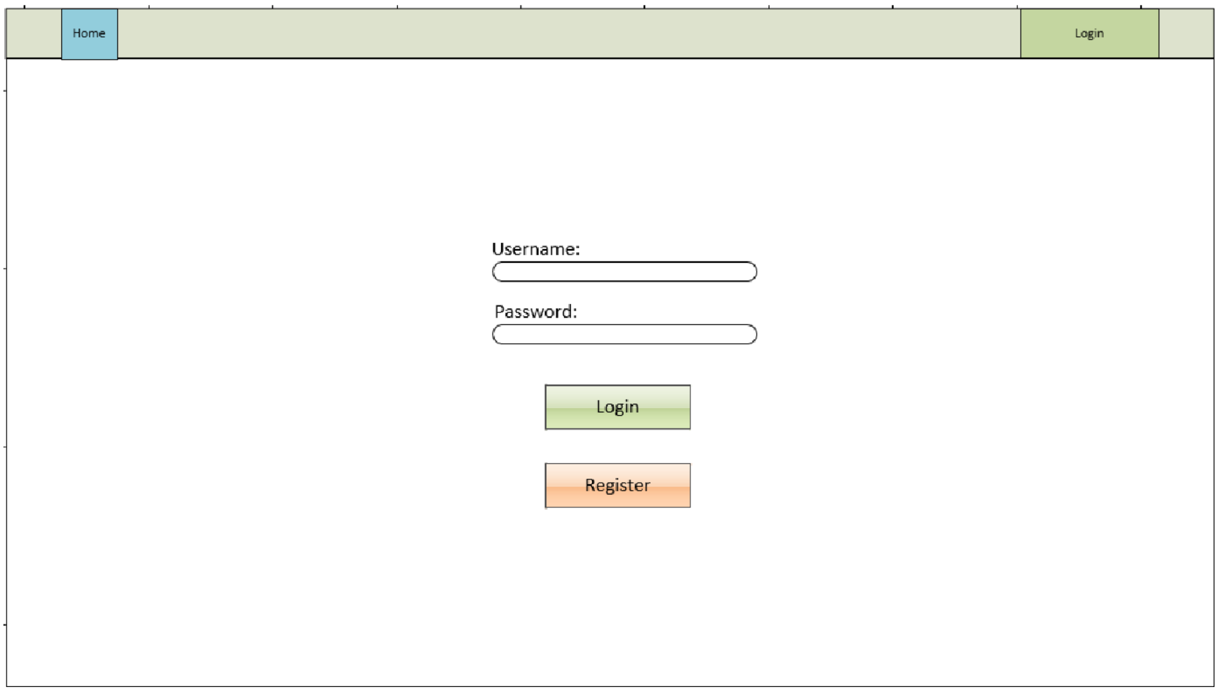


Figure 3-4: Login page for existing user(s)

