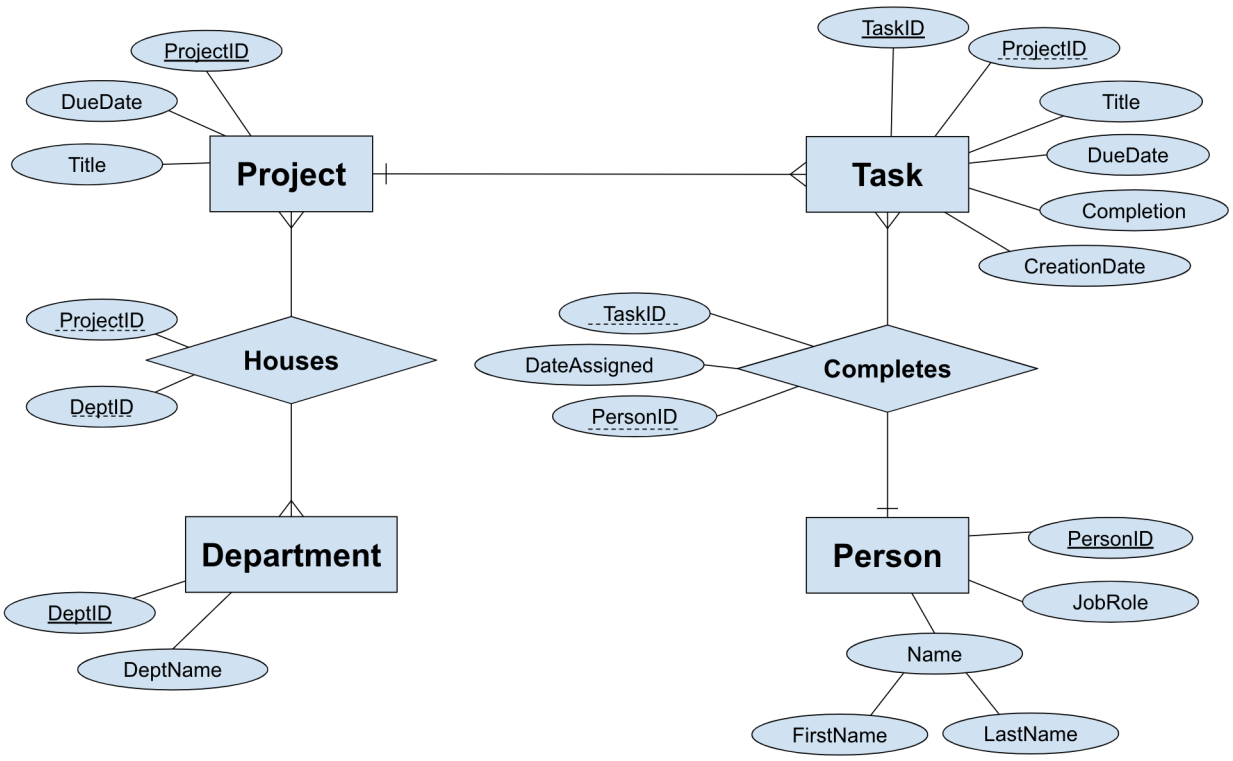
Task Organizer

Bennett Wendorf and Connor Marks

Abstract

The idea for this project comes largely from something that we, as students and software developers, are intimately familiar with. The need for most individuals and teams to have some way to manage their tasks on a given project, class, or day is something that many if not all students and developers have had to struggle with. Our goal with this project is to create a good solution for managing tasks for whatever use case our users might need. We also included a more overarching idea of a project that tasks can be a part of. This would be especially useful for teams that are working on multiple projects to help them organize what they need to work on and what those individual tasks are a part of. Much of the inspiration for how to format this project comes from Microsoft To Do. Microsoft To Do is a task management software that allows management of tasks for a single user. One of our main goals was to take this idea and expand on it to allow management of tasks for multiple projects as well as teams of people. We chose to implement this as a web app so that it can conceivably be run on cloud infrastructure and be accessed from anywhere. In theory, this structure would also allow us to easily implement a mobile app as a secondary platform for accessing the same data. We wanted to provide a fully functional task management system that one could easily use to provide task and project management functionality for a single user, all the way up to a large team with a clean UI and easy access to all the data one might need.

Database Design



Project: This table stores the data for various projects. The idea with projects is to provide a way to organize groups of tasks so they can be more easily organized and referenced. Projects have a projectID to uniquely identify them and also have a title that may identify them but will be more of a description. Projects also have a DueDate which represents when all tasks in the project should be completed by.

