2020

TicketQ

SYSTEM DESIGN DOCUMENT

SHIRLEY, JEFFREY

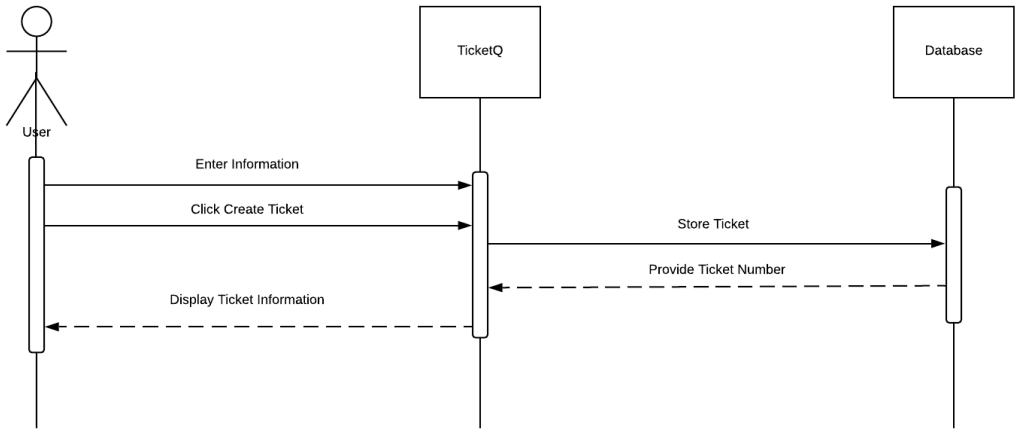
Table of Contents

Part I

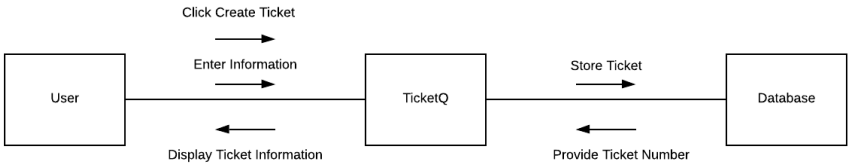
1.1 Interaction Designs

Use Case 1: User receives a complaint and creates a ticket.

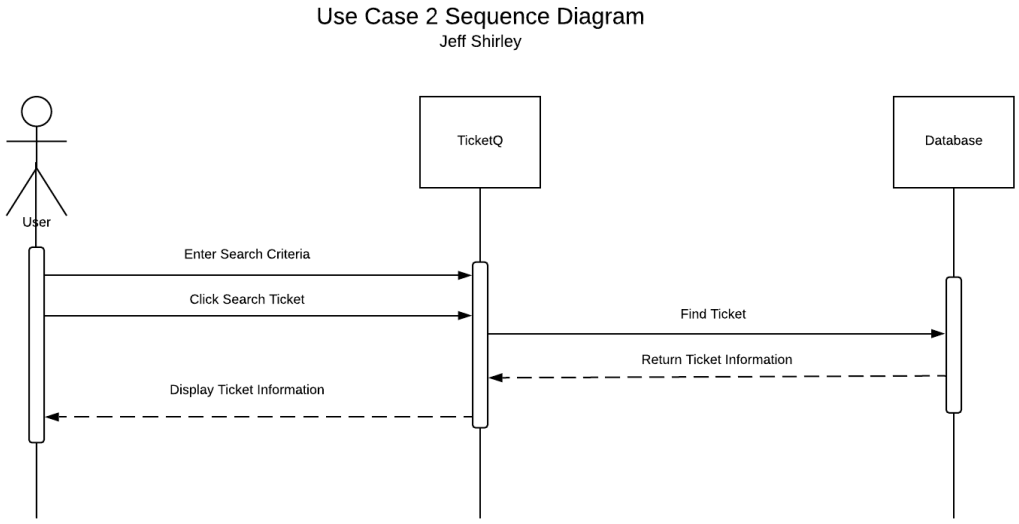
Use Case 1 Sequence Diagram
Jeff Shirley

