Spring 2020

Amanda Kuznecov
Net ID: anr431

CS-GY 6083: Project #2

1 Project Description

The purpose of this project is to develop a web-based issue tracking system integrated with a database management system. The system will allow users to create projects, set up workflow schemas, report project issues, assign issues to other users, and update the status of issues based on the workflow schema. The project has been divided into two major parts; the first part focuses on developing and implementing a database system, and the second part focuses on developing the web application. Both parts of the project are fully described in this paper.

2 Entity-Relationship Model

An Entity-Relationship Model was developed to describe the entities and their relationships with one another as shown in Figure 1.

Everyone who uses the system is considered a user and will have personal login credentials. Each user can have two types of roles – a 'Project-Level' role, which is mandatory and assigned upon user creation, and an 'Issue-Level' role, which is assigned if the user is involved with a specific project issue. An example of a 'Project-Level' role is a Project Lead, which is the default setting for a user who creates a project. Other types of 'Project-Level' roles include Team Members, the default role for new users. Existing Project Leads can add other users as Project Leads or add other users as Team Members. An example of an 'Issue-Level' role is a user who reported an issue – they automatically become the sole Reporter of the issue. After the issue has been created, Project Leads can assign the issue to other users who will become Assignees. Note that if a user never reports or is never assigned an issue for a specific project, they will not have an 'Issue-Level' role. Each role has a set of pre-defined permissions.

Once a project is created, a Project Lead can create a single workflow that is specified for the project, and it cannot be modified after it is set. The workflow will follow a set of statuses predetermined by the Project Lead. Workflows do not always follow a linear path, and a single status may have multiple possible subsequent statuses, as indicated by the roles *current_state* and *subsequent_state*. As updates are made to issues, the issue history is tracked as an entry with its updated status, which is then updated in the *issue* entity to reflect the most current status on the issue.

The E-R diagram defines certain constraints that the database contents must conform to.

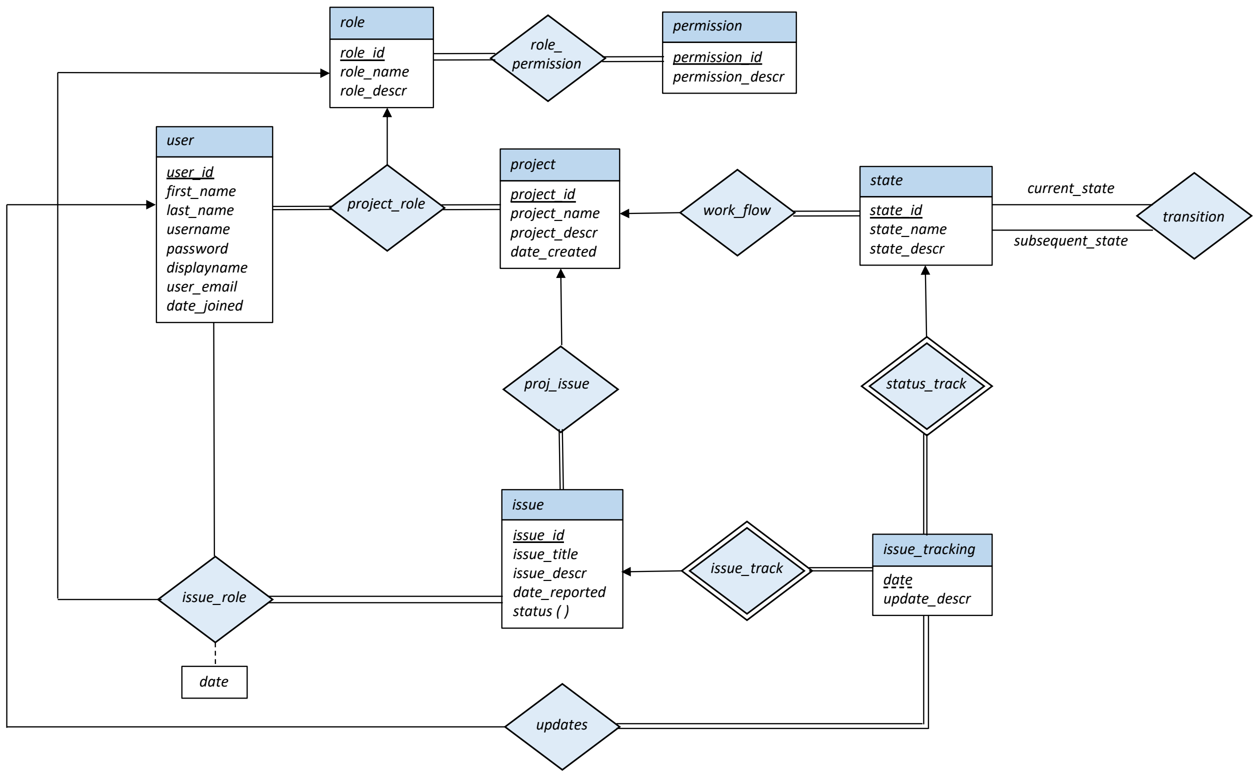


Figure 1: E-R Diagram

2.1 Relationship Sets

The following relationship sets are binary:

- *role_permission:* relating roles with permissions
- *proj_issue:* relating projects with issues
- *work_flow:* relating projects with workflow statuses
- *issue_track:* relating tracked history entries with issues
- *status_track:* relating tracked history entries with workflow statuses
- *updates:* relating tracked history entries with users who performed the update
- *transition:* relating statuses with itself on role indicators *current_state* and *subsequent_state*

The following relationship sets are ternary:

- *issue_role:* relating users with roles with issues
- *project_role:* relating users with roles with projects

These relationships are ternary because the project or issue that a user works on is directly related to the user's role on either that project or issue. Therefore, the combined relationship between all three of these entities is necessary.

2.2 Cardinality

For the binary relationship sets, cardinality is as follows:

- **Many-to-one:**
  - From relationship set *proj_issue* to entity set *project*. A project can have multiple issues, but a single issue can only belong to one project.
  - From relationship set *work_flow* to entity set *project*. A project can have many statuses (workflow items), but a single status can only be associated with one project.
  - From relationship set *issue_track* to issue. An issue can have multiple tracked history entries, but a single tracked history entry can only be associated with one issue.
  - From relationship set *status_track* to status. A status can belong to multiple tracked history entries, but a single tracked history entry can only be associated with one status.
  - From relationship set *updates* to user. A user can perform many status updates, but a single tracked history entry can only be performed by one user.

- **Many-to-many:**
  - For relationship set *role_permission.* A permission can belong to many roles, and a role can have many permissions.
  - For relationship set *transition.* The current state can be associated with many possible subsequent states, and a subsequent state can be associated with many possible current states.

For the ternary relationship sets, cardinality is as follows:

- **Many-to-one:**
  - For relationship set *issue_role* to entity set role. A user can have at most one role on a specific issue, however a user can report or be assigned to many issues, and an issue can be assigned to many users.
  - For relationship set *project_role* to entity set *role.* A user can have at most one role on a specific project, however a user can work on many projects, and a project can be worked on by many users.

2.3 Weak Entity Sets:

The entity set *issue_tracking* is uniquely identified by *issue_id* from *issue*, *state_id* from *state*, *user_id* from user, and the *date* when the tracking update was made. Therefore, *issue_tracking* is reliant on the identifying entity sets *issue*, *state*, and user, and has a discriminator *date*, which records the date and time of the status update.

2.4 Participation Constraints:

For binary relationships, participation constraints are as follows:

- **Total Participation:**
  - Participation of *role* in the relationship set *role_permission*. Every role entity is related to at least one permission.
  - Participation of *permission* in the relationship set *role_permission*. Every permission is related to at least one role.
  - Participation of *issue* in the relationship set *proj_issue.* Every issue is associated with at least one project.