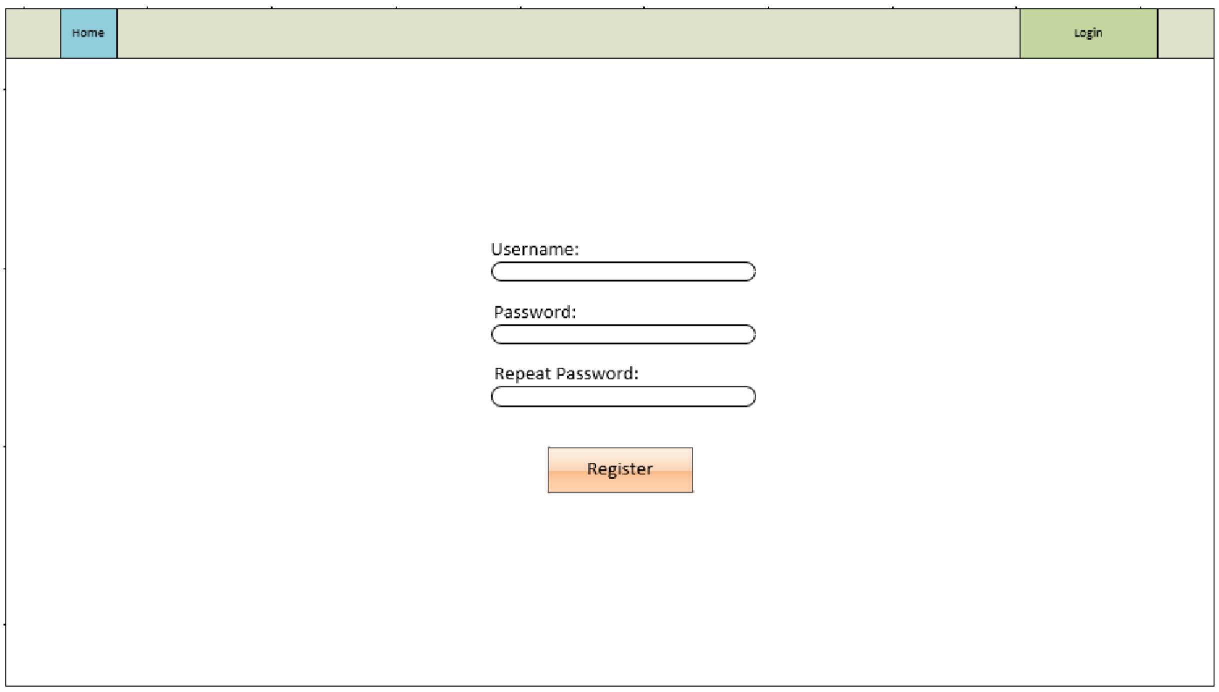


Figure 3-5: Register page for new user(s)

3.4 User Task Flow

This subchapter will present how the bug report system will work using flowcharts showing its behavior when browsing the web modules. Figure 3-6 shows a user task flowchart which highlights the interaction process between the user and the system, such as bug reporting, feature requesting and logging in.

Starting from the Homepage the user selects one of three paths (buttons) which lead to different processes. The bug reporting and feature requesting have very similar process where upon choosing bug report or feature request the user is sent to a fill-out form for the user to provide relevant information in the input fields and submitting it by clicking the "Submit" button. When the button is pressed a quick check on whether all necessary fields are filled out correctly (no empty fields, correct datatype, etc.). Upon successful submission the user is sent back to the Homepage where they are given a message thanking them for their report or request. Should an error occur during the submission process, like for example empty input fields, the user will be sent back to the form with a warning message describing the error, which in the above-mentioned example would be something along the lines of "Please provide the necessary information".

