**Section 1 Introduction**

**1.1 Project Description**

Implement a streamlined process for users to submit trouble tickets to appropriate agent, and track ticket status until a resolution is reached.

**1.2 Solution Description (This is a high level description created from Design Overview)**

My SQL database will hold tickets and user's login information. PHP will be used as server side scripting language to manipulate database and create website for both portal A and B. The desktop application will be installed on user's desktop and query database for ticket information.

**1.3 Development guidelines**

1. Try not to hinder other developers
   1. Make sure code compiles before committing it.
   2. Make sure code passes all tests (if they exist) before committing.
   3. If you need to check in broken or incomplete code, use a branch, or somehow minimize the impact on other developers.
2. Commit code that is neat, portable, and documented
   1. Indent using spaces instead of tabs, usually 4 spaces, just be consistent.
   2. Use braces consistently (cuddled or not) within a file.
   3. Use braces for all if-statements, including one-line conditionals.
   4. Use 80-character lines max (this is very important for code).
   5. Use appropriate, descriptive names for classes and variables.
      1. Camelcase for variable names, starting with lowercase (e.g., myVariable).
      2. Camelcase for class names, starting with uppercase (e.g., MyClass).
   6. Use javadoc or equivalent comments for every class and method
      1. Explain the purpose and intent of a class and how it fits into the overall architecture when writing docs.
   7. Remove extraneous code that is not used (classes, and methods in classes).
3. Communicate with other developers
   1. Commit frequently, in small and logically related patches with good log messages.
   2. Use a transparent build system to expose all dependencies and requirements to build a project. We like [Ant](http://ant.apache.org/), but other build tools are possible too.
4. See <http://techfragments.com/css-style-guide-examples/> for other reference.

**1.4 Standards / Best practices to be followed**

https://code.tutsplus.com/tutorials/top-15-best-practices-for-writing-super-readable-code--net-8118

1. Commenting and documentation.
   1. Have a clear explanation for exactly what your method, class, chunk etc will be doing.
2. Consistent indentation.
3. No obvious / unclear comments.
4. Group Coding.
   1. Keep similar code grouped together for consistency and readability.
5. Consistent naming convention
   1. See Development Guides, 2.
6. Don’t repeat yourself.
   1. Have code have a clear a definitive purpose.
7. Avoid multi level nesting.
8. Limit line length.
9. Organize files and folders.
10. Consistent temporary names.
11. Capitalize SQL related words.
12. Separate code from data.
13. Alternate syntax inside templates.
14. Know the difference between object oriented vs procedural code.
15. Read open source code.
16. Produce clean code, reformat unclean code.

**Section 2 Assumptions and Constraints**

**2.1 Assumptions**

1. Users will appropriately categorize their issues
2. Administrators will appropriately assign priority

**2.3 Constraints**

1. The system will not review the submitted tickets for validity, agents must review them manually and determine the appropriate action to take, if any.
2. The system will not sort tickets by anything other than priority.
3. The system will not notify admins when a new ticket is submitted.
4. Desktop Application will be pre installed on user's Windows computer

**Section 3 Design Overview**

**3.1 Architectural Style**

1. The system will be based upon a client/server architecture

**3.2 Technologies being employed**

1. MySQL Database
2. SQL Alchemy (Object Relational Mapper)

**3.3 Programming Language**

1. PHP - Used for connecting web portals to database and testing
2. Python- Used for Desktop Application
3. SQL/SQLAlchemy - Used in creation and manipulation of database

**3.4 Development environment**

1. Various Text Editors
2. PHPMyAdmin
3. MySQL Workbench

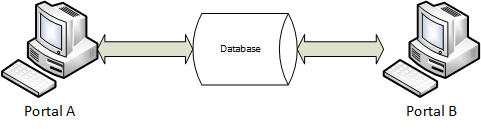
**3.5 Runtime Environment**

1. Windows 7, 8, 10

**3.6 General approach including things like security or logging errors**

1. User authentication and security will be handled by a users table in the SQL database Portal A and Portal B will have different rights as listed in the Access matrix
2. If a user has a login error a popup will appear with contact information for whom to contact. This error popup will also appear should a user input an invalid ticket number or ticket not found.

**3.7 Diagram showing how the different technologies fit together**



**3.8 Testing methodology**

1. Regression Testing with The MySQL Test Suite

**Section 4 Application Architecture**

**List of programs including description of what they do and why**

1. MySQL: open source relational database, Structured query language. Used to hold tables and information about tickets created on WebPortal A and have them accessible for agents from WebPortal B. Chosen because MySQL works well.
2. PHP: Hypertext Preprocessor, widely-used open source general-purpose scripting language that is suited for web development and can be embedded into HTML.