- Participation of *project* in the relationship set *work_flow.* Every project is associated with at least one status.
- Participation of *state* in the relationship set *work_flow.* Every status is associated with at least one project.
- Participation of *current_state* and *subsequent_state* in relationship set *transition.* Each of these states are associated with at least one status.
- Participation of *issue_tracking* in relationship set *issue_track*. Every tracked history entry is associated with at least one issue. Also, because *issue_tracking* is a weak entity set, it is required to have total participation in the identifying entity set, *issue track*, that it is dependent on.
- Participation of *issue_tracking* in relationship set *status_track.* Every tracked history entry is associated with at least one status. Also, because *issue_tracking* is a weak entity set, it is required to have total participation in the identifying entity set, *state,* that it is dependent on.
- Participation of *issue_tracking* in relationship set *updates*. Every tracked history entry is associated with at least one user. Also, because *issue_tracking* is a weak entity set, it is required to have total participation in the identifying entity set, *updates*, that it is dependent on.

- **Partial Participation:**
  - Participation of *project* in the relationship set *proj_issue*. Not every project is associated with an issue. For instance, a project may have no issues – this is unlikely but is possible when the project is first created or in its early stages.
  - Participation of *issue* in relationship set *issue_track*. Not every issue is associated with a tracked history of an issue. For instance, the issue may have no status updates and therefore it does not exist as a tracked history entry.
  - Participation of *state* in relationship set *status_track*. Not every status is associated with a tracked history of an issue. For instance, some issues may only have a couple status updates, and depending on the workflow, they may entirely bypass some of the status changes.

For ternary relationships, participation constraints are as follows:

- **Total Participation:**
  - Participation of *user* in the relationship set *project_role*. Every user is related to at least one project, and every user is related to at least one role.
  - Participation of *project* in the relationship set *project_role*. Every project is related to at least one user with a specific role.
  - Participation of *issue* in the relationship set *issue_role.* Every issue is related to at least one user with a specific role.

- **Partial Participation:**
  - Participation of *user* in the relationship set *issue_role*. Every user is not required to be involved in an issue. The only users involved in an issue are Reporters and Assignees, which may not cover all users. Therefore, users are not required to have an 'Issue-Level' role.

- Participation of *role* in the relationship set *issue_role.* Every type of role does not need to be assigned to users of a specific issue. Reporters and Assignees are the only 'Issue-Level' roles, and the only types of roles assigned in this relationship.
- Participation of *role* in the relationship set *project_role.* Every type of role does not need to be assigned to users of a specific project. For example, there can be projects with only Project Leads.

3 Relational Schema

The E-R model can be reduced to a relational schema *issue_tracker*, which is outlined below:

user (user id, first_name, last_name, username, password, display_name, user_email, date_joined)
Primary Key: user_id

role (role_id, role_name, role_descr)
Primary Key: role_id

project (project_id, project_name, project_descr, date_created)
Primary Key: project_id

issue (issue_id, issue_title, issue_descr, project_id, date_reported, status)
Primary Key: issue_id
Foreign Keys: project_id (references project_id in *project*)

permission (permission_id, permission_descr)
Primary Key: permission_id

state (state_id, state_name, state_descr, project_id)
Primary Key: status_id
Foreign Key: project_id (references project_id in *project*)

role_permission (role_id, permission_id)
Primary Keys: role_id, permission_id
Foreign Keys: role_id (references role_id in *role*), permission_id (references permission_id in *permission*)

project_role (project_id, user_id, role_id)
Primary Keys: project_id, user_id
Foreign Keys: project_id (references project_id in *project*), user_id (references user_id in *user*), role_id (references role_id in r*ole*)

issue_role (issue_id, user_id, role_id)
Primary Keys: issue_id, user_id
Foreign Keys: issue_id (references issue_id in *issue*), user_id (references user_id in *user*), role_id (references role_id in *role*)

issue_tracking (issue_id, state_id, updated_by, date, update_descr)
Primary Keys: issue_id, state_id, updated_by, date

Foreign Keys: issue_id (references issue_id in i*ssue*), state (references state_id in *state*), updated_by (references user_id in *user*)

transition (current_state, subsequent_state)
Primary Keys: current_state, subsequent_state
Foreign Keys: current_state and subsequent_state (reference state_id in *state*)

The relation *project_role* requires primary keys *project_id*, *user_id* in order to create a unique identifying key in the relation. Since a user can only have one role on a specific project, *role_id* is not required as an identifier. This is similar for the relation *issue_role,* which requires primary keys *issue_id, user_id*. A user can only have one role on a specific issue, and therefore *role_id* is not required as an identifier.

For the *issue_tracking* relation, the primary key is comprised of all four attributes because it is possible for a certain issue to have a specific *state_id* more than once in history. For example, the issue's status may loop back to a previous status in the workflow depending on the project's workflow. This is why it is necessary to include the date as an identifier.

4 Database System

The relational schema described was created as a database system using mySQL. This section will discuss default values, implemented constraints, testing data, and testing performed. It will also address the tasks specified in the project outline.

4.1 Database Relations and Constraints

Attributes were created for each relation as outlined in the previous section. Data types were declared for each attribute as well as other special constraints such as not null, unique, and auto-increment. Default values were also assigned in some cases. All columns in the database were set to Not Null, as we do not wish to store any empty or missing information. Primary keys in the *user, role, project, issue, permission,* and *status* relations were set to auto-increment so that once a new entry is made for any of these entities, the identifier can automatically be set to the next number without needing to be specified.

For some new row insertions, the date is desired to be recorded to capture key events in the database. In these cases, the date and time of insertion is automatically entered as the default value. This was applied to the *user* relation, which records when a new user joins, the *project* relation, which records when a new project is created, and the *issue* relation, which records when a new issue is reported. Dates are also automatically recorded in the *issue_role* relation when a user reports an issue, or is assigned to an issue. In the *issue_tracking* relation, dates are recorded when the status of an issue is updated.

In the *user* relation, attributes *username* and *display*name were specified to be unique values. The web application will require users to have unique login information so that they can be properly identified when entering the system. Therefore, users cannot share the same username, otherwise the system would not be able to differentiate between users upon login. Unique display names are also required so that users in the system can differentiate between one another. If users shared display names, they would not be able to tell the difference between users if they saw the same display name attached multiple times to a single project or issue.

In the *role* relation, the *role_name* is required to be unique because if there were multiple roles with the same name and different permissions, this would cause confusion.

In the *issue* relation, *status* is a special value as it does not actually need to be specified upon row insertion. Once a new issue is created, it is given a default initial status depending on its project-specific workflow. Since mySQL does not allow function expressions for default values, a trigger was created before insertion of new rows into the *issue* relation. The trigger looks up the *project_id* of the new issue within the status entity to find the associated workflow statuses. Then, the new issue's status is set to the *state_name* with the lowest ranked *state_id* for that specific project.

It is necessary to check whether a user being assigned to an issue actually works on the project associated with the issue. In order to do so, a trigger was created before inserting a new row in the *issue_role* relation. The trigger looks up the *user_id* in the *project_role* relation to check whether the user has a 'Project-Level' role on that specific project. If they do not, the insertion is rejected and an error message states that the user does not work on that project.

After a new row has been inserted into the *issue_tracking* relation to reflect the status update of an issue, the current status must be updated in the *issue* relation. This was done by creating a trigger after insertion into the *issue_tracking* relation. The new *state_id* is looked up in the *state* relation to retrieve the *state_name*. In the *issue* relation, *state* is then updated with this new value.

4.2 Database Testing

Test data was inserted into the database to perform checks and ensure proper constraints were in place to adhere with the project outline. For all relations with auto-increment values and auto-generated dates, tests were performed upon insertion of new rows to check that these functions were running properly.

In the *issue* relation, the initial status setting was tested by inserting a new issue for a project and then checking to see if the trigger correctly inserted the initial status defined by the project's workflow. The check was performed by manually looking up the new issue's corresponding *project_id* in the *state* relation, and finding the lowest ranking *state_name*. To test if the status was properly updated in the *issue* relation after a status update, entries were created in the *issue_tracking* relation. After each new status update, a check was performed on the *issue* relation to see whether the trigger properly updated the *state* based on the latest information from the *issue_tracking* relation.