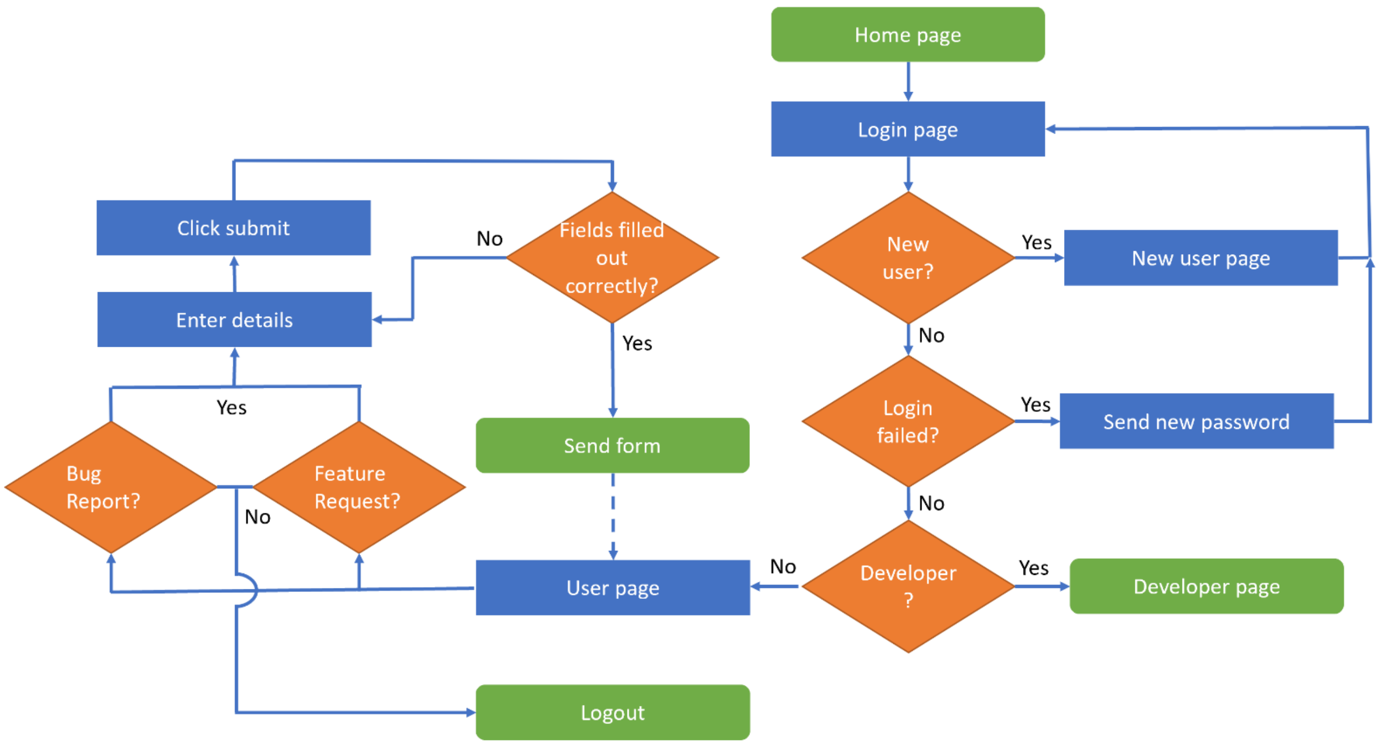


Figure 3-6: User flowchart

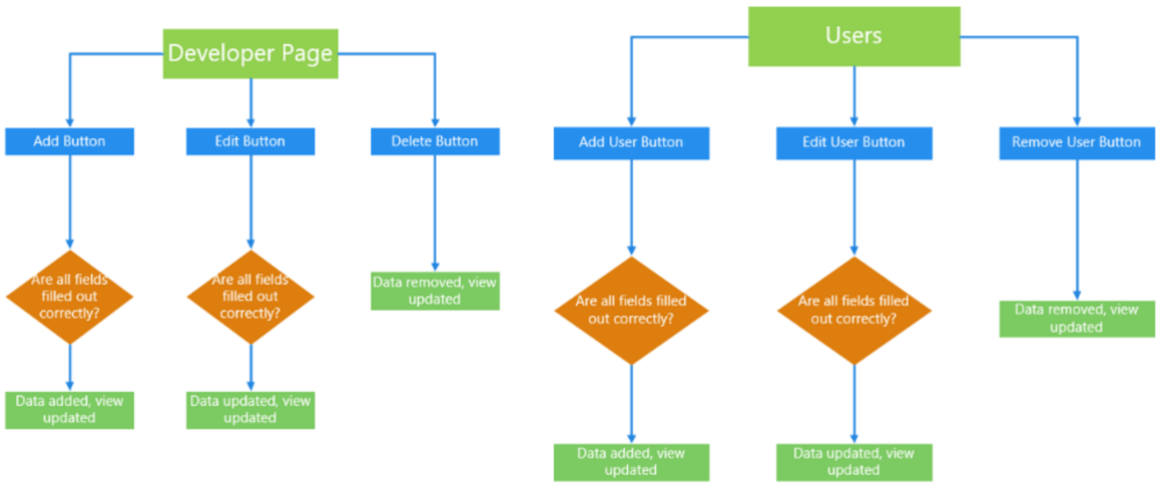
When a user selects the login button, they are then sent to the login authentication form where they must provide username and password to proceed. Additionally, a "Register" button is present for registration of new users, clicking it takes the user to a registration form which when filled out and submitted using the "Register" button sends the user back to the login page. Just as when using the report form, should some fields in the login or registration pages be filled incorrectly or left empty a warning message will appear, prompting the user to fill out the forms correctly.