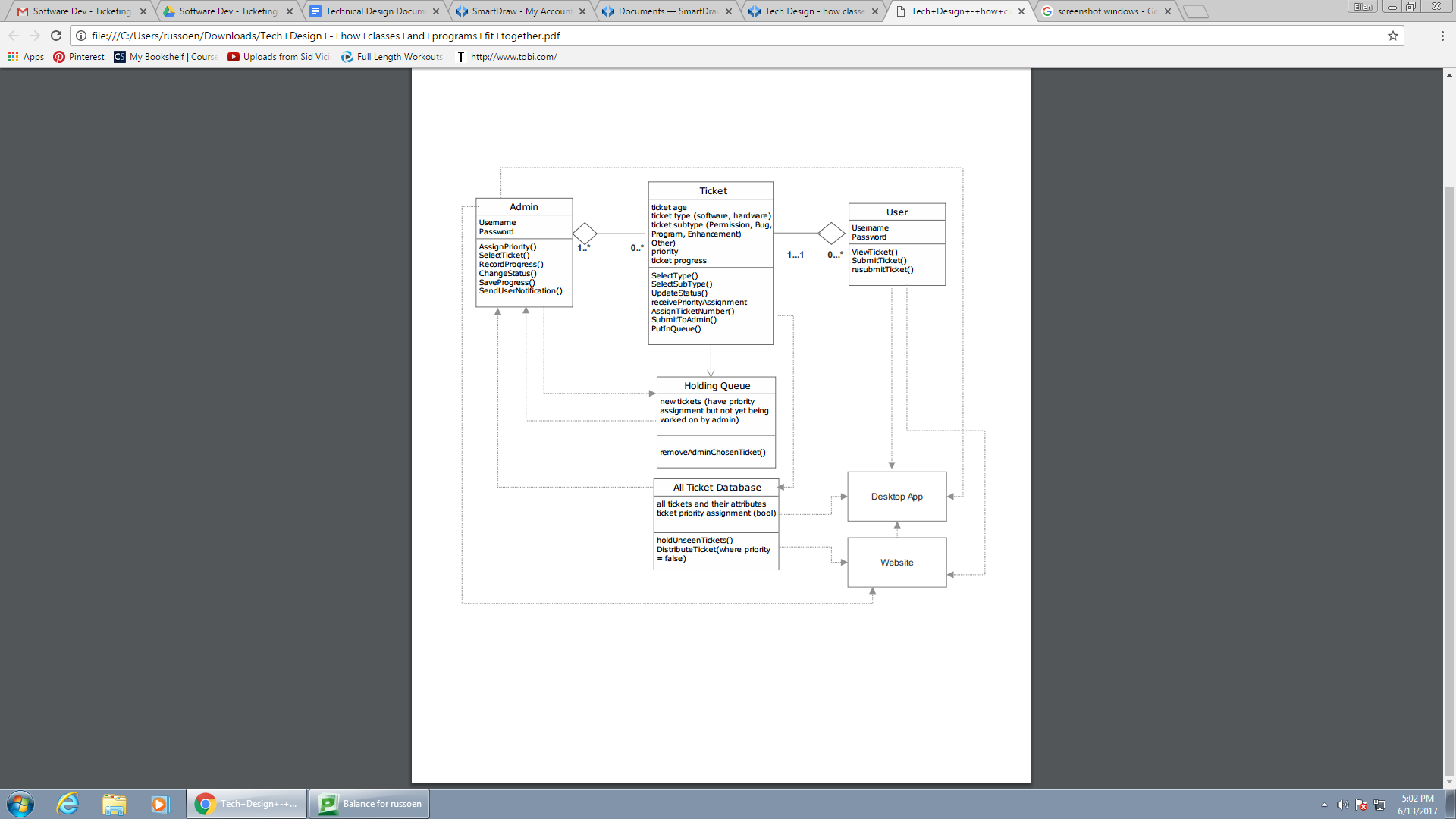
**List of classes including all public, private and protected members**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class Name** | **Description** | **Public** | **Private** | **Protected** |
| User | This class holds the user information and functions that users can perform. | Variables:  String username  <List> SubmittedTickets  Functions:  ViewTicket() | Variables:  String Password  Functions:  SubmitTicket()  resubmitTicket() | Variables:  Functions: |
| Admin | This class holds the admin information and functions that admin can perform. | Variables:  String username  List <int> TicketsWorkedOn  Functions: | Variables:  String Password  Functions:  AssignPriority()  SelectTicket()  RecordProgress()  ChangeStatus()  SaveProgress() | Variables:  Functions: |