Some users were assigned to work on all projects, while some only assigned to a couple. For testing purposes, some users hold a variety of roles on different projects, while some have the same role on all projects they are involved in. Testing was performed on the *project_role* relation to ensure that a single user in a project could only be assigned one 'Project-Level' role. This was done by attempting to insert a user who already had a role on a specific project with another role on the same project.

Testing was performed on the *issue_role* relation to ensure that users were not assigned to issues on projects they were not working on. This was done by attempting to insert the *user_id* of someone not on a specific project team in conjunction with an issue pertaining to that project.

Testing was performed on the *issue_tracking* relation to ensure that a status belonging to a different project's workflow could not be added to an issue's status history. This was done by adding a new entry to *issue_tracking*, and attempting to enter a status from a different project's workflow. Testing was also performed to check whether the new status was an allowable transition from the current status. This was done by attempting to enter subsequent states that did not exist in the *transition* relation.

5 Web-Based User Interface

Using PHP and HTML, a web-based user interface was created by using a local web server, database and browser. The program was connected to the backend database and used prepared statements to return results as a web page.

5.1 Web Application Layout

The general layout of the web application is shown in
Figure **2**. This section describes each web page in detail. All web forms use the POST method to submit user entries to a PHP page for processing and validity checks.

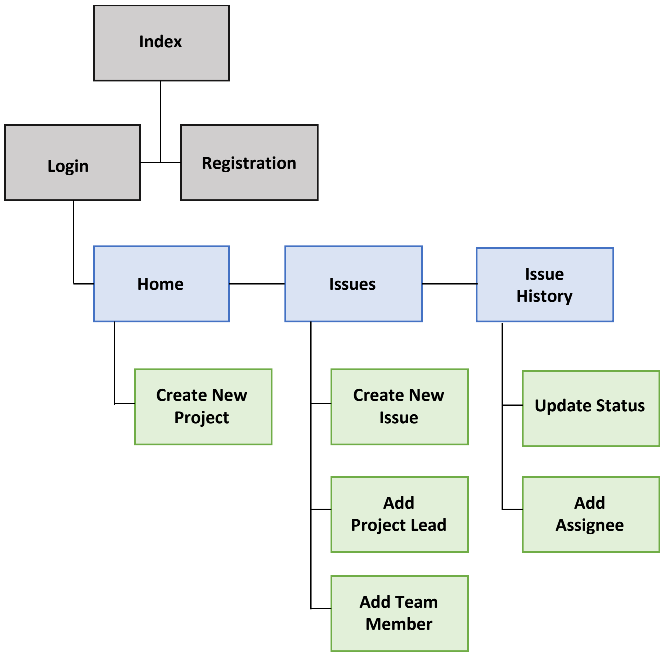


Figure 2: Web Application Layout

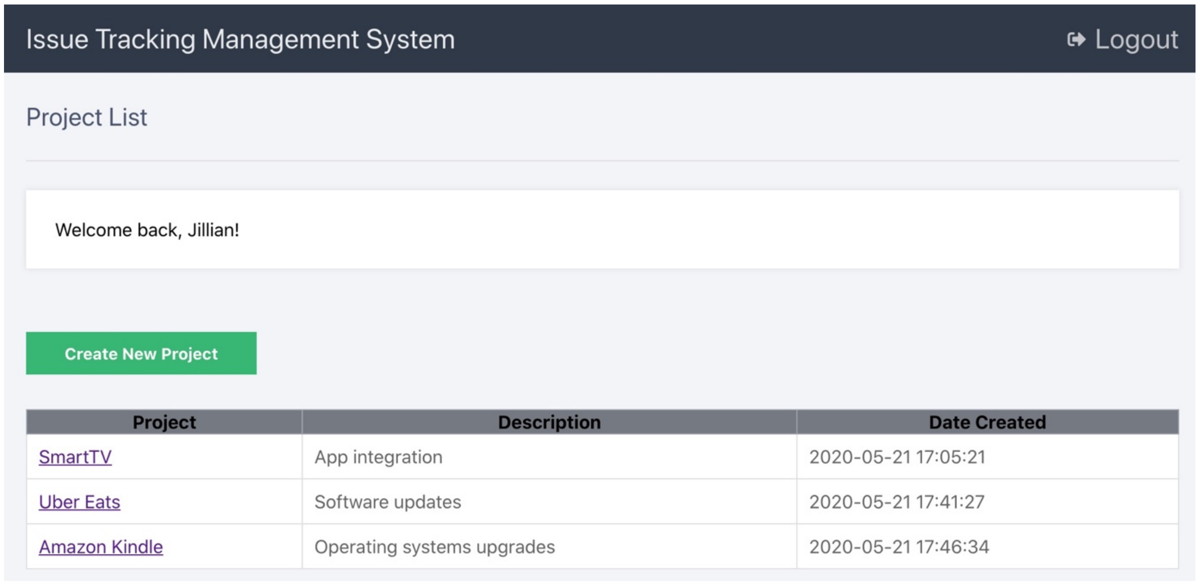
5.1.1 Index Page

Users are greeted by an Index page where they have the option of logging in if they already have a username, or they can register. During the registration process, a form of input text boxes is presented, and users are asked to enter their first name, last name, username, password, display name, and email. All input boxes are required, so if a user tries to submit the form before filling it out in its entirety, they will be prompted to fill out the missing field. Names are allowed to consist of any letters, and usernames and passwords are allowed to consist of any combination of letters or numbers. Passwords are not directly stored within the database, but rather the ‘password_hash’ function is used to encrypt the user’s password. Using the PHP method ‘FILTER_VALIDATE_EMAIL’, email addresses are validated before accepting the new user. A check is then performed to see if an account with the same username already exists. If the username already exists, the user is prompted to choose a new one. If the username does not already exist, the new user information is inserted into the *user* relation, and the user is redirected back to the Login Page where they may login with their new username and password.

Both fields are required for login. Once a user submits their login information, the username is looked up within the database to check its corresponding *user_id*. If the query does not return any results, then no user is found, and the user is prompted to re-enter their information. However, if the user does exist, their entered password is verified using the ‘password_verify’ function. A session is then created for the logged in user to store values that can be accessed across all pages while they are logged in. Variables stored in the session at this point include $_SESSION[‘loggedin’], which is a boolean set to TRUE, as well as the *username* they logged in with, the associated *user_id*, and their *first_name*. For each page to be accessed upon login, the $_SESSION[‘loggedin’] variable is checked, and if set to True, the user is allowed to see the page, otherwise they are sent back to the login page. The session_start() function is used to resume the user’s session from the previous page. The user is then redirected to the Home Page.

5.1.2 Home Page

Upon arriving at the Home Page, the page is customized to list all projects that the user is associated with, including the project’s description and the date it was created. A snapshot of the Home Page is provided in Figure 3. Projects are listed as links that redirect to project-specific pages listing all issues associated with the particular project. Once a user clicks on a project link, its *project_id* and *project_name* are passed through to the Issues Page. The Home Page also has a button the user may click on to create a new project.



