DIYup

Team 206
Phyo Htut (phutut@mail.sfsu.edu) [Team Lead]
Eduardo Ramos (eramos4@mail.sfsu.edu) [QA/UI/UX]
Myles Pedronan (mpedrona@mail.sfsu.edu) [Backend Lead]
Antonio Carmona (acarmona@mail.sfsu.edu) [DB/Git Master]
Jason Wei (zwei@mail.sfsu.edu) [Frontend Lead]

Software Engineering
CSC 648/848 Section 4
Fall 2019

Version Table

Date	Milestone	Version
10/17/19	Milestone 2	1.0.0
10/10/19	Milestone 1	1.0.1
10/3/19	Milestone 1	1.0.0

Table of Contents

Data Definitions

- *Users*
    - **Registered User:** A user of the site that is logged in (and has an account). A ***Registered User*** has a unique *email address,* a unique *username*, an *avatar* image, and *password*.
    - **Guest User:** A user of the site that is not logged in (and may have an account). The information of a ***Guest User*** is not stored in any form.
    - **Admin:** A site administrator that is logged in (and has an account). An ***Admin*** has a unique *email address*, a unique *username*, and *password*.

- *Entities*
    - **Tutorial:** A tutorial posted by a ***Registered User***. A ***Tutorial*** is associated with the *email address* of a single ***Registered User*** and has a unique *ID*, a *title*, an *image*, a *category*, a *description*, an *author difficulty*, a *viewer difficulty*, a list of *materials*, a list of *tools*, and a *rating*.
    - **Step:** A single step of a ***Tutorial***. A ***Step*** is associated with the *ID* of a single ***Tutorial***, has a unique *ID*, some *content*, an *image*, and is associated with the *IDs* of the previous and next ***Steps*** in the ***Tutorial*** that it is associated with.
    - **List:** A list of items associated with a ***Tutorial***. A ***List*** is associated with the ID of a single ***Tutorial***, has a unique ID, a list of *contents*, and a list of *links*.
    - **Comment:** A comment posted on a ***Tutorial*** by a ***Registered User***. A ***Comment*** is associated with the *ID* of a single ***Tutorial*** and the *email address* of a single ***Registered User***. It has a unique *ID*, some content, an *image*, a *timestamp*, and is associated with the *IDs* of the previous and next ***Comments*** on the ***Tutorial*** that it is associated with.

- *Attributes*
    - **Category:** Used to describe the main area of work that a ***Tutorial*** is associated with. Several examples include Woodworking and Electronics.
    - **Difficulty:** Used to describe the amount of experience required in the specified **Category** associated with a ***Tutorial*** that is needed to complete the ***Tutorial***.
        - Author Difficulty: The difficulty of a ***Tutorial*** set by its author (a ***Registered User***).
        - Viewer Difficulty: The difficulty of a ***Tutorial*** set by ***Registered Users***.
    - **Rating:** Used to represent the quality of a ***Tutorial*** based on feedback from ***Registered Users***.

Functional Requirements

Guest User

Priority 1:

1. Guest users must be allowed to register for an account for each unique email.
2. Guest users must be allowed to search for tutorials posted by registered users.
3. Guest users must be allowed to view tutorials posted by registered users.
4. Guest users must be allowed to view comments posted by registered users.
5. Guest users must be allowed to view tutorials based on categories.
6. Guest users must not be allowed to comment on or post any tutorials.

Priority 2:

1. Guest users desired to be allowed to sort with difficult and rating.

Registered User

Priority 1:

1. Registered users must be allowed to log out or recover their account passwords..
2. Registered users must be allowed to search for tutorials posted by registered users.
3. Registered users must be allowed to view tutorials and the comments under the tutorials posted by registered users.
4. Guest users must be allowed to view tutorials based on categories.
5. Registered users must be allowed to post tutorials.
6. Registered users must be allowed to include tools lists and materials lists with their posted tutorials.
7. Registered users must be allowed to edit or delete their posted tutorials.
8. Registered users must be allowed to post comments.

9. Registered users must be allowed to delete their own posted comments.

Priority 2:

1. Registered users are desired to be allowed to rate tutorials posted by other registered users.
2. Registered users are desired to provide difficulties to tutorials posted by other registered users.
3. Registered users are desired to be allowed to edit their posted comments.
4. Registered users are desired to allowed to post pictures in their tutorials.
5. Guest users desired to be allowed to sort with difficult and rating.
6. Registered users are desired to be allowed to add tutorials posted by other registered users to a personal favorites list.
7. Registered users are desired to allowed to remove tutorials posted by other registered users from a personal favorites list.

Priority 3:

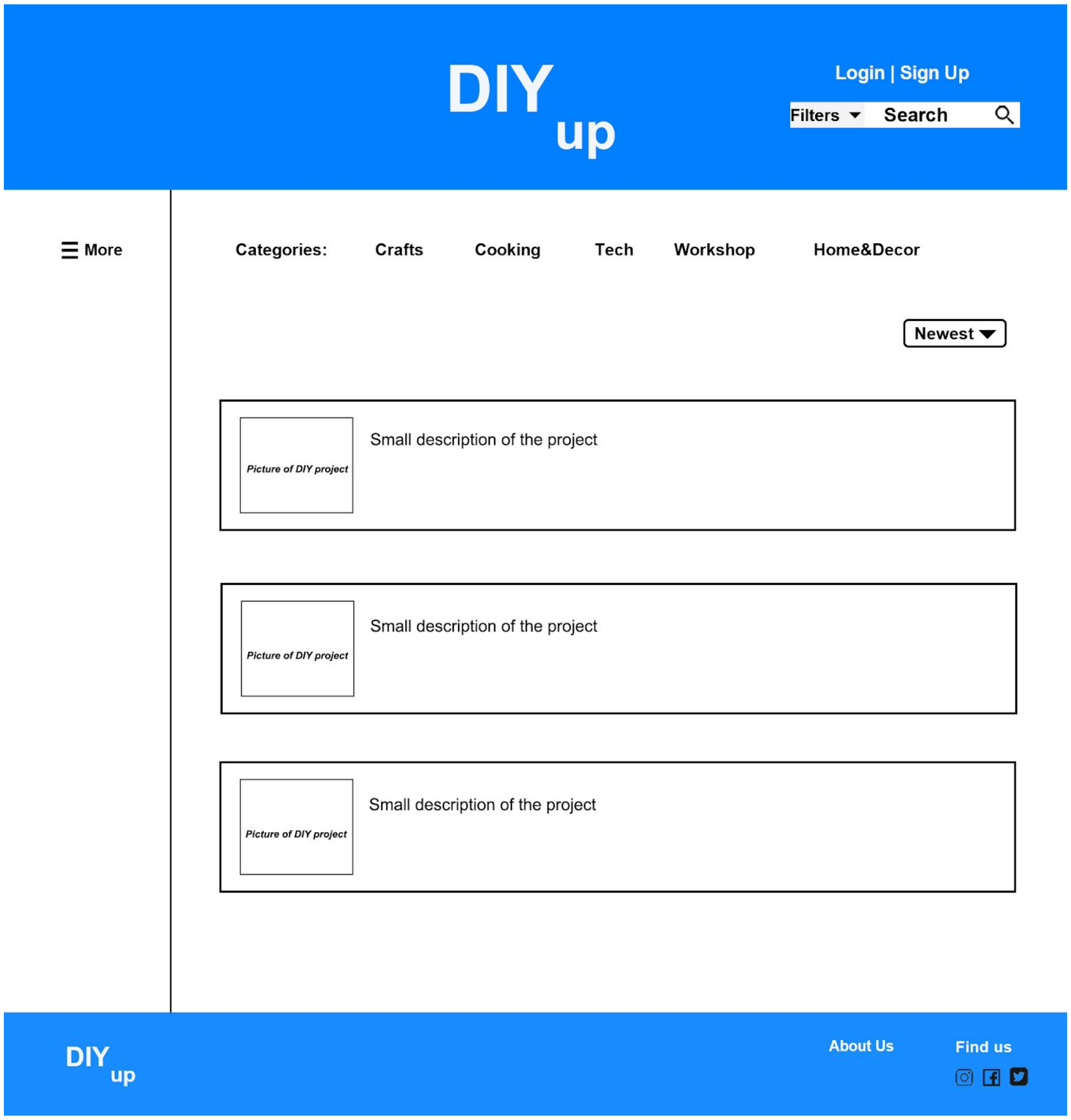
1. Registered users is good to be able to report the tutorials or comments posted by other registered users.
2. Registered users is good to have a verified code to recover their account passwords.

Admin

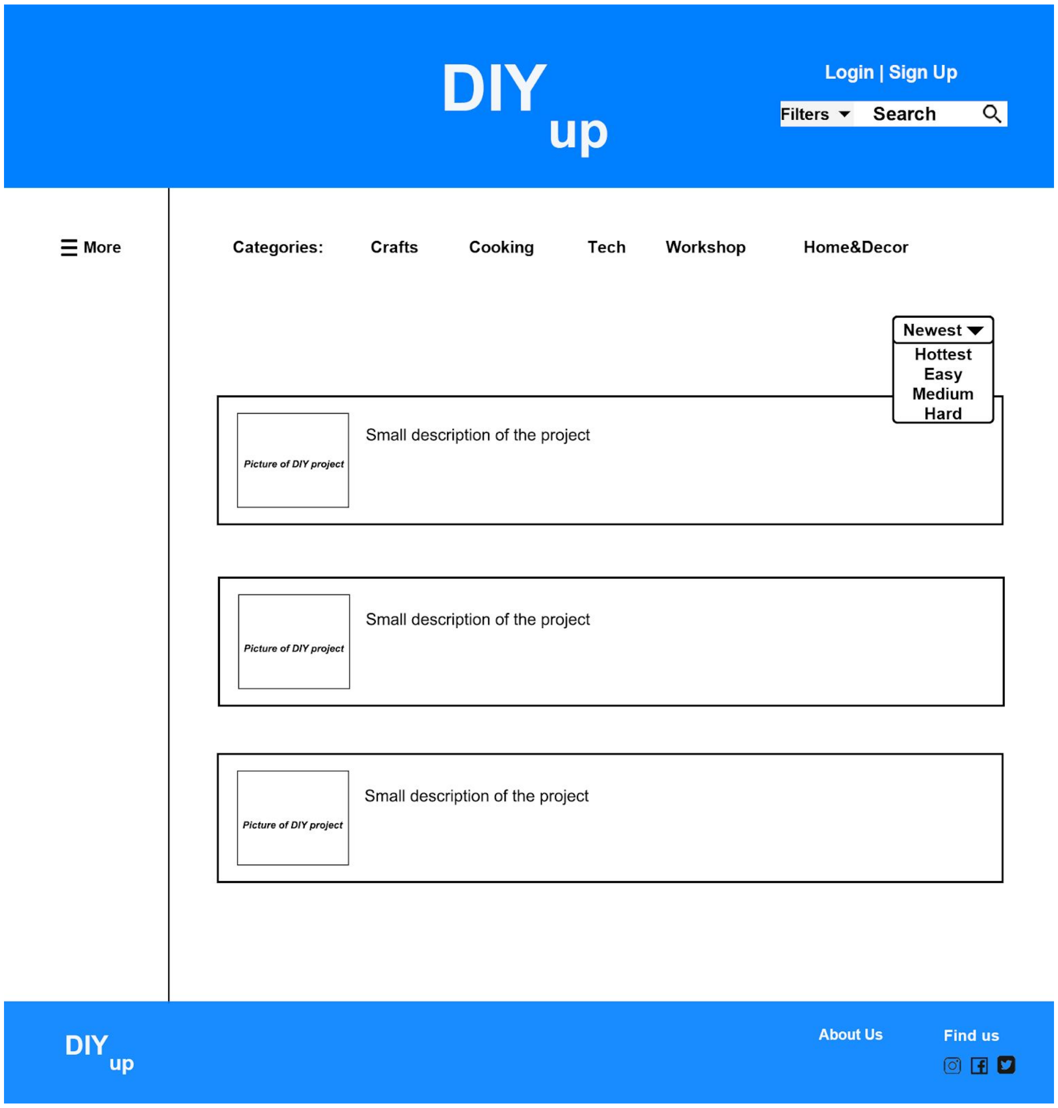
Priority 1:

1. Admin accounts must be reserved to the developers.
2. Admin accounts must have created on the server side.
3. Admins must have all of the abilities of registered users.
4. Admins must be allowed to delete any tutorials.
5. Admins must be allowed to delete any comments.