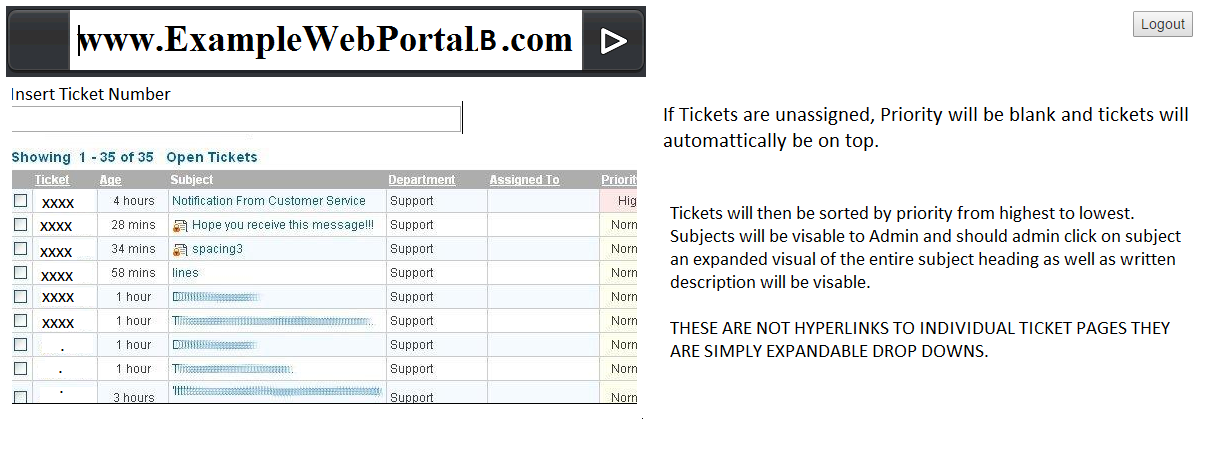
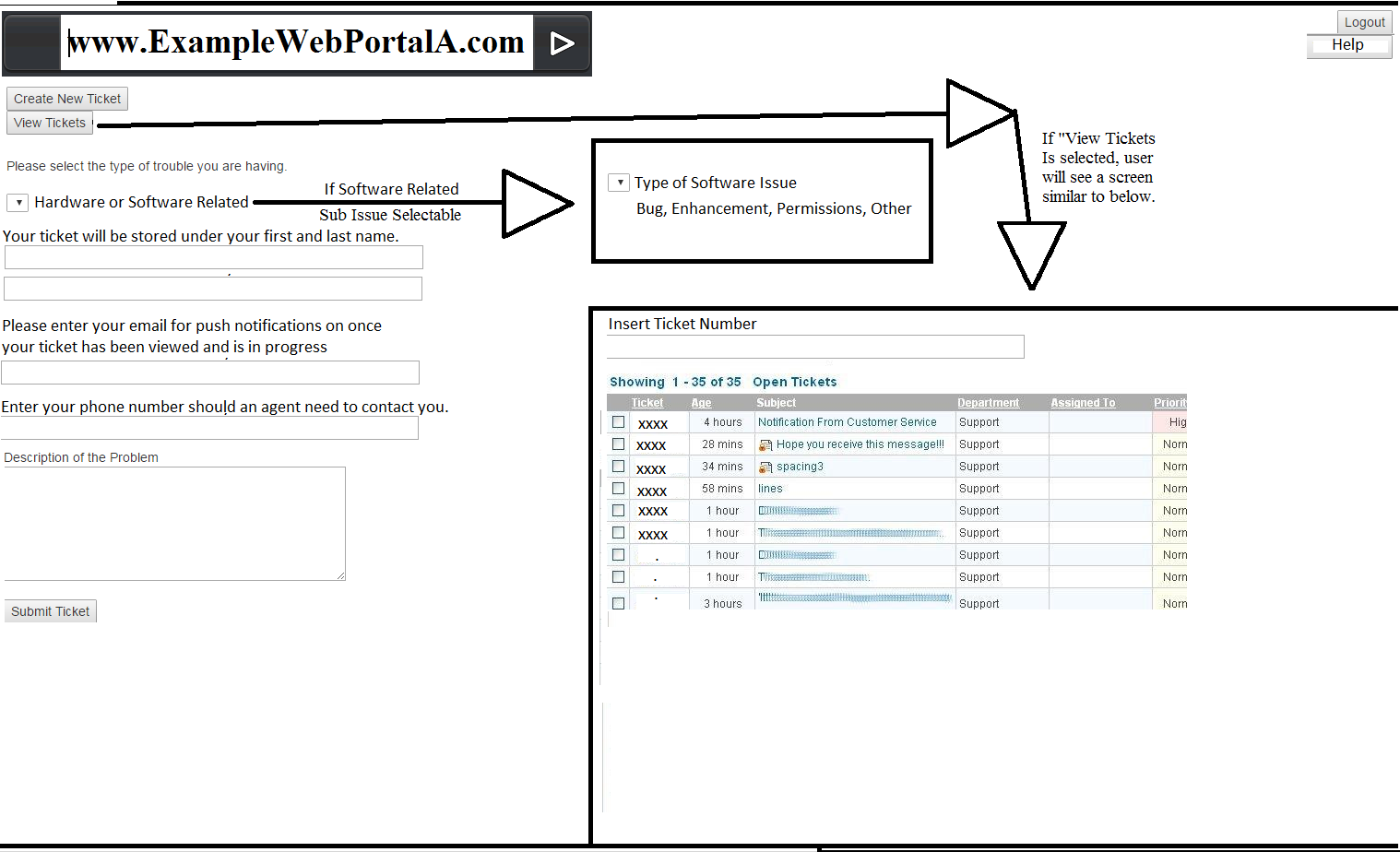
**List of structures, enumerations, tables, etc…**