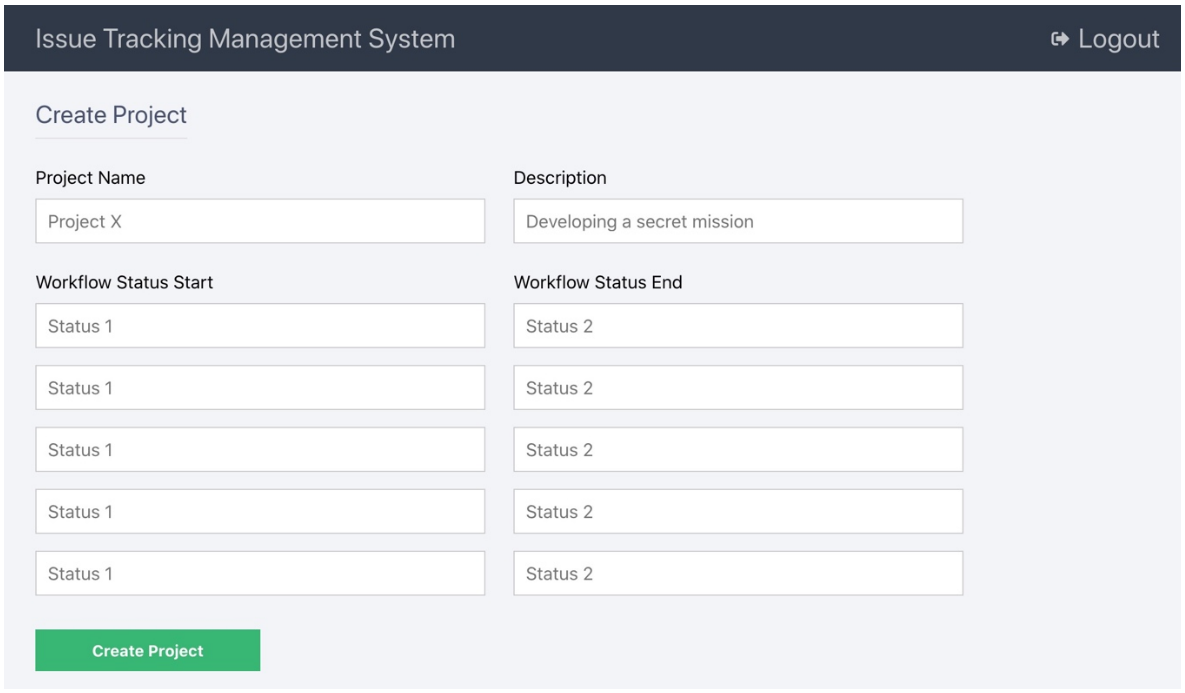
Welcome back, Jillian!

Project	Description	Date Created
SmartTV	App integration	2020-05-21 17:05:21
Uber Eats	Software updates	2020-05-21 17:41:27
Amazon Kindle	Operating systems upgrades	2020-05-21 17:46:34

Figure 3: Home Page

5.1.3 Create New Project Page

After clicking to create a new project, the user is redirected to the form shown in Figure 4. Here, the user may enter the new project's name and description as well as workflow transitions. In the workflow section, the user can enter the starting state on the left, followed by its subsequent legal transition to the right. They may continue to do so moving down the sheet until they have entered all that is needed for their particular workflow. If a state has multiple subsequent states, the state must be entered in multiple rows on the left side, followed by all its possible subsequent states on the right side. Additionally, if it is possible to go back and forth between two states, the state and subsequent state must be listed as both transitions. For example, if it is possible for In Progress → Under Review, and Under Review → Progress, these transitions must be listed on two separate rows. Not all fields are required to be filled out for workflow, but a minimum of the first row must be filled out.

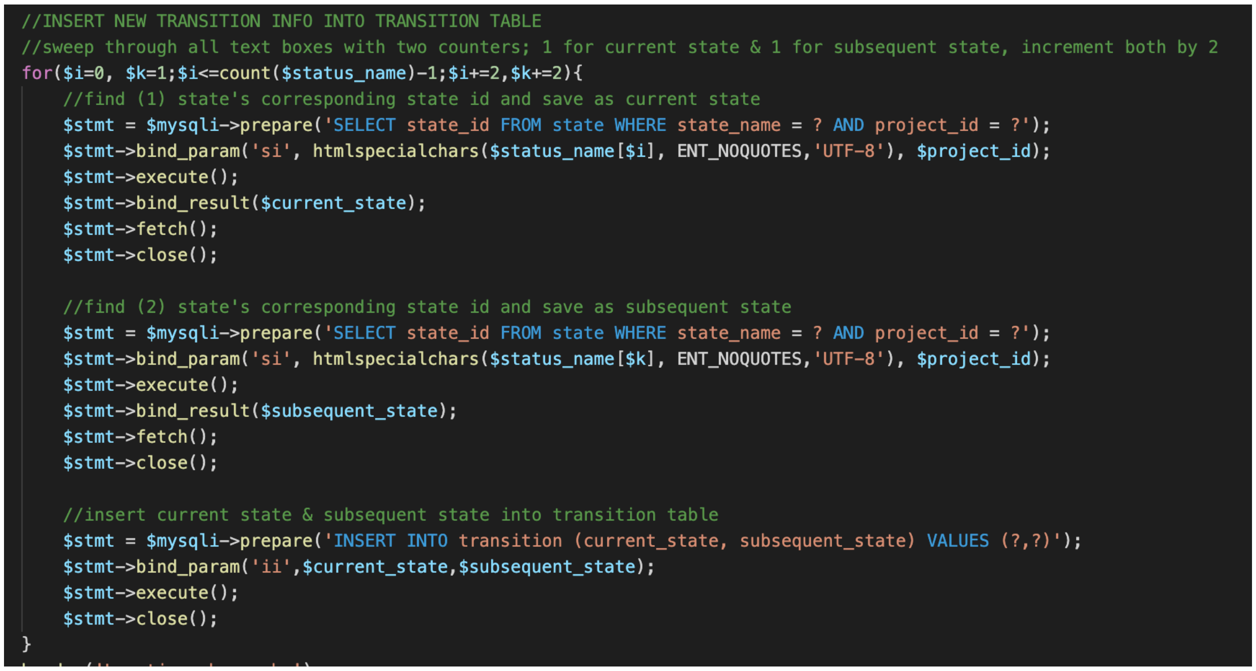


Project Name	Description
Project X	Developing a secret mission
Workflow Status Start	Workflow Status End
Status 1	Status 2
Status 1	Status 2
Status 1	Status 2
Status 1	Status 2
Status 1	Status 2

Figure 4: Create New Project Page

The new project name and description are checked for validity, and then entered into the *project* relation. The current user is then assigned as Project Lead on the project, which gets inserted into the *project_role* relation using the $_SESSION['id'] and *role_id* = 1, representing the Project Lead role. Because not all status fields were required to be filled out, but each current state must be associated with a subsequent state, there must be an even number of text boxes filled out. A check must be performed that counts the number of entries filled out and if the total number is even. Since some of the states may have been entered multiple times due to the nature of the workflow, (ie. "Reviewed" appears twice in: Open → Reviewed, Reviewed → Closed), we must check if the state has already been entered in the database system for this particular project. It is only necessary to enter each *state_name* once for a particular project in the database. All input status entries are stored in an array which is then looped through to perform this check. If the *state_name* does not already exist in the database for the project, it is entered into the *state* relation which contains a list of all possible *state_id*.

In order to enter the workflow status transitions as specified by the user, two counters were created (one for inputs on the left, and one for inputs on the right), to sweep through and find the matching *current_state* and *subsequent_state* to be entered in the *transition* relation. Both counters are then incremented by two to move to the next line of status transitions. Queries used for inserting these new transitions are shown in Figure 5. After clicking "Create Project", the user is redirected back to the Home Page.



```
//INSERT NEW TRANSITION INFO INTO TRANSITION TABLE
//sweep through all text boxes with two counters; 1 for current state & 1 for subsequent state, increment both by 2
for($i=0, $k=1;$i<=count($status_name)-1;$i+=2,$k+=2){
    //find (1) state's corresponding state id and save as current state
    $stmt = $mysqli->prepare('SELECT state_id FROM state WHERE state_name = ? AND project_id = ?');
    $stmt->bind_param('si', htmlspecialchars($status_name[$i], ENT_NOQUOTES,'UTF-8'), $project_id);
    $stmt->execute();
    $stmt->bind_result($current_state);
    $stmt->fetch();
    $stmt->close();

    //find (2) state's corresponding state id and save as subsequent state
    $stmt = $mysqli->prepare('SELECT state_id FROM state WHERE state_name = ? AND project_id = ?');
    $stmt->bind_param('si', htmlspecialchars($status_name[$k], ENT_NOQUOTES,'UTF-8'), $project_id);
    $stmt->execute();
    $stmt->bind_result($subsequent_state);
    $stmt->fetch();
    $stmt->close();

    //insert current state & subsequent state into transition table
    $stmt = $mysqli->prepare('INSERT INTO transition (current_state, subsequent_state) VALUES (?,?)');
    $stmt->bind_param('ii',$current_state,$subsequent_state);
    $stmt->execute();
    $stmt->close();
}
```