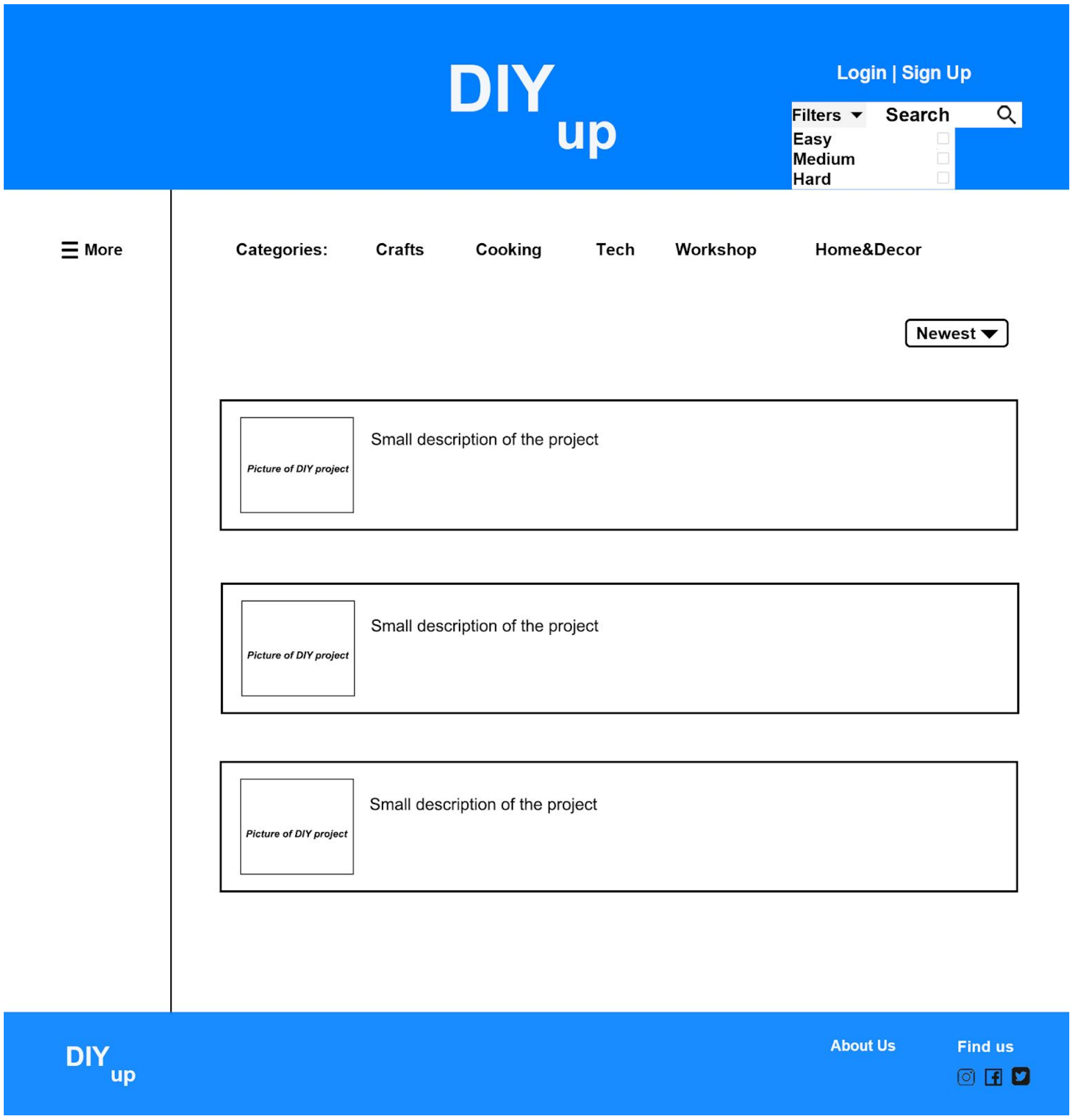
UI Mockups and Storyboards



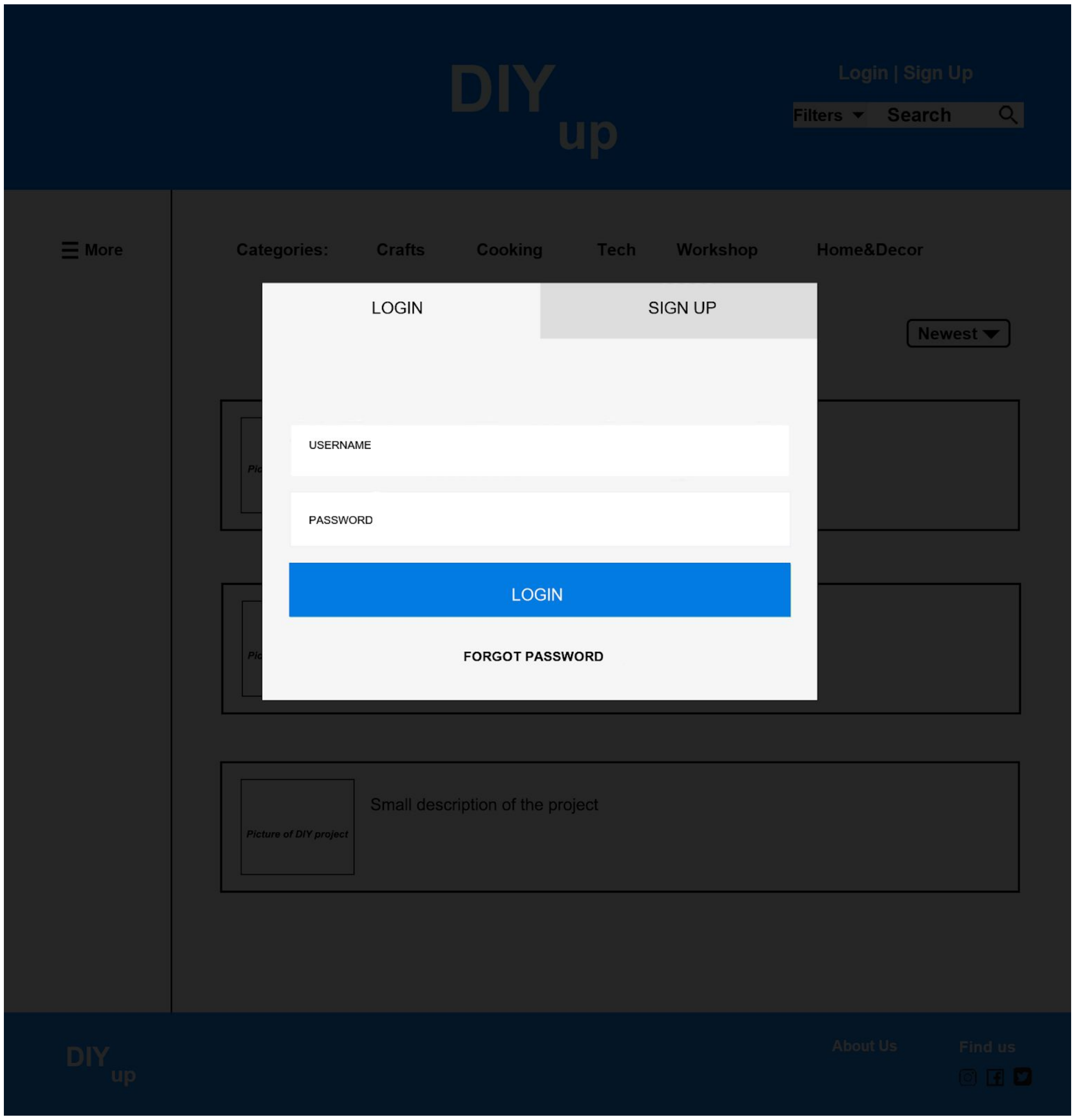
Main homepage for our website mockup where users can view, sort and search for DIY projects and view projects in different categories. Projects are presented with a small picture of the project and a small description of what the project is.



Users can sort through projects on the home page according to their liking such as the newest and hottest projects and also by the difficulty of the project such as easy, medium and hard. This makes it easy for users to be able to quickly find a DIY project that they like.



We made it easy for users looking for a specific project to find what they're searching for with the search bar at the top and also filter their search by the level of difficulty of the project to suit their level of building expertise.



Users can login with their username and password by hitting the login button on the top banner and also sign up for an account with the sign up button.

High Level Database Architecture and Organization

Business Rules

1. Users can post multiple Tutorials.
2. Users can post multiple Comments.
3. Tutorials belong to exactly one User.
4. Tutorials can have multiple Comments.
5. Tutorials must have at least one Step.
6. Tutorials must have exactly one list.
7. Comments belong to exactly one User.
8. Comments are made on exactly one Tutorial.
9. Steps are associated with exactly one Tutorial.
10. Lists are associated with exactly one Tutorial.

Entities, Attributes, and Relationships

- Users
  Users who have registered with DIYup and have login information (an email address and a password.

  - email_address (VARCHAR)
    A user's email address. This is used by a user to log into their account and is never publicly displayed. This must be unique for every user and is never null.

  - username (VARCHAR)
    A user's username. This is used to publicly identify a user and is attached to the tutorials and comments that a user posts. This must be unique for every user and is never null.

  - password (VARCHAR)
    A user's password. This is used by a user to log into their account and is stored in a hashed form for security reasons. This is never null.

  - is_admin (TINYINT)
