NYU Tandon School of Engineering April 8, 2020
Computer Science and Engineering
CS6083, Spring 2020

Project #1 (due 4/26)

The goal of our course project is to build a web-based issue tracking system similar to *Jira* -- but kind of different and much simpler, of course. Issue tracking systems are tools that are commonly used for collaborative software and other product development. It allows members to create projects, report issues/bugs for projects, assign the issue to certain people for fixing, and change the status of issues in the workflow. If you are not familiar with issue trackers or bug trackers, you may visit ***Minecraft Jira***, ***Google Issue Tracker***, ***Python Bug Tracker***, and ***Github Issues***, for a quick look.

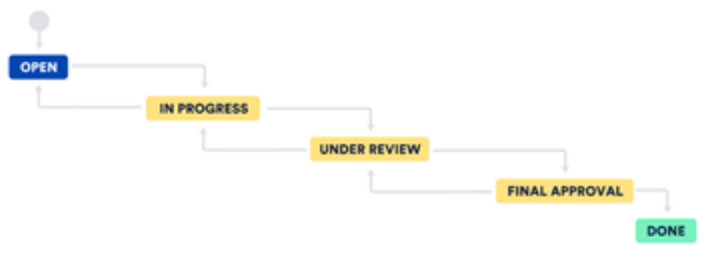
Problem Outline: We first give a few more details about the envisioned system. Users can sign up for the system by providing an email address, and can choose their username, display name, and password. Users can create projects. When creating a project, the user should provide a suitable name, maybe with a short description. Each project should have one or a few project leads, who will be responsible for the project. The user who has created the project will then automatically become the first lead of the created project, and existing leads can add other users as leads of the project.

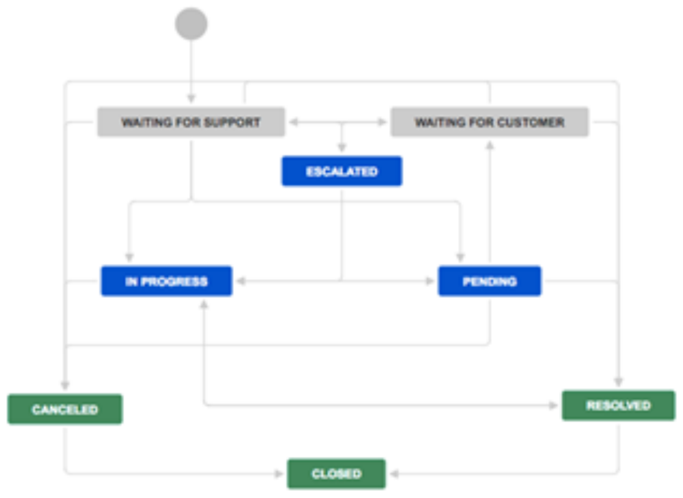
An issue is a kind of "task" - once someone has found a bug, he/she can report it as an issue, which can be assigned to suitable members of the team for processing/fixing. Each issue should belong to a certain project, and each issue should have a title and description. The creator of the issue is called the "reporter". To simplify this project, the issue creator will automatically become the only reporter of this issue; we assume that any user can report issues to any project (somewhat similar to *Github Issues*).

Project leads can then assign an issue in his/her project to a couple of users, who will become the "assignees" of this issue, and who are in charge of addressing and if necessary fixing the issue. Once an issue is created, it should be set to a default status (usually "OPEN"). Assignees and project leads can change the current status of an issue, based on the project issue workflow (as introduced in the following paragraphs). The system should keep track of the issue's creation and update times, and in general the history of status changes.

Once an issue has been reported, it usually goes through several steps to be processed. It is possible that it enters a "step loop" until finally finished. The life cycle of an issue is called "workflow", which is basically a finite-state machine. A current step (or state) in the workflow is called "status", and if we can move from one status to another status, we say that there exists a "transition" between them. In general, a workflow contains several statuses, usually starting in one special status (sometimes called "OPEN") and ending in another special status (often called "CLOSED"). Each status may have one or more possible successor statuses, as described by a directed graph. When the project lead or assignee changes the status, he/she can only change it to one of the successors statuses of the current status.

Different projects can have different workflows, but all issues in the same project follow the same workflow. Project leads can customize the workflow schema for their project, including what statuses exist and what the possible transitions are. This workflow graph needs to be stored in your database. The following two graphs are examples of workflow graphs:





Project Guidelines: In this first part of the course project, you should focus on designing a suitable relational schema that can be used to store data and perform queries for this system - including information such as users, projects, project administrators, issues, issue reporters and assignees, the history of status changes for an issue, and allowable workflow status transitions for a project. In the second part of the project, you will need to build a web-accessible front-end interface, with a back-end server interacting with your database, which should implement the functions mentioned and allow users to access the service through a web browser.

You should use your own database system on your laptop or an internet-accessible server. Use a system that supports full text search operators similar to "contains". Most of you will probably use mySQL, SQL Server, Postgres, or Oracle - if you want to use another system, please ask the instructor for permission.

Please note that you need to implement user system and content access control inside your web application logic, which means that there should not be a separate DBMS account for each user! Instead, the web application itself will log into the database using an account created for it. Thus, the system you implement can see all the content, but has to make sure at the application level that each logged-in user is identified through the use of cookies in the second part of the project, and then restrict accesses appropriately. Make sure to always store timestamps for any user actions, such as creating a project, reporting an issue, assigning an issue to other members, or changes in the status of a project.

Both parts of the project may be done individually or in teams of two students. You will receive an announcement from the TAs asking you to sign up as a group or an individual. The second part of the project will be due around or after May 10. The second project builds on top of this one, so you cannot skip this project.

Project Steps: Following is a list of steps for this part of the project. Note that in this first part, you will only deal with the database side of this project, and a suitable web interface will be designed and implemented in the second part. However, you should already envision and plan the interface that you intend to implement.

(a) Design, justify, and create an appropriate relational schema for the above scenario. Make sure your schema avoids redundancies. Show an E-R diagram of your design, and a translation into relational format. Identify keys and foreign key constraints. Note that you may have to revisit your design if it turns out later that the design is not suitable.

(b) Use a database system to create the database schema, together with key, foreign key, and other constraints.

(c) Write SQL queries (or sequences of SQL queries or scripting language statements) for the following tasks. You may use suitable placeholder values in the statements.

(1) Create a new user account, together with email, password, username, and display name.
(2) Create an issue for a project with title and description, and initialize the status of this issue.
(3) For a current user and a certain issue, first check if this user is authorized to assign it to other users (i.e., is a lead); then write a query to add an assignee.
(4) List all possible next statuses of a certain issue, based on its current status
(5) Show the status change history of a certain issue, sorted by change timestamps in descending order.
(6) List any issues for the project with name "Amazon Kindle" where the issue title contains the term "screen", user "Jeff Bezos" is one of the assignees, and the status of the issue is "OPEN".

(d) Populate your database with some sample data, and test the queries you have written in part (c). Make sure to input interesting and meaningful data and to test a number of cases. Limit yourself to a few users and a few messages and threads each, but make sure there is enough data to generate interesting test cases. It is suggested that you design your test data very carefully. Draw and submit a little picture of your tables that fits on one or two pages and that illustrates your test data! (Do not submit a long list of insert statements, but show the resulting tables.) Print out and submit your testing.

(e) Document and log your design and testing appropriately. Submit a properly documented description and justification of your entire design, including E-R diagrams, tables, constraints, queries, procedures, and tests on sample data, and a few pages of description. This should be a paper of say 10-12 pages with introduction, explanations, E-R and other diagrams, etc., which you will then revise and expand in the second part.