* Tables in Database
  + Users
  + Tickets
* Structures
  + User
    - String: Username
    - String: Password
    - Login()
    - CreateTicket()
    - CheckTicket()
    - ReSubmitTicket()
  + Admin
    - String: Username
    - String: Password
    - Login()
    - AssignPriority()
    - SelectTicket()
    - RecordProgress()
    - ChangeStatus()
    - SaveProgress()
    - SendUserNotification()
  + Ticket
    - Int: Ticket Number
    - Enum: Issue Type
    - Enum: Issue Subtype
    - Enum: Status
    - Enum: Priority
    - Enum: Explanation
    - List<String>: Progress Notes
    - User\*: Ticket Owner
    - Admin\*: Admin (person working on ticket)
    - SetType()
    - SetSubType()
    - UpdateStatus()
    - AssignTicketNumber()
    - SubmitToAdmin()
    - PutInQueue()
* Enumerations
  + Issue Type {Software, Hardware}
  + Issue Subtype {Bug, Permission, Enhancement, Other}
  + Ticket Status {New, In Progress, On Hold, Completed}
  + Priority {1, 2, 3, 4, 5}
  + Explanation{user’s explanation of the issue}

**Diagram of how classes and programs fit together**



**Application GUI Example**



If you are implementing security, describe (access matrix)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Matrix** | | | | |
| **User** | **Ticket Submission** | **Ticket Tracking** | **Ticket Information** | **Archived Tickets** |
| Portal A User | submitTicket() | getStatus() | getTicketInfo()  updateTicketInfo() | N/A |
| Portal B User | N/A | getStatus()  updateStatus() | getTicketInfo()  updateTicketInfo() | getTicketInfo() |

**Section 5 Database Architecture**

* List of tables including attributes and details to create the tables

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Table Name** | **Attribute** | **Data Type** | **Required** | **Field Length** | **Default Value** | **Notes** |
| Ticket | User | Pointer | Yes | N/A | Not NULL |  |
| Ticket ID | Int | Yes | 6 Characters | Not NULL | Assigned  Incrementally |
| Category | Choice | Yes | N/A | Not NULL | Drop Down  Menu of choices |
| Sub-Category | Choice | Yes | N/A | NULL | Drop Down  Menu of choices |
| Description | String | Yes | 150 Characters | NULL |  |
| Priority | Choice | Yes | N/A | Not NULL | Drop Down  Menu of choices |
| Ticket Status | String | Yes | 10 | Open |  |
| Assigned Admin | String | Yes | 20 | NULL |  |
| Users | Employee ID | Int | Yes | 10 Characters | Not NULL |  |
| Username | String | Yes | 20 Characters | Not NULL |  |
| Password | String | Yes | 30 Characters | Not NULL |  |
| Email Address | String | Yes | 30 | Not Null |  |
| First Name | String | Yes | 20 | Not Null |  |
| Last Name | String | Yes | 20 | Not Null |  |
| Phone Number | Int | Yes | 20 | Not Null |  |
| Administrator | Bool | Yes | 1 | Not Null |  |