Once the user login has been validated, the user is finally sent to the developer page where they have additional interactions with the system. Figure 3-7 shows the interactions with all the existing features for the system. Clicking the “Search” button will provide a table listing view in the data-field of all data stored in the system. The user can also provide inputs for date, software name, version, type, and responsible person to narrow down the search.

Selecting a row in the data-field and clicking delete will remove the row with all its information from the system and updating the data-field.

Clicking the “Edit” button with a row selected, will overwrite said row with left side input field data. Should the fields be empty or incorrectly filled in a warning message will appear instead.

Finally, the “Add” button will add the data from the left side input fields into the system database and update the data-field. Just as with the “Edit” button should the input fields be empty or filled in incorrectly a warning will be prompted.



Developer Page
Add Button
Edit Button
Delete Button
Are all fields filled out correctly?
Are all fields filled out correctly?
Data removed, view updated
Data added, view updated
Data updated, view updated
Users
Add User Button
Edit User Button
Remove User Button
Are all fields filled out correctly?
Are all fields filled out correctly?
Data removed, view updated
Data added, view updated
Data updated, view updated

Figure 3-7: Developer page user flowchart

3.5 Data Specifications

To prevent unwanted interactions with the database, such as custom injection queries or browsing of private data by the users (leaks), all interactions between the database and user or management modules will transact with specified procedures that only allow for a specific interaction (stored procedures). Viewing of multiple table content will be done in a similar fashion by using custom view procedures.

Additionally certain types of data are unwanted in input fields. An example being “Title” input should contain string data type, thus any input from this input field is going to be converted into string data type before adding it to the database.

4. Database design

This chapter will present the database design and plans for how it will be used in the system.

The main objective of the database is to keep records of the reports that are registered in the system. This is done by sending in a fill-out form with information which is then stored in the “REPORT” table columns. In addition to problem description other data such as the software that the report is about and whether the report is about a bug or a request for a feature.

In Figure 4-1 the table structure is shown with the following tables:

- REPORT table
- TYPE table
- SOFTWARE table
- STATUS table
- REPORT_PERSON table
- RESPONSIBLE_PERSON table
- PERSON table
- USER table

The following sub chapters will describe the table structure and reason for their implementation in the model.