Figure 5: Queries to insert into the *transition* relation

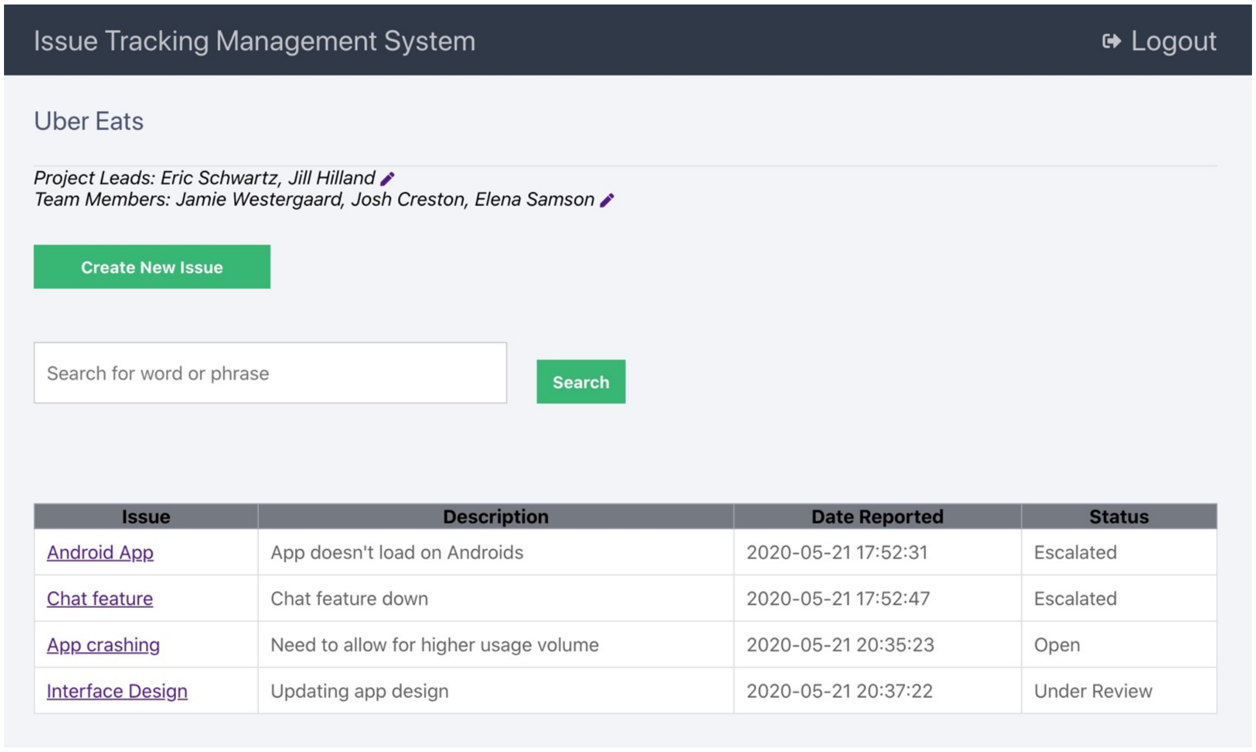
5.1.4 Issues Page

The Issues Page shows issues pertaining to the project the user clicked on from the Home Page, as shown in Figure 6. Issue title, description, date reported, and its current status are presented in the table. Issues are listed as links that redirect to issue-specific pages listing the particular issue's history.

Session variables $_SESSION['project_id'] and $_SESSION['project_name'] are updated based on the current project being viewed. These variables are used as we move through the web application, but once a different project is selected from the Home Page, they are automatically modified to reflect that. This helps when we move to a different page relating to the particular project and want to perform a query based on *project_id* or *project_name.* For example, the headings on each page are set by these session variables, and queries within the Create New Issue Page are reliant on these session variables.

A list of all Project Leads and Team Members is displayed at the top of the page for easy reference. An array of the query results was created by looping through each result, concatenating the results and then using the implode function to separate each value by a comma. A current user who is a Project Lead on the project being viewed is able to add Project Leads or Team Members by clicking on the pencil next to these lists. The pencils are hidden from users who are not Project Leads on the particular project. All users associated with the project are able to create a new issue by clicking on the Create

New Issue button. Additionally, users are given the option of filtering table rows by entering a key word or phrase that they are searching for. This was done by creating a search form and using the LIKE operator to match the user's inputs with the table results, ultimately filtering out anything that does not match. If no search word or phrase has been entered, the query to show project issues is performed as usual without filtering.



Uber Eats

Project Leads: Eric Schwartz, Jill Hilland
Team Members: Jamie Westergaard, Josh Creston, Elena Samson

Issue	Description	Date Reported	Status
Android App	App doesn't load on Androids	2020-05-21 17:52:31	Escalated
Chat feature	Chat feature down	2020-05-21 17:52:47	Escalated
App crashing	Need to allow for higher usage volume	2020-05-21 20:35:23	Open
Interface Design	Updating app design	2020-05-21 20:37:22	Under Review

Figure 6: Issues Page

5.1.5 Create New Issue Page

After clicking to create a new issue, the user is redirected to the form shown in Figure 7. Here, the user may enter the new issue's title and description, as well as select an Assignee. The dropdown list was created using a select form. The selections shown are a queried version of *displayname,* listing all users who work on the project and could potentially be assigned to the issue, excluding the logged in user who actually becomes the Reporter of the issue.

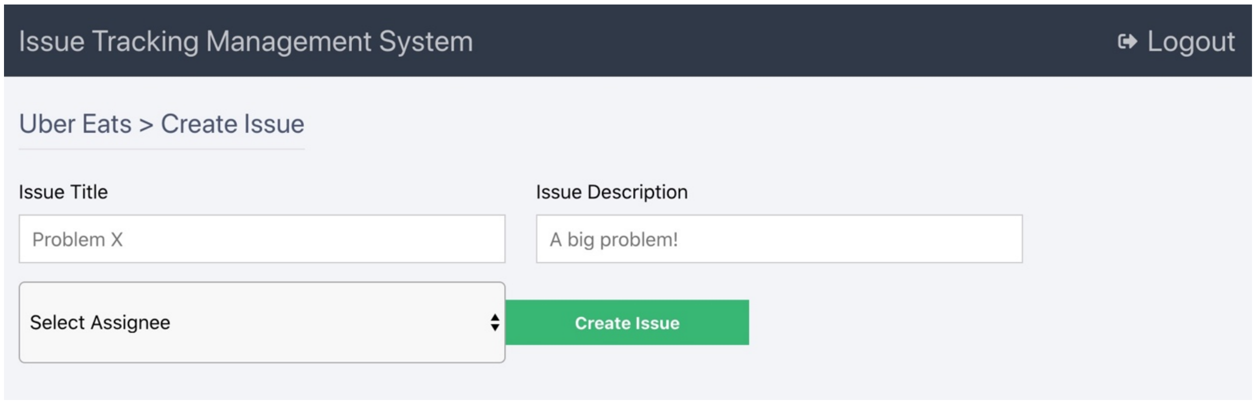
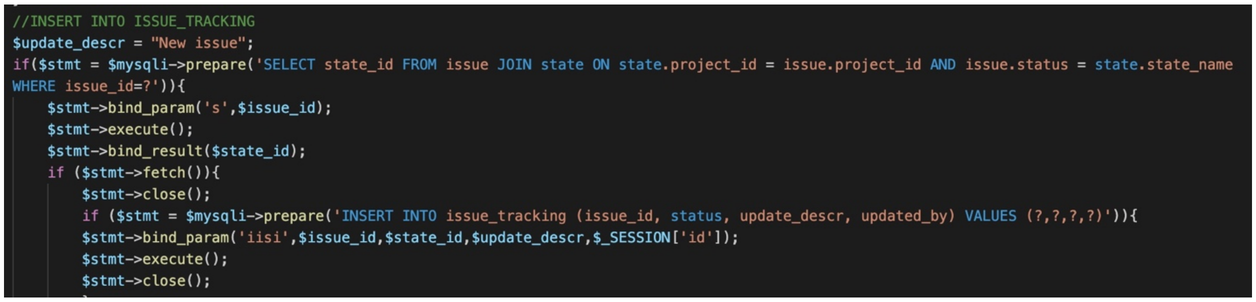