    Indicates whether or not a user should have access to administrator tools. This is never null and has a default value of 0.

  - avatar (VARCHAR)
    A user's avatar image. This is displayed alongside a user's username in the tutorials and comments that a user posts.

- Tutorials

Tutorials that are posted by users.

  - id (INT)
    The unique ID used to identify a tutorial. This is unique, is never null, and is automatically incremented as new tutorials are created.

  - author_id (VARCHAR)
    The username of the user who created a tutorial. This is never null.

  - title (VARCHAR)
    The title of the tutorial. This is never null.

  - image (VARCHAR)
    The image displayed next to the title of the tutorial. This is stored in the form of a link to the image.

  - category (VARCHAR)
    The category that the tutorial belongs to. This is never null.

  - description (VARCHAR)
    The description of the tutorial that is displayed after the title of the tutorial. This is never null.

  - author_difficulty (DECIMAL)
    The difficulty of the tutorial (on a scale of 1 to 5) specified by the author of the tutorial.

  - viewer_difficulty (DECIMAL)
    The difficulty of the tutorial (on a scale of 1 to 5) averaged from difficulties specified by viewers of the tutorial.

  - rating (DECIMAL)
    The quality rating of the tutorial (on a scale of 1 to 5) averaged from the ratings specified by views of the tutorial.

  - keywords (VARCHAR)
    The keywords associated with the tutorial for search purposes.

- <u>Comments</u>
  Comments made on Tutorials by Users.

  - id (INT)
    The unique ID used to identify a tutorial. This is unique, is never null, and is automatically incremented as new comments are created.

  - author_id (VARCHAR)
    The username of the user who created a comment. This is never null.

  - tutorial_id (INT)

The ID of the tutorial that a comment was made on. This is never null.

  - content (VARCHAR)
    The text content of a comment. This is never null.

  - image (VARCHAR)
    The image displayed alongside a comment. This is stored in the form of a link to the image.

  - edited (TINYINT)
    Indicates whether or not a comment has been edited by the user that created it after its initial creation. This is never null and has a default value of 0.