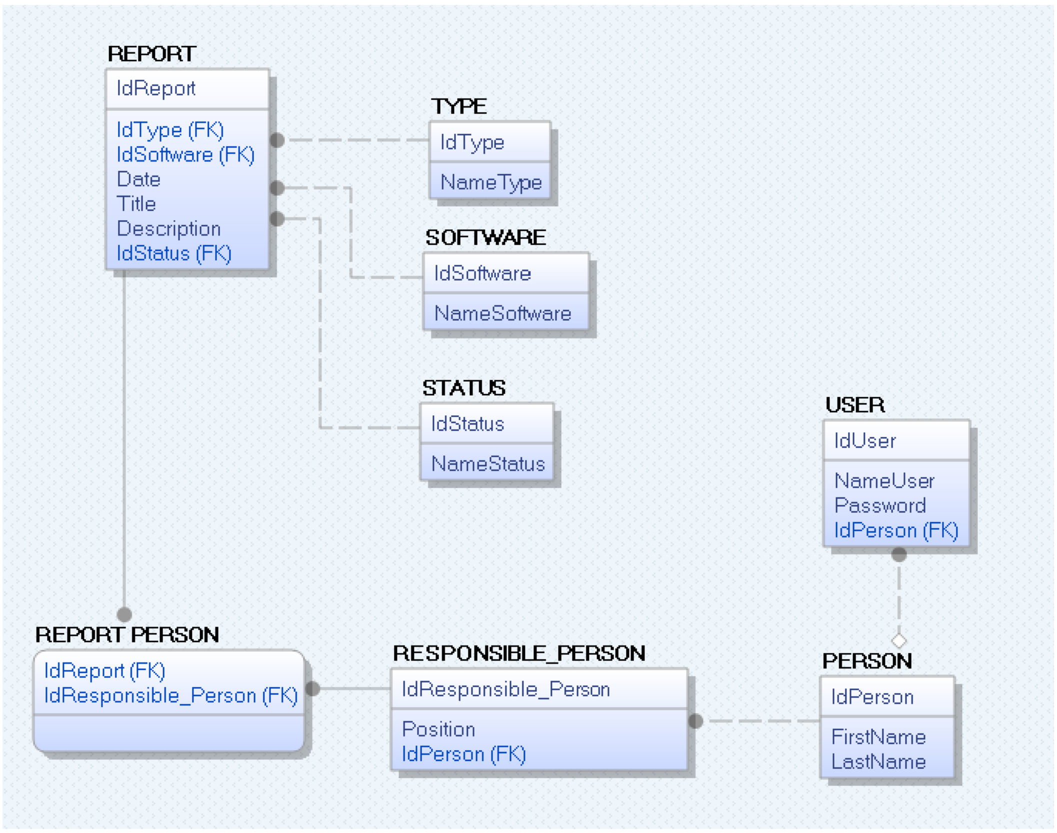


Figure 4-1: Database table model

4.1 Database tables

The key component in the entire system is the “REPORT” table where most of the inputted data will be stored. The table is made to have a unique ID number “IdReport” which keeps track of every unique report and helps to identify them. A different solution to this would have been using the title as the identifier of the report, however as there is a possibility that some users may happen upon the same software issue or request the same feature with the same title name a numerical value is chosen instead.

Other components in the “REPORT” table are “Date” which will showcase the date and time of when the report was submitted, which is very useful when keeping track of errors in multiple software. The description column contains the content of the report. Expected content is a text description, but images or even file uploads may be a possible future upgrade for the system.

The additional tables “TYPE”, “SOFTWARE” and “STATUS” are there to supplement the “REPORT” table with information, some of which is not to be seen by the person(s) sending in the report, like for example the responsible person (more on that in the next sub-chapter).

The aforementioned tables are designed to be split apart from the main table as their contents are not to be decided by the end user and can be tailored after the user and to limit some choices, for example if the bug tracking will be used only to track bugs of a set of software the person sending in the report should only be able to report a problem about that particular software and no other. The “TYPE” table supplements the “REPORT” table with what type the report is, whether it is a bug report, a feature request, or some other kind of user-specified report type. Similarly, the “STATUS” table supplements the “REPORT” table with status of the report, examples being “New”, “Under Assessment”, “Fixed”, etc.

One of the important features of the database is the ability to assign a person from a list of employees to work on a reported problem. This is done by first having a listing of people in a table of its own called “RESPONSIBLE_PERSON”. The identifier from this table, along with the identifier from the “REPORT” table are taken and put in the “REPORT_PERSON” table where a connection between a report and a responsible person is established, allowing to identify the person responsible for each report who has an assigned person. The main reason why an entire table is used for this is to resolve a “many-to-many” connection as many reports can be assigned to many people and many people can be assigned to many reports. Additionally, a table “PERSON” exists to prevent possible identification of an employee’s name in case the report table is hacked. The same reasoning is also made for the “USER” table which will hold the login information for registered users.

5. UML

This chapter will present user interactions with the system via use case diagram and sequence diagram. Additionally, this chapter will also contain a class diagram which unlike the other two is focused more on the structure of the system.

5.1 Use Case Diagram

As mentioned in the system architecture, chapter 2.1, the entire system consists of 3 modules: Report module, Status Module and Administrative module. Figure 2-1 shows the general interactions between the modules, the normal end user, and the developer end user.