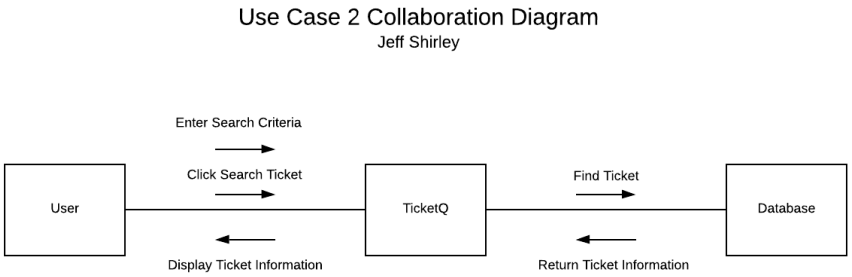


Use Case 1 Collaboration Diagram
Jeff Shirley

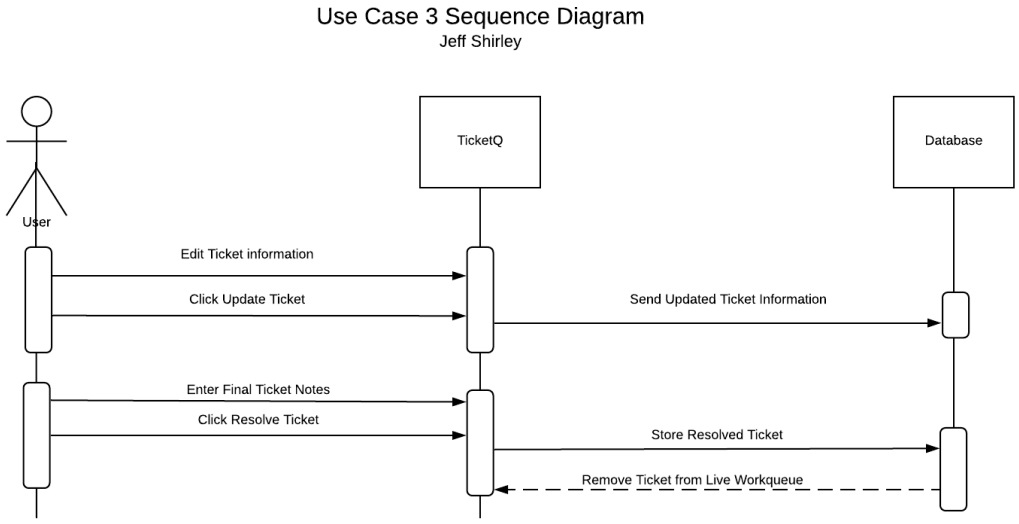


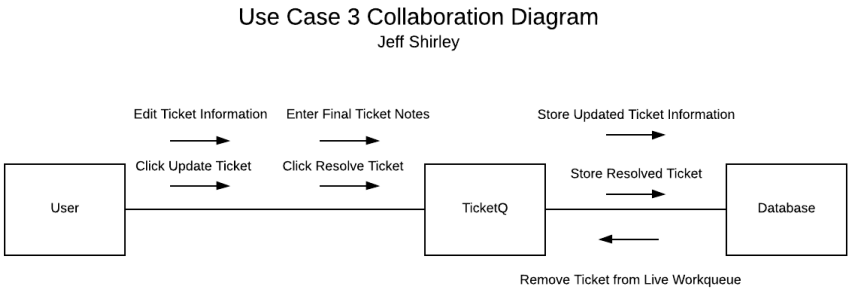
Use Case 2: User searches for a ticket and displays information.





Use Case 3: User edits/resolves ticket information.