- Steps
  Steps that make up the process described by a Tutorial.

  - tutorial_id (INT)
    The ID of the tutorial that a step belongs to. This is never null.

  - index (INT)
    The index of a step in the tutorial that it is associated with. This is never null.

  - content (VARCHAR)
    The text content of a step. This is never null.

  - Image (VARCHAR)
    The image displayed alongside the text content of a step. This is stored in the form of a link to the image.

- Lists
  Lists contain the materials and tools required to follow a Tutorial.

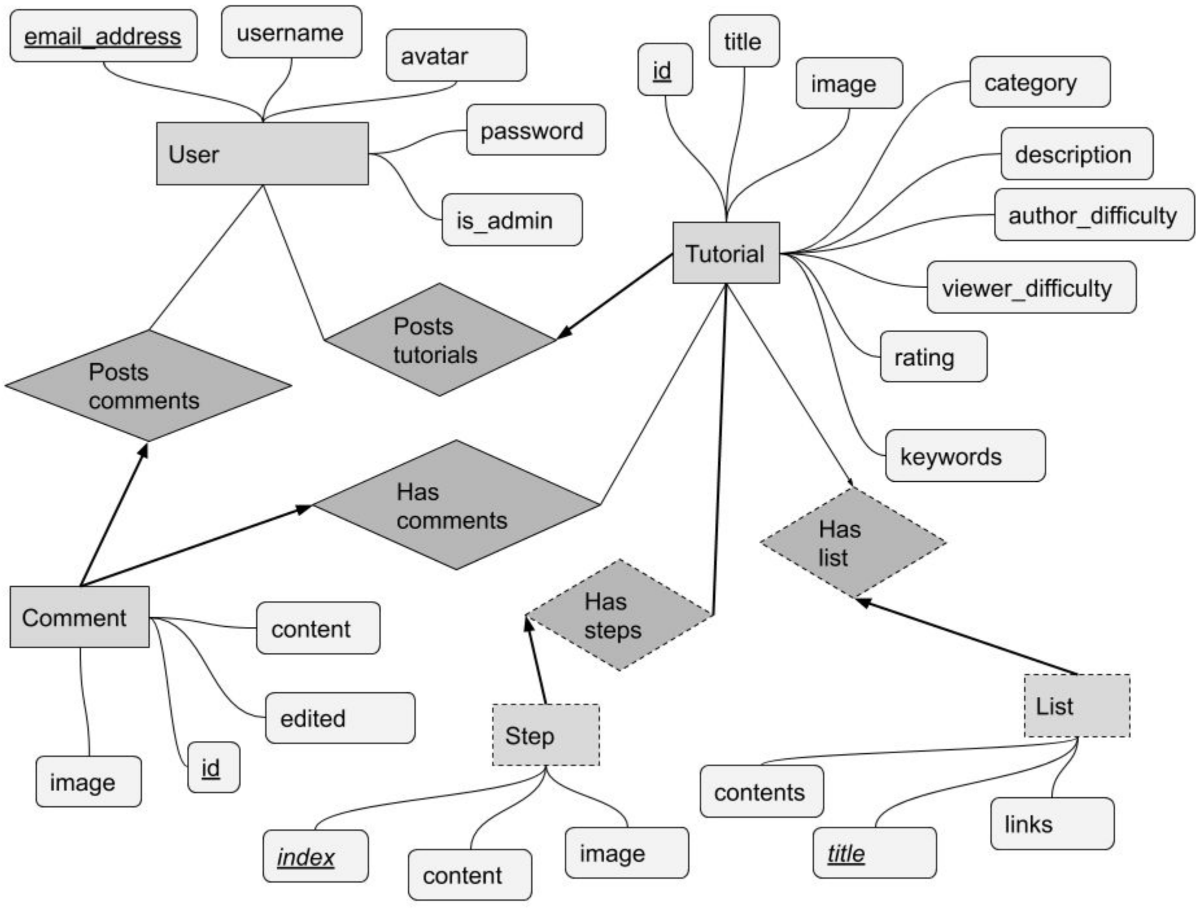
  - tutorial_id (INT)
    The ID of the tutorial that a list belongs to. This is never null.

  - title (VARCHAR)
    The title of the list. This is never null.