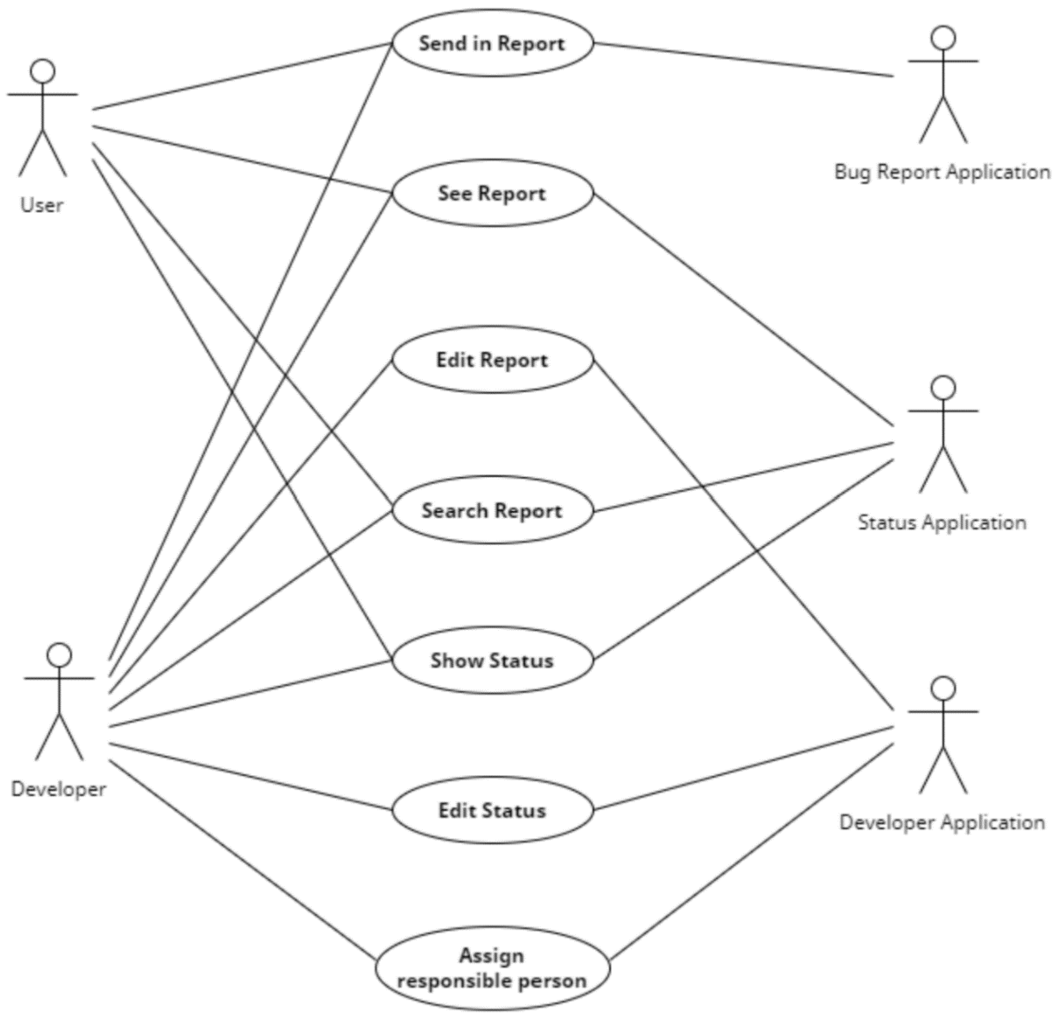


Figure 5-1: Use Case Diagram

5.2 Sequence Diagrams

This subchapter will present a closer and more detailed view of interaction for the BugSpot reporting system. The focus is to showcase how the objects interact with one another and with the user.

Figure 5-2 shows the sequence of interactions taken by a typical user who submits a report, gets a confirmation message that the report is successfully registered and is then able to view the status of said report or search for a specific report which they have submitted previously.