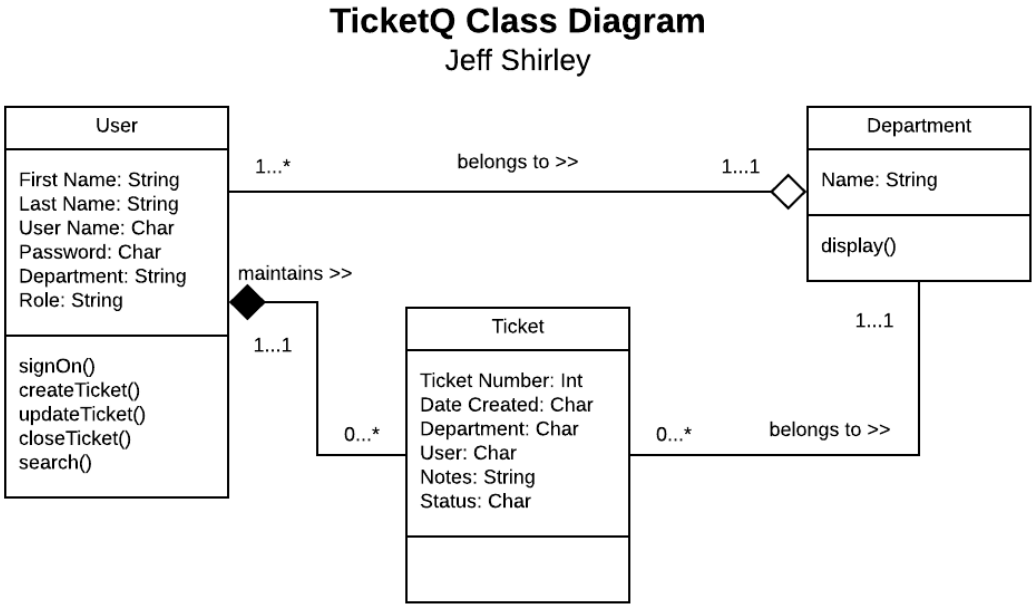
1.2 Design Principals

The design principals I decided to focus on were geared towards making these processes efficient and simple. I followed YAGNI (You aren't gonna need it) specifically on Use Case 3 and built out the process while walking through the steps of editing and resolving tickets. I tried to focus on MVP (Minimum viable product) and POLA (Principle of least astonishment) to keep everything stream linned and simple. I wanted to make sure everything does exactly what it says it will do.

Part II

2.1 Class Diagram and Interface Specification

2.1.1 Class Diagram



2.1.2 Data Types and Operation Signatures

We have narrowed down to 3 classes. First the User will have the following attributes: First Name, Last Name, User Name, Password, Department, and Role. First and Last Name will be strings consisting of the Users actual name. User Name and Password will be characters due to the User being able to create these themselves so they can contain letters and numbers. Lastly, Department and Role will be strings. Role will be a simple choice between Clerk and Manager. The User will be able to preform many operations such as signOn(), createTicket(), updateTicket(), closeTicket(), and search().

Next the Ticket will have the follwing attributes: Ticket Number, Date Created, Department, User, Notes, and Status. Ticket Number will be an integer provide by the system. Date Created will be a character field in the following format, 00/00/0000. Department will be a string. User will be a character field that copies the User Name from the User who creates the Ticket. Notes will be a string field accpeting large chunks of information. Status will be a character field with either a Y for open or N for closed. The Ticket will not perform any operations itself, it will just hold information and be manipluated by the User.

Lastly the Department will have only one attribute of Name. Name will be a string. The Department will be responsible for the operation display() which will display any open tickets assigned to that department in the workqueue.

2.1.3 Traceability Matrix

Req't	PW	UC1	UC2	UC3
Req1	8	x		
Req2	4	x		
Req3	6		x	x
Req4	1			
Req5	7		x	
Req6	3			x
Req7	8	x		
Req8	5	x		
Req9	6		x	x
Req10	2			
Req11	3		x	x
Total PW		25	22	18