Figure 7: Create New Issue Page

After clicking to create the issue, *issue_title* and *issue_descr* are inserted into the *issue* relation and the *issue_tracking* relation. Prior to inserting into *issue_tracking*, a query must be performed on the *issue* relation to fetch the initial state of the new issue, which was pre-determined by the project's workflow and automatically inserted into the *status* field of the *issue* relation through a trigger event. Upon obtaining the *state_id,* an insertion can be made into the *issue_tracking* relation with the *updated_by* attribute being set to the current user, and *update_descr* is automatically set to "New issue". Queries used to update the *issue_tracking* relation are provided in Figure 8. The Reporter is inserted into the *issue_role* relation by inserting the $_SESSION['id'] and associating it with *role_id* = 3, representing the Reporter *role_id.* The Assignee is inserted into the *issue_role* relation by looking up the *user_id* of the user's Assignee selection and associating it with *role_id* = 4, representing the Assignee *role_id*.



```
//INSERT INTO ISSUE_TRACKING
$update_descr = "New issue";
if($stmt = $mysqli->prepare('SELECT state_id FROM issue JOIN state ON state.project_id = issue.project_id AND issue.status = state.state_name
WHERE issue_id=?')){
    $stmt->bind_param('s',$issue_id);
    $stmt->execute();
    $stmt->bind_result($state_id);
    if ($stmt->fetch()){
        $stmt->close();
        if ($stmt = $mysqli->prepare('INSERT INTO issue_tracking (issue_id, status, update_descr, updated_by) VALUES (?,?,?,?)')){
        $stmt->bind_param('iisi',$issue_id,$state_id,$update_descr,$_SESSION['id']);
        $stmt->execute();
        $stmt->close();
```

Figure 8: Queries to insert into the *issue_tracking* relation

The user is then redirected back to the Issues Page. Since the *project_id* is saved in the session, we are redirected to the queried Issues page for the particular project we were viewing.

5.1.6 Add Project Lead & Add Team Member Pages

On the Issues Page, the pencil next to the list of Project Leads and list of Team Members is only viewable by the Project Lead on a particular project. Once the pencil is selected, the user may add additional Project Leads or Team Members depending on which pencil they clicked.

For adding a new Project Lead, the user is redirected to a page with a dropdown list of available employees who either already work on the project but not as a Project Lead, or employees who do not work on the project at all, as shown in Figure 9.

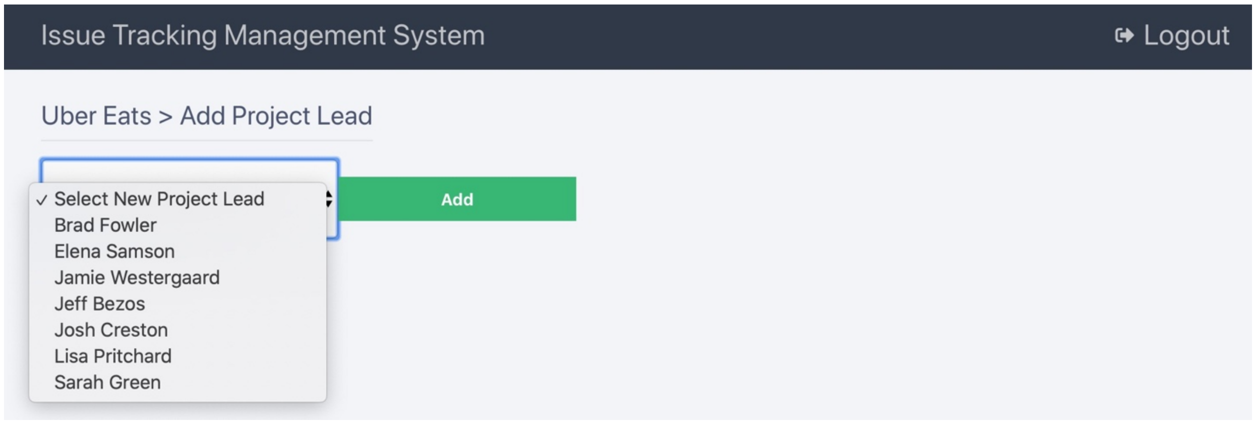
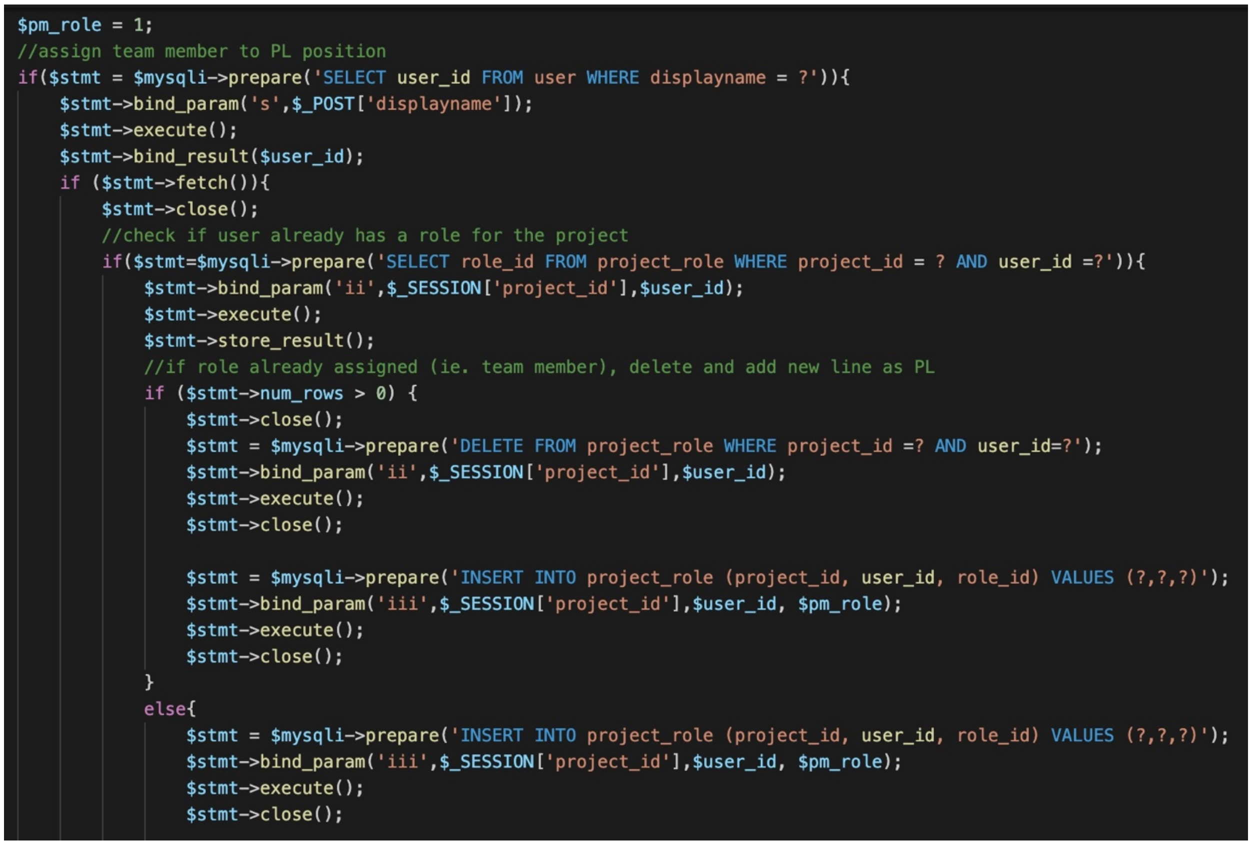


Figure 9: Add Project Lead Page

Once the new Project Lead has been selected, a series of queries is performed. Since the *project_role* relation has primary keys *project_id, role_id, user_id*, a user can only have one role on a specific project. Therefore, since some of these users may already exist as Team Members on the project, this instance must be deleted, and then a new instance for their new Project Lead role must be added. This was done using the queries shown in Figure 10. The new instance is added by looking up the selected user's *user_id* and associating it with *role_id* = 1, representing the Project Lead role. As suspected, if the user was previously a Team Member, this will cause them to be removed from the list of "Team Members" on the Issues Page, and now appear on the list of "Project Leads".