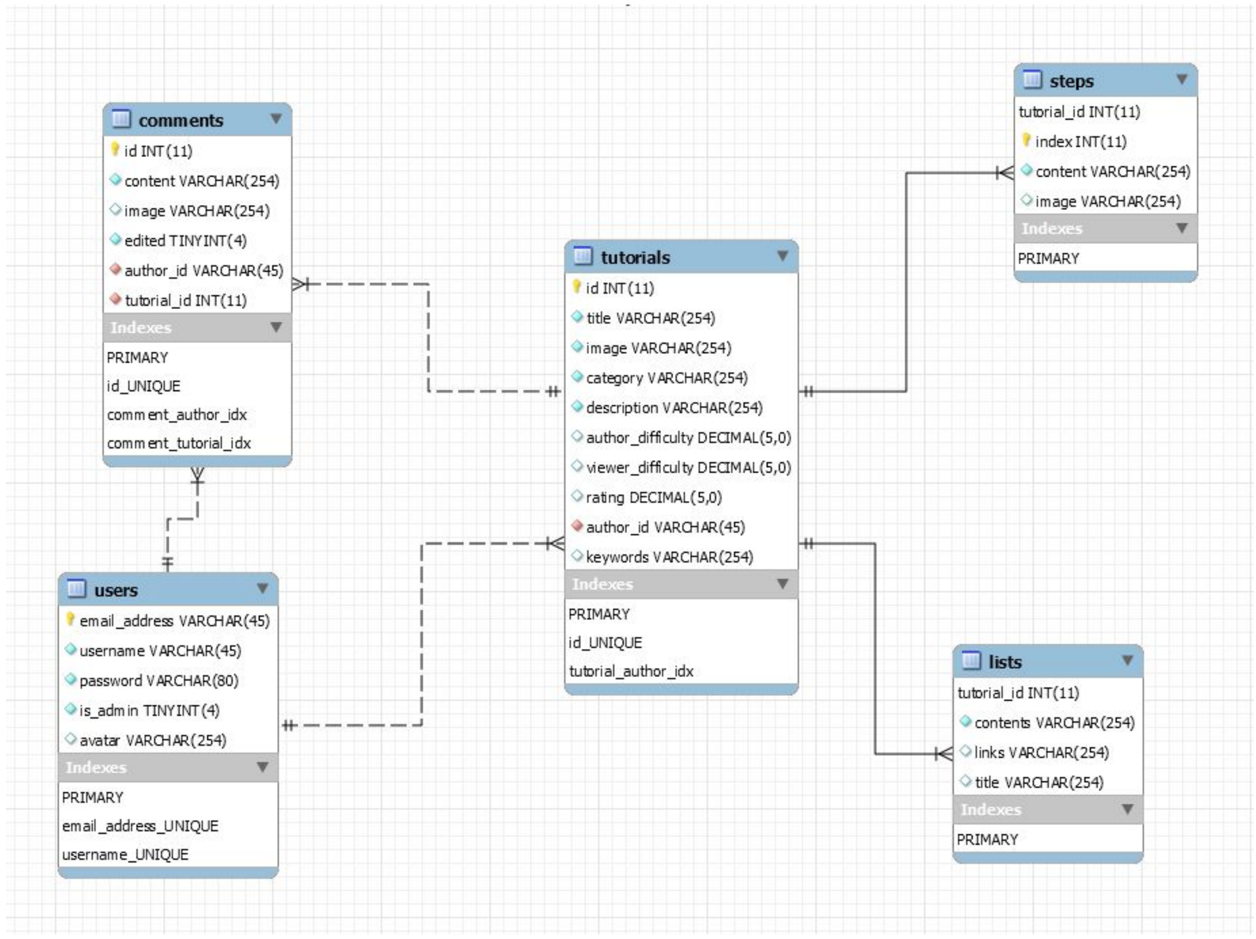
  - contents (VARCHAR)
    The items in the list, separated by commas.

  - links (VARCHAR)
    The links associated with the items in a list, separated by commas.

Entity Relationship Diagram



Database Model



Database Management System

We have decided to use MySQL to create the database because more members of the team have experience with it in comparison to other Database Management Systems.

Media Storage

We have decided to use Imgur to store images uploaded by users. We will use Imgur's api to upload user images to Imgur and will store them in the MySQL database in the form of links. We will allow users to upload JPEG and PNG files under 2MB in size.

Search/Filter Architecture and Implementation

Both registered uses and guest users will be able to search for tutorials by their title, keywords associated with the tutorial, and the user that authored them. Searches can also be filtered by category, author difficulty, viewer difficulty, and rating.

High Level APIs and Main Algorithms

The API created for this application are organized around the REST architecture style created using Flask. The API has easy to navigate URLs and returns JSON-encoded responses using standard HTTP response methods (GET, POST, PUT, and DELETE). Below shows the routes created for the API. Routes with * denote registered or admin user access is required and routes with ** denote admin user access is required.

Admin Routes

- @app.route('/api/user/get', methods=['GET'])
    - get_all_users **
        - Admin access required
        - Gets all users and user information from the database

- @app.route('/api/user/<email_address>', methods=['GET'])
    - get_one_user **
        - Admin access required
        - Gets one user and user information from the database
        - Variables needed
            - User email address

- @app.route('/api/user/<email_address>/promote', methods=['PUT'])
    - promote_user **
        - Admin access required
        - Promotes a registered user to an admin user
        - Variables needed
            - User email address

Registered User/Admin Routes

- @app.route('/api/user/current_user', methods=['GET'])
    - get_current_user *
        - Registered user access required
        - Gets one user and user information from the database

- @app.route('/api/user/current_user', methods=['GET'])
    - create_user
        - Gets current user and current user information from the database

- @app.route('/api/user/<email_address>/demote', methods=['PUT'])
    - demote_user **
        - Admin access required
        - Demotes an admin user to a registered user
        - Variables needed

            - User email address

- @app.route('/api/user/<email_address>', methods=['DELETE'])
    - delete_user **
        - Admin access required
        - Deletes user
        - Variables needed
            - User email address

- @app.route('/api/login')
    - login
        - Registered user and admin user login

- @app.route('/api/tutorial/get', methods=['GET'])
    - get_all_tutorials
        - Gets all tutorials from the database

- @app.route('/api/tutorial/<username>', methods=['GET'])
    - get_all_tutorials_by_user
        - Gets all tutorials by a user from the database
        - Variables needed
            - User username

- @app.route('/api/tutorial/<username>/<tutorial_id>', methods=['GET'])
    - get_one_tutorial
        - Gets a single tutorials by a user from the database
        - Variables needed
            - User's username
            - Tutorial ID

- @app.route('/api/tutorial/create', methods=['POST'])
    - create_tutorial
        - Creates a tutorial

- @app.route('/api/tutorial/<username>/<tutorial_id>', methods=['DELETE'])
    - delete_tutorial
        - Deletes a tutorial by a user
        - Variables needed
            - User's username
            - Tutorial ID

- @app.route('/api/tutorial/<username>/<tutorial_id>/step', methods=['GET'])
    - get_all_steps
        - Gets all steps of a tutorial by a user from the database
        - Variables needed
            - User username
            - Tutorial ID

