# Project Description

*CS 410/510*

## Overview

In this project, you will work in groups of 3–4 to design and prototype a database-driven web application for tracking bugs or tasks in a software development project. This project is modeled after tools such as GitHub Issues, Bugzilla, and JIRA. Your project will not be as complex as JIRA.

## Requirements

Your project must support a number of features:

- Users need to be able to create accounts and to log in with a username and password.[1] Users also need to be able to have *display names*, that are typically their full/normal name, and e-mail addresses.
- Each bug may have several pieces of information associated with it:
  - Title or Summary
  - Creation date
  - Creator
  - Details
  - Assignee (user who is working on the bug, may be null if the bug is unassigned)
  - One or more tags
  - Status (open, closed, rejected — you can consider adding additional statuses)
  - Close date (for when the bug was closed or rejected)
- Bugs can have *comments*, each of which has:
  - Date (and time)
  - Author
  - Text

  It is up to you whether comments are threaded or not.

[1]We will talk later in the class about how to build password authentication securely.

- Users can search for bugs with text strings.
- Users can browse bugs by status, tag, assignee, or creator. Lists should generally be sorted by age, with the newest (or most recently updated) bugs first.
- Users can easily browse the bugs that they created, and the bugs that are assigned to them; they should be able to see open bugs, closed bugs, or all bugs.
- Each user has a profile page that displays some information about them:
  - Their user name
  - Their display name
  - Their e-mail address
  - Their assigned bugs
- Users can add new bugs to the system, and edit the title and details of existing bugs.
- Users need to be able to *subscribe* to bugs, where they will be notified of changes. This has several implications:
  - When a user creates, is assigned, or comments on a bug, they are automatically subscribed to it.
  - There should be a link or button to unsubscribe from a bug.
  - Users can view all bugs to which they are subscribed (filterable by status).
  - Users have a *news feed*, that shows activity on they bugs they are subscribed to. The following activities, at least, should be reported:
    - Bug created
    - Bug comment added
    - Bug assigned

In addition to the above, you must design and provide at least two additional features for users. Here are few ideas for potential features:

- Milestones, where bugs can be assigned to a milestone and the user can view the bugs for that milestone, along a milestone description and information on how many are open, closed, etc. to estimate the milestone's progress.
- A *change log*, that keeps track of all changes to bugs and who made the change. So rather than just changing the title when a user changes the title, or adding a tag when the user adds a tag, it also records a log of those change records so that when users view the bug, a list of changes is available.
- *Mentions*, where one user can mention a user (typically using @username syntax) in another bug comment; doing so should make the comment show up in the user's news feed.
- A permissions system.

- Components, where a bug can be associated with a component. Users can be subscribed to components, and are automatically subscribed to all bugs associated with their subscribed components.
- Subscribing to tags, which would work like components.
- A time or effort tracking system.

### User Schemas

Think of your project as having at least two user schemas:

- The *developer* view, where a user cares about the bugs they are assigned, and the discussions about those bugs. They probably also care about bugs they've reported, or bugs associated with components they work on.
- The *user* view, where a user cares about a few bugs they have reported, but not the rest of the project.

Adding a new user schema is also a good starting point for an ambitious extra feature. For example, you could have a Manager schema, and build out some views and forms from that schema, that allows a manager to see the bug activity of the people who report to them.

### Technology

I will be demonstrating web/DB connectivity in Java using the Spark framework. This is a lightweight web framework, similar in many ways to Bottle for Python or Sinatra for Ruby. I will also be providing infrastructure templates to help me run Java-based applications.

If your team would prefer to work in another language such as Ruby or Python, that is acceptable, subject to a few conditions:

- Don’t use an object-relational mapper. We may discuss ORMs later in the semester, but for the project they are out-of-scope. Write SQL directly, or perhaps using a very lightweight shim such as SQLAlchemy’s SQL-generation capabilities.
- Prefer a bare-bones framework such as Flask, Bottle, or Sinatra over a heavyweight framework like Django or Rails.
- You are responsible for creating a Dockerfile to go with your submission that, when built, will create an image for a Docker container that will run your app and expose it on a TCP port. I will provide a Dockerfile for Java+Spark applications.

## Deliverables

The following table summarizes the project deliverables; see below for more details on each deliverable.

| Deliverable | Grade Pct | Due |
|---|---|---|
| **Initial Data Model** | 5% | **Oct. 22** |
| **Peer Review** | 10% | **Oct. 22/24** |
| **Revised Data Model** | 25% | **Nov. 1** |
| **Development Midterm Report** | 10% | **Nov. 12** |
| **Final Presentation** | 20% | **Dec. 3** |
| **Final Code and Report** | 30% | **Dec. 7** |

All deadlines are at 6:30 PM.

## Initial Data Model

By Oct. 22, you must submit a draft data model. This draft must consist of:

- Entity-relationship diagram of your logical data model.
- Chicken-feet diagram of your relational data model
- Primary keys for each table
- 3 example queries to retrieve data for display to the user. These queries should be non-trivial, involving multiple tables. At least one must have an aggregate function.
- A brief description of your custom features.

On Oct. 22, each group will give a short presentation to the class about their data model to get feedback. Each group will also be responsible for providing written feedback to 2 other groups (to be assigned).

## Peer Review

By Oct. 24, you must submit a written version of your peer feedback. Send your feedback to each group using the TRACS e-mail tool by Oct. 24 at midnight, and submit a compiled document for grading consisting of:

- The feedback you sent to each group
- A short (1—3 paragraphs) description of one thing you plan to improve in your data model as a result of seeing other groups' ideas.

## Revised Data Model

Submit a revised data model by Nov. 1, consisting of the same materials you submitted in the Initial Data Model, updated and revised based on the feedback you received from your peers and me.

## Midterm Report

On Nov. 12, submit a midterm report on your development status. This report must include:

- Updated data model diagram, if you have changed anything since the Revised Model.
- A list of indexes that you are creating.
- The SQL DDL (`CREATE TABLE`, etc.) statements to set up your database in PostgreSQL. This should include all data types, foreign keys, etc.
- A list of the different pages that your application will have (e.g. bug details, create bug, user profile). Each page needs:
  - An example URL
  - A 1—3 sentence description
  - Development status (completed, in testing, not started)
- A description of your custom feature

Submit your report as 2 files: a PDF containing the text and diagrams, and an SQL script file with the DDL.

## Final Presentation

On Dec. 3, each group will give a short presentation to the rest of the class demonstrating their application. This presentation should contain:

- A demonstration of the app's key features, including your custom feature
- An explanation of 2 of the interesting SQL queries that you used

## Final Code and Report

By Dec. 7, submit your final project. This should be a zip archive containing:

- The final data model diagram.
- Your source code, including all Python files, SQL scripts, HTML templates, and images (everything that I will need to run the project inside the Vagrant image)
- A list of the different pages. For each page, provide:

    - An example URL
    - A 1–3 sentence description
    - The SQL queries that are used to display that page

  This list should be a single PDF file.

- Instructions to run it (should just be ‘python bugserver.py’

## Group Contribution Report

On Dec. 7 (the same day as the final project is due), each group member must individual submit a short report (can be less than a page) assessing their own contribution and that of their group members. This report will not be directly graded, but may influence the grading of the group.

If your group has great dynamics and everyone pulls their weight, shows up for meetings, etc., this can even be a single sentence.