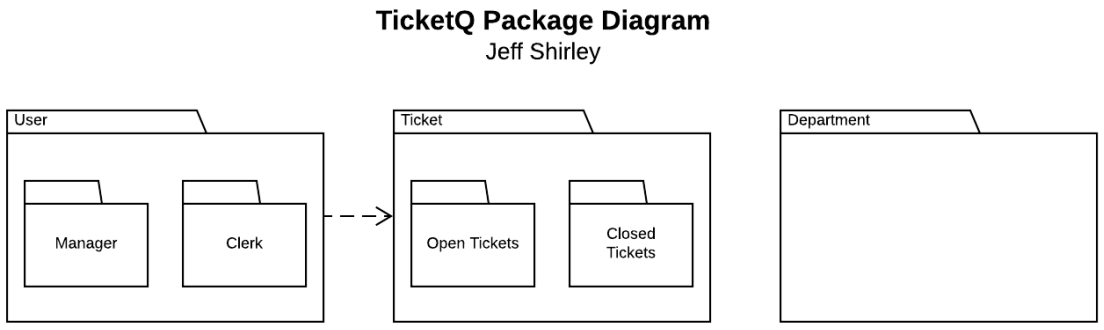
This chart displays the priority weights of the three main Use Cases presented eariler in this document.

3.1 System Architecture and System Design

3.1.1 Architecture Styles

The architecure style of TicketQ is very much an event-driven design. Nothing happens without the user making a decision. The user will prompt the system to create a ticket, search for a ticket, edit a ticket, and close tickets. It will even be users who generate user names and passwords for themselves or others.

3.1.2 Identifying Subsystems



3.1.3 Persistent Data Storage

This system will use a relational database to store information. All three main classes, User, Ticket, and Department, will store information on the database. User will search Tickets and manipulate the information. The relational database will be created using SQL.

3.1.4 Global Flow Control

TicketQ is an event-driven system. The user will determine how and when to initiate and manipulate tickets. Nothing will happen in this system without a users input. The user will sign

on the system using a self created user name and password. The user will create, search and manipulte tickets. The only thing the system does without user input is store information.

3.1.5 Hardware Requirements

The basic hardware requirements for the system consit of a screen display capable of displaying grayscale, keyboard, and a mouse. The resolution for the basic app will be 800x480 and will not have any color components. Users will use keyboard and mouse to enter data and click buttons. The user will need 1GB of free disk space for storage of the system. No other requirements will be needed for the system.

Part III

4.1 Data Structures

The data structure used in this project is an Array List. The array list allows us to be flexible in the fact that we can keep creating objects, tickets, to add to it. It also allows us to define the tickets and store the full array of information in each ticket.

5.1 Testing

Our system is fairly easy to test. When creating a ticket the system provides the ticket number and date. This leaves the remaining inputs for the user. Fortunately, these inputs are text fields and can not have invalid entries. So for unit testing we just need to make sure that our main functions proivde the automatic data for the tickets and accept user input.

When testing the system we just need to verify that the array list is holding our tickets. From there we can add other functionality like searching and editing tickets. We will also verify that our users can log in. We will also want to ensure that our system can display all open tickets.

6.1 Project Management and Plan of Work

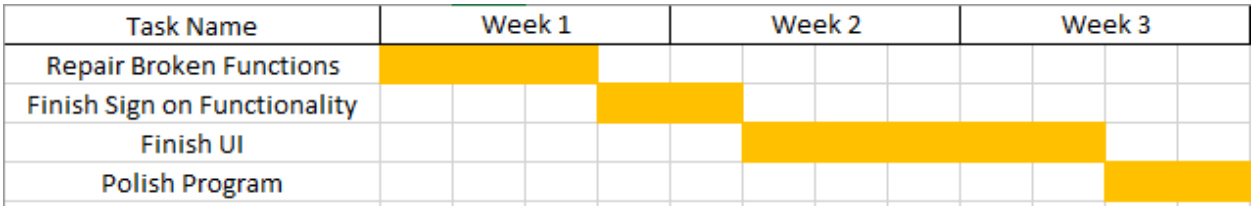
6.1.1 Project Coordination and Progress Report

Use Case 1: Create tickets has been created and is completely funtional.

Use Case 2: Search for ticket and display information. Not able to search for specific ticket but able to display all tickets

Use Case 3: Edit tickets. Not functioning, work in progress.

6.1.2 Plan of Work



References

https://www.lucidchart.com/

Textbook and Lectures