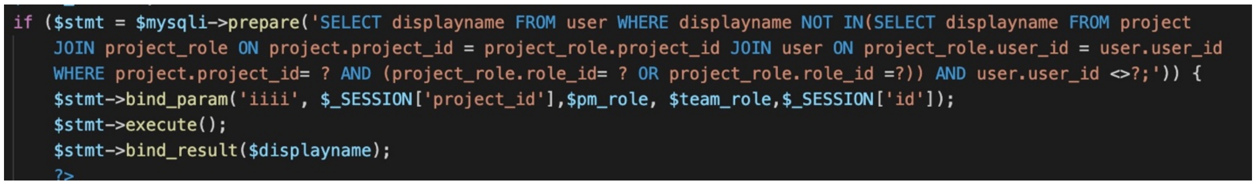
```
$pm_role = 1;
//assign team member to PL position
if($stmt = $mysqli->prepare('SELECT user_id FROM user WHERE displayname = ?')){
    $stmt->bind_param('s',$_POST['displayname']);
    $stmt->execute();
    $stmt->bind_result($user_id);
    if ($stmt->fetch()){
        $stmt->close();
        //check if user already has a role for the project
        if($stmt=$mysqli->prepare('SELECT role_id FROM project_role WHERE project_id = ? AND user_id =?')){
            $stmt->bind_param('ii',$_SESSION['project_id'],$user_id);
            $stmt->execute();
            $stmt->store_result();
            //if role already assigned (ie. team member), delete and add new line as PL
            if ($stmt->num_rows > 0) {
                $stmt->close();
                $stmt = $mysqli->prepare('DELETE FROM project_role WHERE project_id =? AND user_id=?');
                $stmt->bind_param('ii',$_SESSION['project_id'],$user_id);
                $stmt->execute();
                $stmt->close();

                $stmt = $mysqli->prepare('INSERT INTO project_role (project_id, user_id, role_id) VALUES (?,?,?)');
                $stmt->bind_param('iii',$_SESSION['project_id'],$user_id, $pm_role);
                $stmt->execute();
                $stmt->close();
            }
            else{
                $stmt = $mysqli->prepare('INSERT INTO project_role (project_id, user_id, role_id) VALUES (?,?,?)');
                $stmt->bind_param('iii',$_SESSION['project_id'],$user_id, $pm_role);
                $stmt->execute();
                $stmt->close();
```

Figure 10: Queries to add new Project Lead in *project_role* relation

Amanda Kuznecov
Net ID: anr431

A similar procedure is performed when adding a new Team Member, but instead, a dropdown list shows all users currently not involved in the project who are available to work on it. The query involves searching for users who do not have a project role on the particular project as shown in Figure 11. Since the employee has yet to be involved in the project, there is no need to search for prior history of the employee in the *project_role* relation. Instead, the new Team Member can immediately be inserted into the *project_role* relation. The new instance is added by looking up the selected user's *user_id* and associating it with *role_id* = 2, representing the Team Member role.



```
if ($stmt = $mysqli->prepare('SELECT displayname FROM user WHERE displayname NOT IN(SELECT displayname FROM project
    JOIN project_role ON project.project_id = project_role.project_id JOIN user ON project_role.user_id = user.user_id
    WHERE project.project_id= ? AND (project_role.role_id= ? OR project_role.role_id =?)) AND user.user_id <>?;')) {
    $stmt->bind_param('iiii', $_SESSION['project_id'],$pm_role, $team_role,$_SESSION['id']);
    $stmt->execute();
    $stmt->bind_result($displayname);
    ?>
```

Figure 11: Query to display employees available to become Team Members

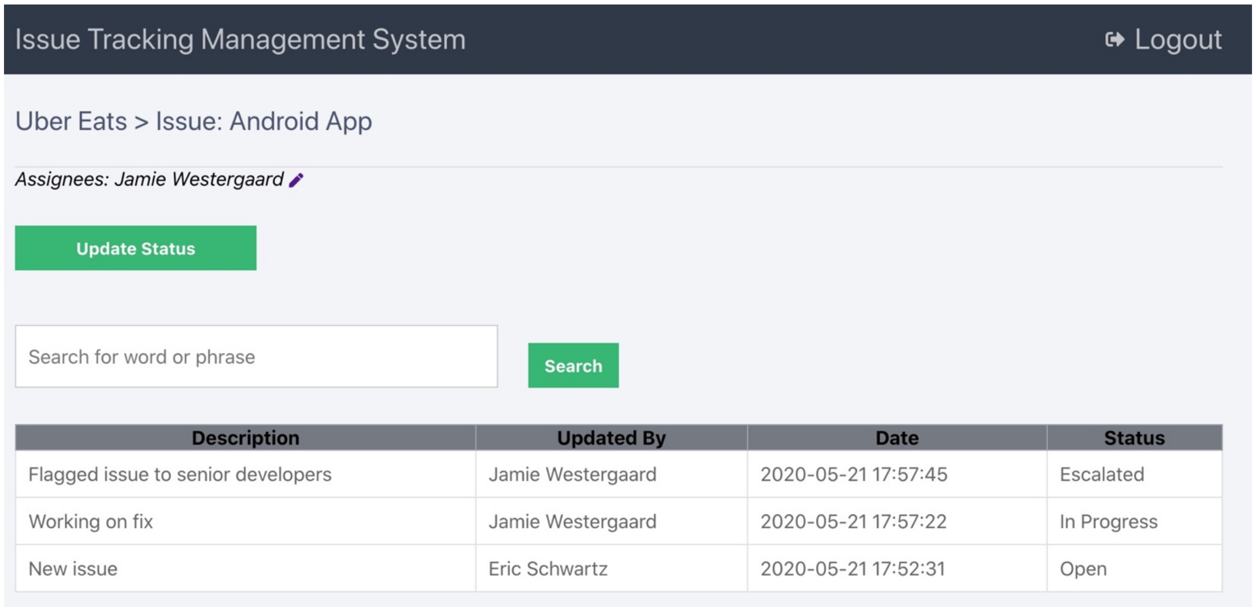
The user is then redirected back to the Issues Page. Since the *project_id* is saved in the session, we are able to be redirected to the queried Issues page for the particular project we were viewing.

5.1.7 Issue History Page

The Issue History Page shows all past status updates for the particular issue a user has clicked on, as shown in Figure 12. This information includes a description of the update, who the update was made by and when, and the status it was updated to.

Session variables $_SESSION['issue_id'] and $_SESSION['issue_title'] are updated based on the current issue being viewed. These variables are used as we move through the web application, but once a different issue is selected on the Issues Page, they are automatically modified to reflect that. This helps when we move to a different page relating to the particular issue and want to perform a query based on *issue_id* or *issue_title.*

A list of all Assignees for the particular issue is displayed at the top of the page for easy reference. In order to do so, an array of the query results is created by looping through each result, concatenating them, and then using the implode function to separate each value by a comma. A current user who is a Project Lead on the issue's project is able to add Assignees by clicking on the pencil next to this list. The pencils are hidden from users who are not Project Leads on the particular project being viewed. Assignees and Project Leads are able to update the status of a particular issue by clicking on the button to update status. Additionally, users are given the option of filtering the table rows by entering a key word or phrase that they are searching for. This was done by creating a search form and using the LIKE operator to match the user's inputs with the table results, ultimately filtering out anything that does not match. If no search word or phrase has been entered, the query to show project issues is performed as usual without the filter.