Additionally, the user may visit the “Features” site where they submit a request form for a feature and browse the latest features added. Figure 5-3 show the sequence diagram for administrators and developers.

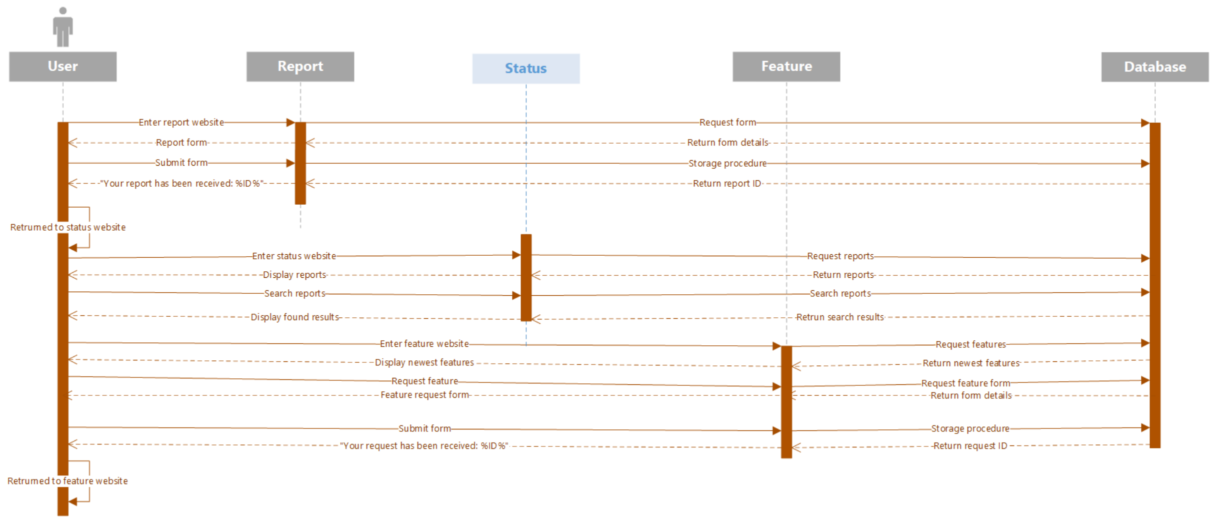


Figure 5-2: Sequence Diagram

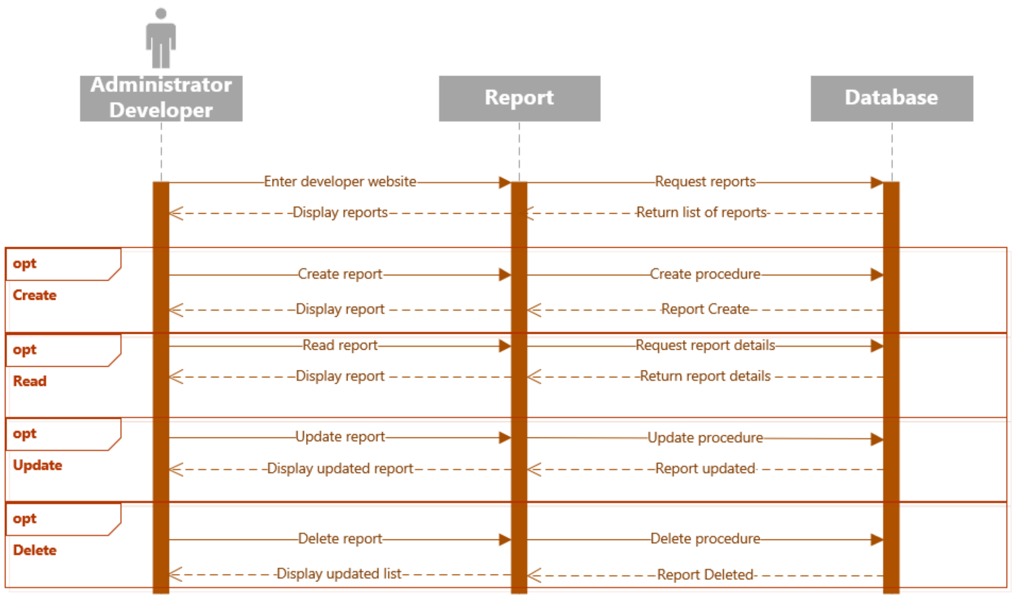


Figure 5-3: Sequence diagram for Administrators and Developers

5.3 Class Diagram

The class diagram presents a detailed view of the program regarding the actual code, specifically object classes, relationships between classes and the various properties and methods they have.