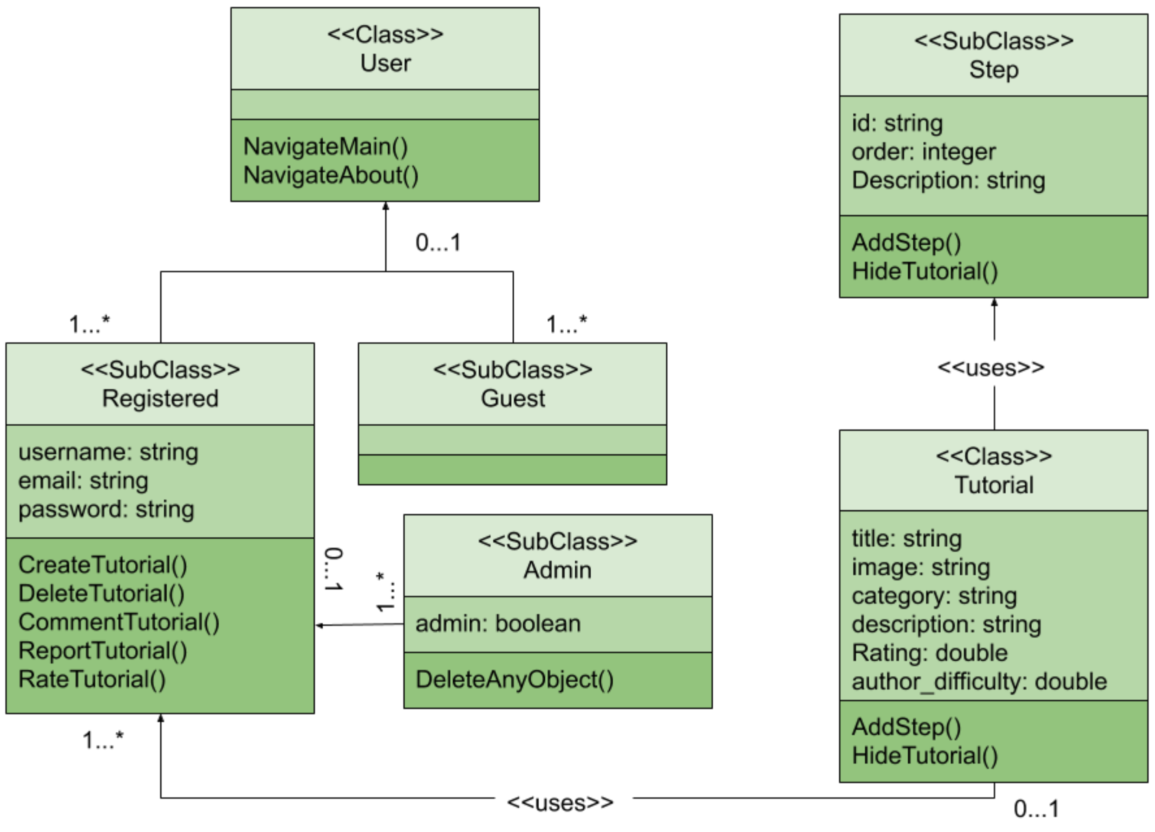
- @app.route('/api/tutorial/<username>/<tutorial_id>/step/<step_index>', methods=['GET'])
    - get_one_step
        - Gets one step of a tutorial by a user from the database
        - Variables needed
            - User username
            - Tutorial ID

            - Step Index

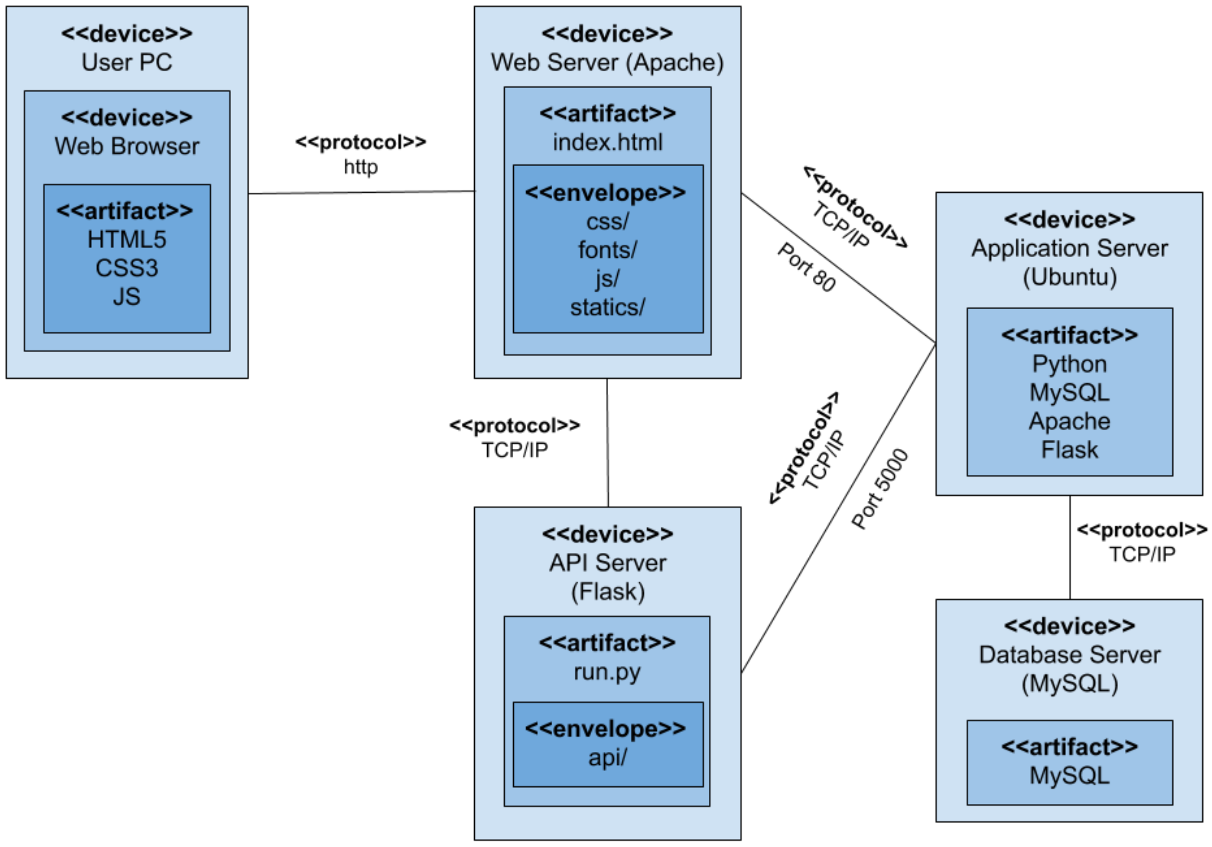
- @app.route('/api/tutorial/<username>/<tutorial_id>/step/create', methods=['POST'])
    - create_tutorial_step *
        - Registered user access required
        - Creates a step for a tutorial by a user
        - Variables needed
            - User’s username
            - Tutorial ID