Task: Task management is the main focus of this program and that is why this table has the most attributes in it and is the most important. TaskID is an auto-incremented value and is used to uniquely identify tasks in the database. Title gives tasks a more meaningful name then just an ID number. All tasks have a creation date and due date assigned to make sure tasks are completed in a timely manner. Tasks also contain a boolean attribute called completion that gets marked true in the database when they are completed.

Completes: Every task could have a person assigned to it , which is the main purpose of this table. This table maintains information about what people are assigned to what tasks. It has two foreign keys, the primary keys from person and task, but it also has a date assigned attribute. This attribute differs from the created date because not all tasks have to be assigned to a person right away. Completion date allows us to see how long a task has been alive and date assigned allows us to see how long someone has been working on a task.

Person: This is a straightforward table. People complete tasks, so we need to store information on those individuals. Each person has a person ID that uniquely identifies them. Each person also has a first and last name, as well as their job title or role. There is no need to have a ton of information about the people because the main goal of this is task management so as long as each person can be uniquely identified the application will run well.

Houses: This table holds a relationship between a department and a project. It is necessary since that relationship is a many to many relationship, meaning that each department can house multiple projects and a single project might be co-managed by multiple departments. All we care about in this relationship is the department and project that are related, so we only have two foreign keys: ProjectID and DeptID.

Department: This table holds some information about departments that projects are housed in. A project may be housed across multiple departments, and also each department might be managing or helping manage many projects, so the relationship between departments and projects is many to many. It is useful to store this information so we can see in the application what departments are doing well in terms of

productivity. This entity only stores some basic information necessary to identify a department by name and a unique ID. This ID attribute will be auto-incremented.

Functionality

- Department Page:
    - Filter what department is showing
    - List all people that have worked for the specified department

- Project Page:
    - Number of tasks remaining in each project

- Users Task Page:
    - Display Users Tasks/All Tasks
    - Popup window for adding new tasks with spaces for user input for many of the attributes
    - Ability to click on an existing task to modify things like its title, due date, etc.
        - Button in this modification interface for deleting a given task

- Management Page:
    - Find users that are free (i.e. Have no tasks assigned or whose tasks are all completed)
    - Display best user (Most Tasks completed)