Most of functionality of BugSpot revolves around some type of form to fill in, whether it is a registration, login, or report, all the fill-out forms can be generalized simply as “Form”. Figure 5-4 shows that all the different forms for use of the system are inherited from the abstract class “Form”.

On the left side of Figure 5-4 the user and developer classes are shown to inherit from the class “Login”. This is to differentiate between the two levels of access. An interface requiring all sub classes who inherit from “Login” to also implement a parameter called “UserType”, which defines the access level.

To the right side of Figure 5-4 is the report form inheritance structure. All report forms can be generalized as “Report”. Bug reports and feature requests are very similar to each other and as such only minor differences are noticeable. Additionally, several interfaces are bound to both “Report” sub classes, requiring the parameters for type of report, status of the report and responsible person for the report.

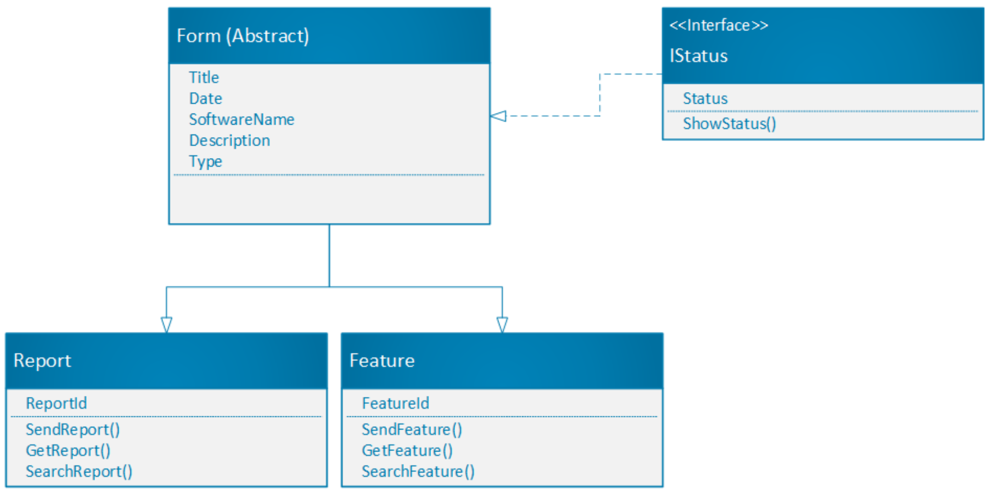


Figure 5-4: Class Diagram

6. Security

Data security is a very important aspect of any web-based application which collects user input data. This application is no exception as certain laws and regulations, namely the GDPR apply in this case. The core aspects of GDPR are:

- Lawful, fair, and transparent data processing
- Limitation of purpose, data, and storage
- Rights to information, access, and erasure of personal data
- Consent for data collection
- Personal data breaches, data protection, transfers, and privacy

To quickly summarize these points, the company running a data collecting website must ensure that the user is fully aware how their shared data, such as name, phone number, email address, location etc. will be used by the company whether it is for identification, case study or other uses they must be transparent for the user. Data collection must be clearly consented to and documented for both parties, such as a contract like for example a checkbox that must be manually ticked off by the user. The end user also has the right to see what data is collected, demand access and use of their own data and demand to erase parts or all collected data about them. Additionally, the company who collects the data from users must ensure that sensitive data they have collected, such as information which can help identify the user must be secured from potential hacker attacks or leaked due to human error, example being sending contact information of a user to the wrong email.