- @app.route('/api/tutorial/<username>/<tutorial_id>/step/<step_index>', methods=['DELETE'])
    - delete_tutorial_step *
        - Registered user access required
        - Deletes a step for a tutorial by a user
        - Variables needed
            - User username
            - Tutorial ID
            - Step Index

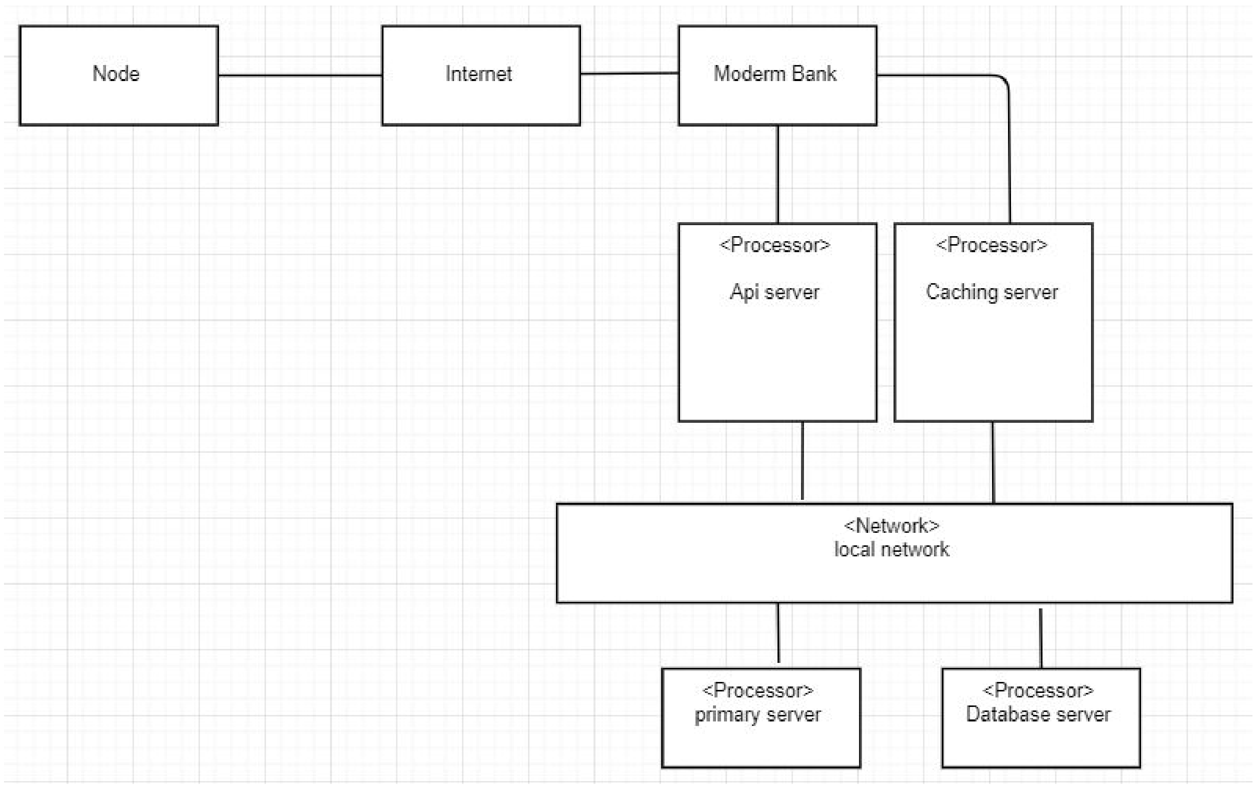
High Level UML Diagrams



High Level Application Network Diagram



Deployment Diagram



Key Risks

Schedule risks: Every teammate has different schedule because of classes or work.

Solution: We have a group meeting every Monday, Wednesday, Friday, and every other Sunday. We usually meet through zoom so that teammates don't have to meet in person if they have classes. For the Sunday meeting, we meet in the library to make the project more efficient. If no one can make it, we will do the thread on slack.

technical risks: Have new technical tools to help make the application.

Solution: Usually when we have a new tools, at least on team members will have general ideas about the new tools. Therefore, team members will look into the resources about that new tools and discuss whether it is worth to make new changes. If we decide to use new technical tools, we will assign another team members to help the members that knows the tools.

Teamwork risks: Team doesn't have enough time to finish all the requirements in priority 1

Solution: In order to ensure the usability and efficiency, We will remove some of the functional requirements in priority 1, and make the other functional requirements in priority 1 perfectly works.

Integration risk: github conflict by integrating the works done by different team members.

Front-end, back-end, and api is not connected properly.

Solution: Before team members do anything, we usually do git pull first so that it can reduce the chance of conflict. However, it still has the chance to get the conflict, so the front-end members will communicate with each other before they push.

Team members will schedule a time to meet in the library and discuss the issues in person which will make the progress faster.

Project Management

As a group, we have talked about using Trello in the future since we believe Trello will help us with easier means to track our timeline. As of Milestone 2, we have heavily used Slack to communicate and keep track of where we are as well as frequent meetings. Our group has 15 minutes SCRUM on every Monday, Wednesday, and Friday and 3-4 hours in-person meeting every other Sunday. Starting from Milestone 3 and onwards, we will try to integrate Trello to manage progress. As for the work distribution of Milestone 2, Myles Pedronan and Antonio Carmona worked on all the backend APIs and database modeling. Jason Wei worked on frontend components to match Eduardo Ramos's UI/UX visions. Phyo Htut worked on integrating frontend and backend to make sure everything required by the prototypes are working seamlessly.