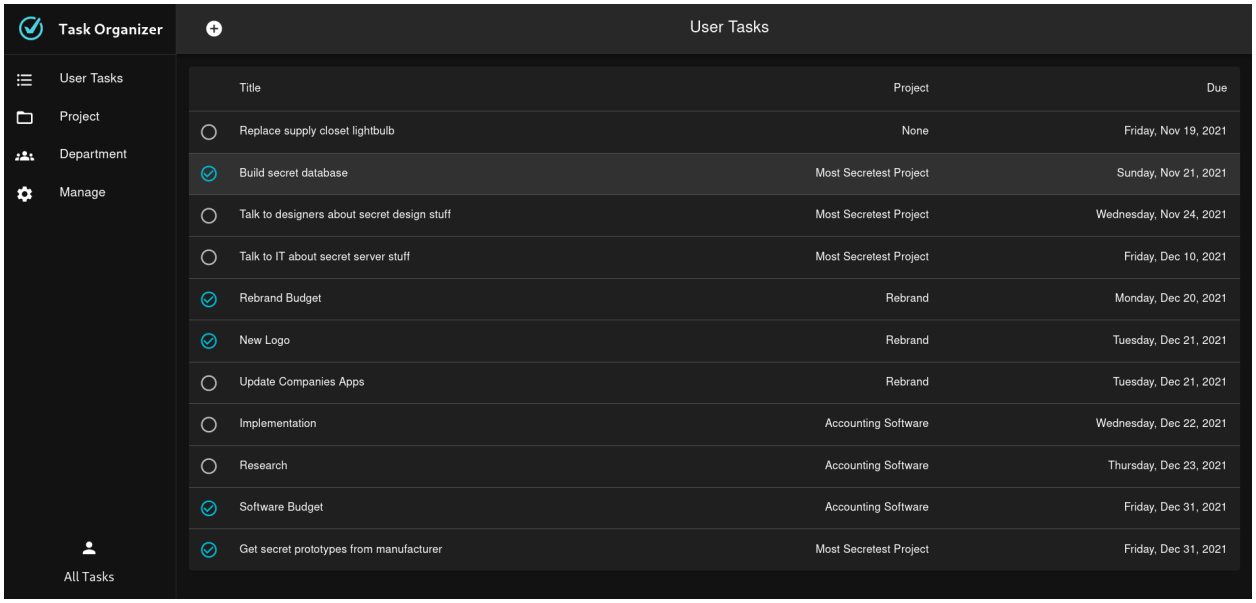
Stakeholders

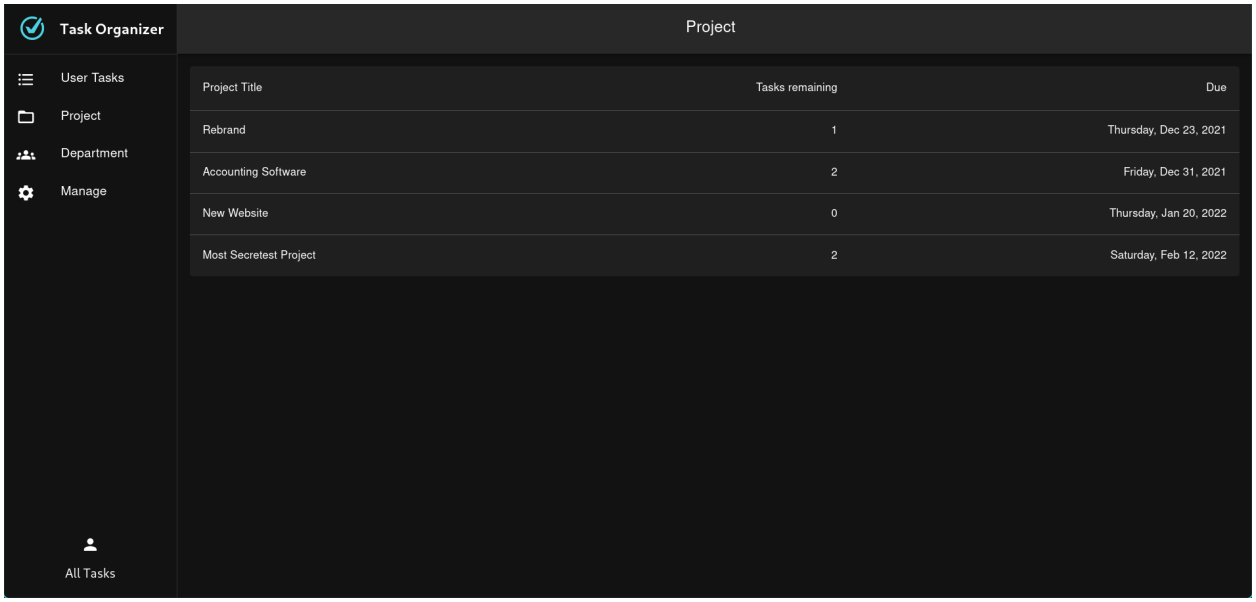
This web application is multi-purpose and very beneficial to a wide group of people. It could be used by an individual user that likes to manage their upcoming tasks and projects in a straightforward easy way to use. In this case there would only be one tuple in the person entity. It could also be advantageous to any business, large or small, that is interested in completing projects and tasks. A business would be able to add all of their employees (Persons) and then can start adding projects, tasks associated with each project and then managers could assign tasks to their employees, all in a very organized way. This application would be for anyone that wants to stay up to date and organized on tasks and projects they have to complete.