List of views

* + None

List of stored procedures

* + Functions internal to the database
  + Archive Ticket
  + Send Notifications
  + Calculate Open Ticket Time

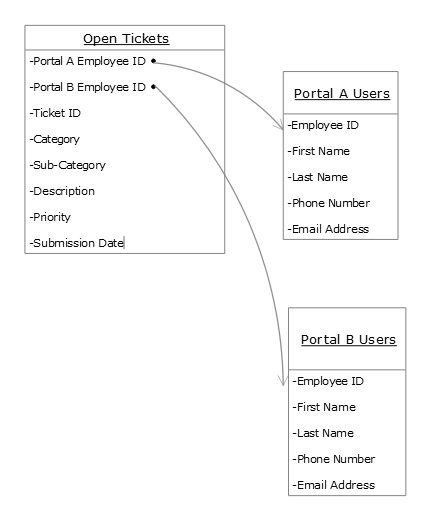
List of triggers

* + Ticket Notifications -when admin is assigned a ticket
  + Archive Ticket - When ticket is closed, change ‘Archive’ from 0 to 1
  + Ticket completion notification - sent to users when their ticket is resolved

List of jobs

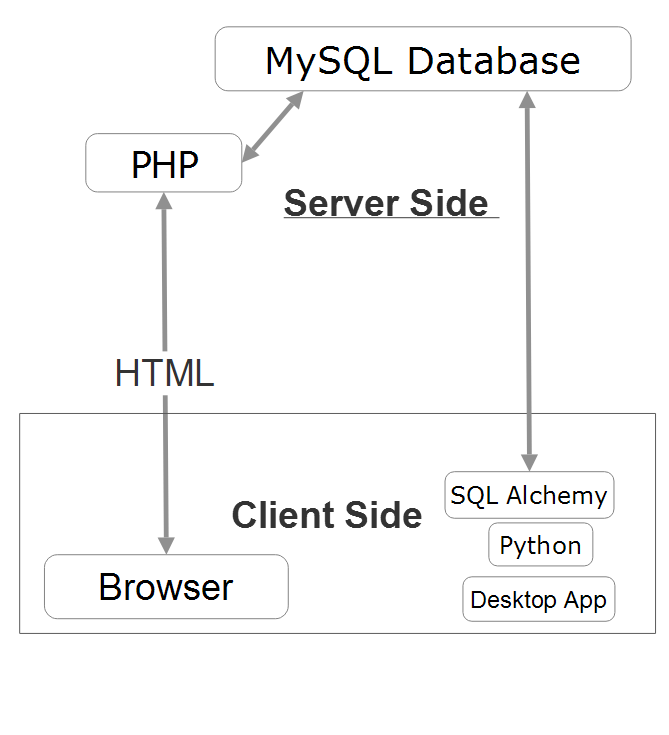
* + N/A

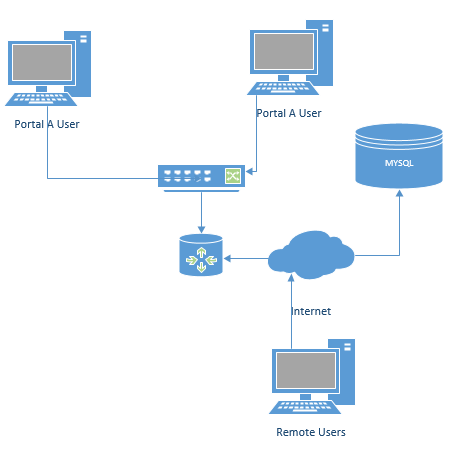
Diagram of table relations



**Section 6 Network Architecture**

* Diagram of how the different components fit together





**Section 7 Appendix**

**Terms and Definitions**

|  |  |
| --- | --- |
| **Term** | **Definition** |
| **Ticket Management System** | The product being described. |
| **Project** | Activities that will lead to the production of the Ticket Management system |
| **User** | This is the person with the issue who is submitting the ticket. They are not expected to have technical knowledge. They will access the client side web portal, where they will be able to submit and track their ticket. |
| **Use Case** | A list of actions defining the interactions between a system and actor |
| **Actor** | User or software system that receives value from a user case |
| **Developer** | The person or organization developing the system |
| **Agent** | This is the person who will review tickets and attempt to address the issues presented. They must have a large amount of technical knowledge, as well as the ability to solve issues in a broad variety of areas. They will access the admin side web portal, where they will be able to view tickets |
| **Ticket** | A created description of a problem to be sent from User to Agent |
| **Portal A** | The interfaces used by user to submit tickets. |
| **Portal B** | The interfaces used by agents to resolve tickets. |