Issue Tracking Management System ⇨ Logout

Uber Eats > Issue: Android App

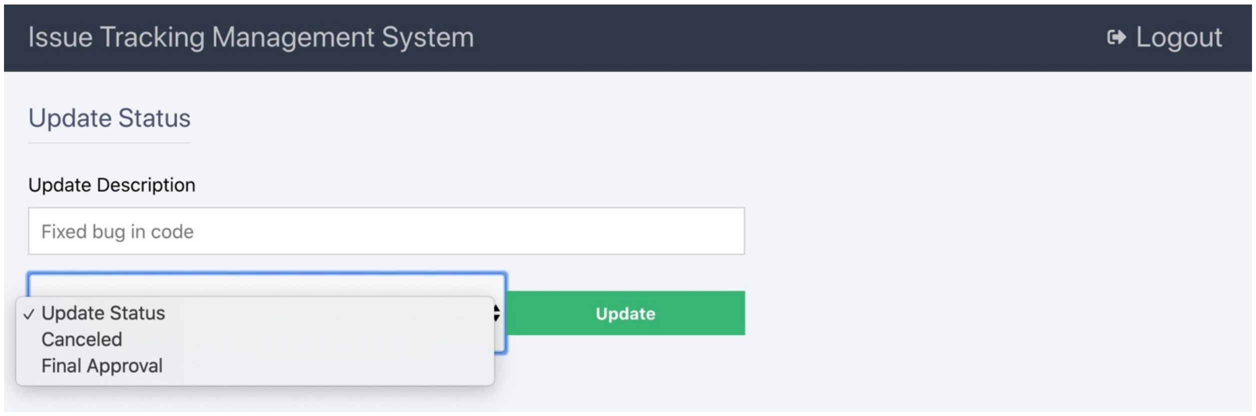
Assignees: Jamie Westergaard

Description	Updated By	Date	Status
Flagged issue to senior developers	Jamie Westergaard	2020-05-21 17:57:45	Escalated
Working on fix	Jamie Westergaard	2020-05-21 17:57:22	In Progress
New issue	Eric Schwartz	2020-05-21 17:52:31	Open

Figure 12: Issue History Page

5.1.8 Update Status Page

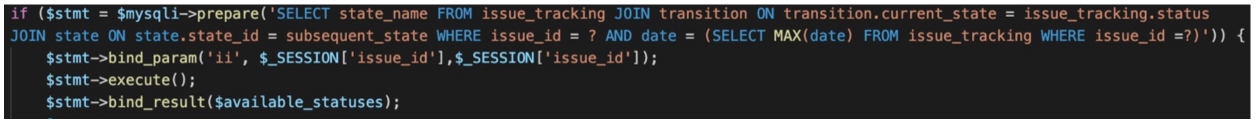
On the Update Status Page, the user has the option to enter a description of the status update and select the next state for the issue. The user is provided with a dropdown menu of possible legal subsequent transitions the issue can make, as shown in Figure 13. This was done by searching for *current_state* from the most recent status history within the *transition* relation, and then outputting all next possible *subsequent_state* transitions, as shown in Figure 14.


Issue Tracking Management System ⇨ Logout

Update Status

Update Description: Fixed bug in code

Figure 13: Update Status Page


```
if ($stmt = $mysqli->prepare('SELECT state_name FROM issue_tracking JOIN transition ON transition.current_state = issue_tracking.status
JOIN state ON state.state_id = subsequent_state WHERE issue_id = ? AND date = (SELECT MAX(date) FROM issue_tracking WHERE issue_id =?)')) {
    $stmt->bind_param('ii', $_SESSION['issue_id'],$_SESSION['issue_id']);
    $stmt->execute();
    $stmt->bind_result($available_statuses);
```
Figure 14: Query for *subsequent_state* based on *current_state*

Upon selecting the new status, the status and update description are inserted into the *issue_tracking* relation, along with the current user, who performed the update. The user is then redirected back to the

Issue History Page. Since the *issue_id* is saved in the session, we are able to be redirected to the queried Issue History page for the particular issue we were viewing.

5.1.9 Add Assignee Page

On the Issue History Page, the pencil next to the list of Assignees is only viewable by the Project Lead on a particular project. Once the pencil is selected, the user may add additional Assignees to the issue. The user is redirected to a page, as shown in Figure 15, with a dropdown list of available employees who already work on the project but have not been assigned to the issue. Once the Assignee has been selected, they are inserted into the *issue_role* relation with the *user_id* of the selected user and *role_id* = 4, corresponding to the Assignee role. The user is then redirected back to the Issue History Page. Since the *issue_id* is saved in the session, we are able to be redirected to the queried Issue History page for the particular issue we were viewing.

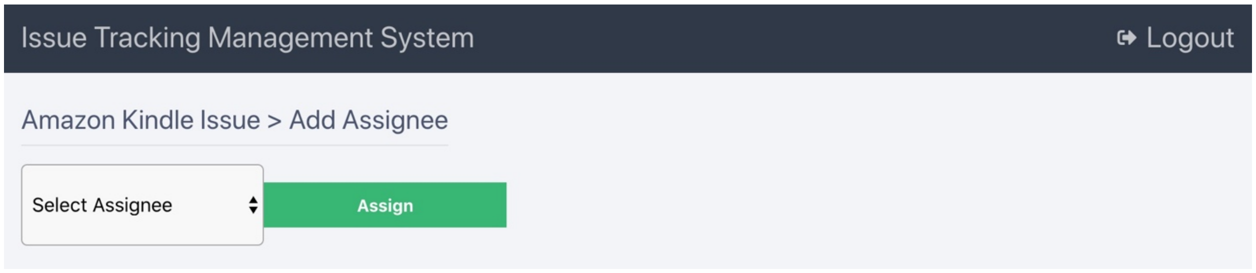

Issue Tracking Management System
Logout
Amazon Kindle Issue > Add Assignee
Select Assignee
Assign

Figure 15: Add Assignee Page

5.2 Security Measures

For security purposes, user passwords are not directly stored in the database. The password_hash function was used to encrypt each user's password as a means of preventing the password from being exposed in the event that the database becomes vulnerable. When a user logs in to the system, the password_verify function is used on the encrypted password to verify the hashed password stored in the *user* relation. The function is safe against blind SQL injections.

The web application was set up to prevent SQL injection from occurring. All SQL statements have been set up as prepared statements with parameterized queries. Parameters have been specified using a '?', signaling the database engine what to filter on. Parameters are binded to placeholders by stating each variable as well as its type. Statements are sent to the database server and parsed separately from the parameters. They are not run until the execute command has been called, which then runs the prepared statement combined with the specified parameter values. Since the statement is sent to the database separately from the parameters, it limits the risk of SQL injection.

In order to protect from an XSS attack, all user inputs in text boxes are validated to ensure the correct data is being used within the system, and preventing any malicious data from entering, potentially causing harm to the site, database, and users. Additionally, the htmlspecialchar function is utilized when users have the option of typing text data. The function ensures that the user input is escaped, meaning that any potentially dangerous characters such as < or > are converted to plain text as "&lt" and "&gt".This prevents an attacker from inputting any potentially harmful URLs and prevents any HTML from being executed. Below is a list of specific pages of the web application and how these measures were specifically implemented:

Amanda Kuznecov
Net ID: anr431

- Registration Page: All user inputs for *first_name*, *last_name*, *username* and *displayname* are checked for validity by matching input on letters and numbers only, and not allowing any special characters. The FILTER_VALIDATE_EMAIL function was used to check the validity of entered email addresses.

- Issues Page & Issue History Page: The function htmlspecialchar is used to convert user's input from the search text box field.

- Create New Project Page: A check is performed on *project_name* and *project_descr* inputs from the user, validating the input by matching it on letters and numbers only, and not allowing any special characters. The function htmlspecialchar is used on the inputs for workflow *status_name* before inserting into the database.

- Create New Issue Page: A check is performed on *issue_title* and *issue_descr* inputs from the user, validating the input by matching it on letters and numbers only, and not allowing any special characters.

- Update Status: A check is performed on the user's input for *update_descr* to match it on letters and numbers only, and not allowing any special characters.