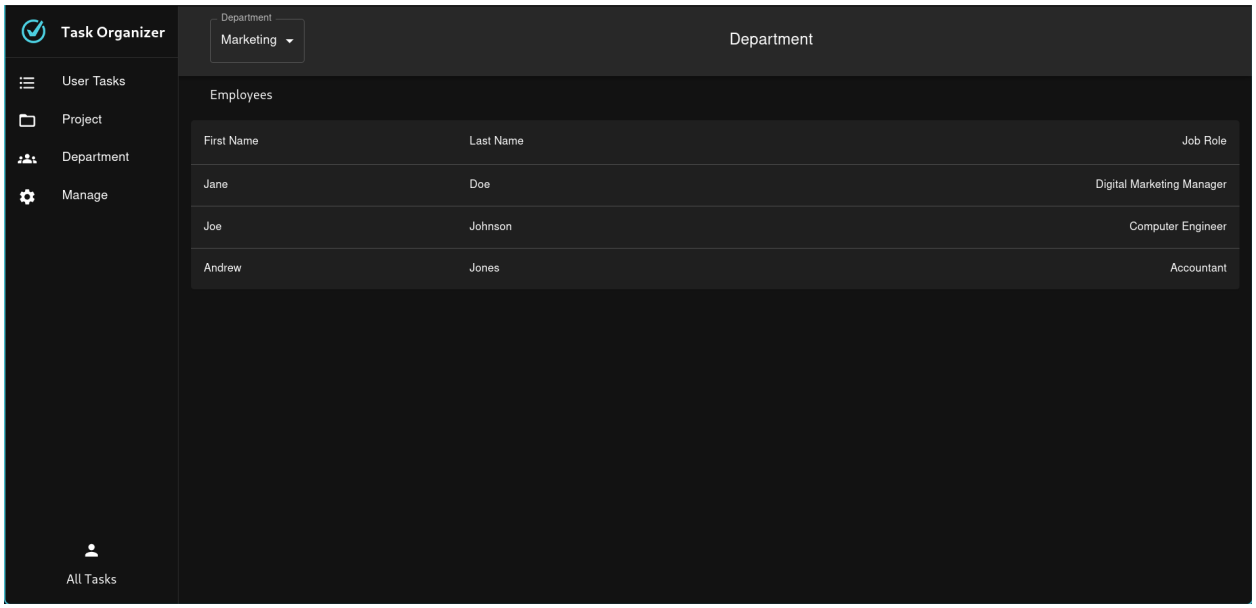
Technological Requirements

For this project we built a web app using React for a front end and Node.js for the back end. Both React and Node are in javascript, so it was fairly simple to be able to work with both. Neither of us have experience directly with these two frameworks, but we both have some experience with web design. I (Bennett) also have a friend who is knowledgeable about React and Node who is willing to help out if we get a bit stuck on that end. We also used SQLite for the database. This makes the most sense to us since it is what we covered in class and what we are both the most familiar with. This allowed us to spend more of our time learning the UI frameworks and writing database queries, rather than trying to learn a different SQL as well. Node has a module called sqlite3 for integrating with this database, which is what we used for the database connector. In terms of sharing code, we have a GitHub repository set up to do this. Connor did not have much experience with git or GitHub when we started, but I (Bennett) have been using git for a while and know my way around GitHub fairly well and helped him wherever needed.

Screenshots







Advanced Queries

Group 1:

```
1: SELECT Project.Title,
count(Task.Completion)-sum(Task.Completion) AS TaskRemaining,
Project.DueDate

     FROM Project LEFT JOIN Task
```

```
        ON Project.ProjectID = Task.ProjectID
    GROUP BY Project.ProjectID
    ORDER BY Project.DueDate ASC
```

Description: This query returns all projects in the database and returns their title, the amount of tasks remaining and due date for each.

```
2: SELECT Task.TaskID, Task.Title, Task.Completion,
Task.DueDate, Task.CreationDate, Task.ProjectID, Project.Title
AS ProjectTitle, Completes.PersonID
    FROM Completes JOIN Task
    ON Completes.TaskID = Task.TaskID
    LEFT JOIN Project
        ON Task.ProjectID = Project.ProjectID
    WHERE Completes.PersonID = ${req.params.id}
    ORDER BY Task.DueDate ASC
```

Description: This query returns all tasks in the database that are assigned to a specific user, and orders the results by their due date.

Group 2:

```
1: SELECT Person.FirstName, Person.LastName, Person.JobRole
    FROM Person JOIN Completes JOIN Task JOIN Project JOIN
    Houses
        ON Person.PersonID = Completes.PersonID
            AND Completes.TaskID = Task.TaskID
            AND Task.ProjectID = Project.ProjectID
            AND Project.ProjectID = Houses.ProjectID
    WHERE Houses.DeptID = ${req.params.id}
```

```
     GROUP BY Person.PersonID

     ORDER BY Person.LastName
```

Description: This query returns a list of people's first names, last names and job roles that have worked for a specific department.

```
2: SELECT Task.TaskID, Task.Title, Task.Completion,
Task.DueDate, Task.CreationDate, Task.ProjectID, Project.Title
AS ProjectTitle, Completes.PersonID

     FROM Task LEFT NATURAL JOIN Completes LEFT JOIN Project

          ON Task.ProjectID = Project.ProjectID

     ORDER BY Task.DueDate ASC
```

Description: This query pulls all tasks from the database, using left join to ensure that we get tasks even if no person is assigned to that task or if that task is not part of a project.

Group 3:

```
1: SELECT PersonID, FirstName, LastName, JobRole,
Max(CompletedTasks) AS TasksCompleted

     FROM (SELECT PersonID, FirstName, LastName, JobRole,
Count(*) AS CompletedTasks

          FROM Person NATURAL JOIN Completes NATURAL JOIN Task

          WHERE Completion = 1

          GROUP BY PersonID)
```

Description: This query finds the “best user”, the user who has completed the most tasks, and returns their information.

```
2:SELECT PersonID, FirstName, LastName, JobRole

     FROM (SELECT PersonID, FirstName, LastName, JobRole,
Task.Completion
```

```
        FROM Person NATURAL JOIN Completes NATURAL JOIN Task
        GROUP BY PersonID, Completion
        ORDER BY Completion ASC)
    GROUP BY PersonID
    HAVING Completion = 1
```

Description: This query finds users whose assigned tasks are all marked completed.