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
11/15/19	Milestone 2	2.0.0
10/17/19	Milestone 2	1.0.0
10/10/19	Milestone 1	2.0.0
10/3/19	Milestone 1	1.0.0

Table of Contents

1.	Data Definitions	3
2.	Functional Requirements	5
3.	UI Mockups and Storyboards	7
4.	High Level Database Architecture and Organization	18
5.	High Level APIs and Main Algorithms	25
6.	High Level UML Diagrams	28
7.	High Level Application Network and Deployment Diagrams	29
8.	Key Risks	31
9.	Project Management	32

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 *UUID*, a *title*, an *image*, a *category*, a *description*, and an *author difficulty*.
    - **Step:** A single step of a ***Tutorial***. A ***Step*** is associated with the *UUID* of a single ***Tutorial***, has an *index*, some *content*, and an *image*.
    - **Comment:** A comment posted on a ***Tutorial*** by a ***Registered User***. A ***Comment*** is associated with the *UUID* of a single ***Tutorial*** and the *email address* of a single ***Registered User***. It has a unique *ID*, some *content*, a *created* timestamp, an edited *timestamp*, an *edited* boolean value, an *image,* and is associated with the *IDs* of the ***Comment*** that it is replying to.
    - **Rating:** A rating for a ***Tutorial*** from a ***Registered User***. A ***Rating*** is associated with the *UUID* of a single ***Tutorial*** and the *email address* of a single ***Registered User***. It has a *rating type* (either “score” for quality or “difficulty” for difficulty) and a *rating* value ranging from 1 to 5.

    - **Item:** A required item for a ***Tutorial***. An ***Item*** is associated with the *UUID* of a single ***Tutorial***. It has a list *index*, a *name*, a *category* (either “tools” or “materials”), and a purchase *link*.

- *Attributes*
    - **Category:** Used to describe the main area of work that a ***Tutorial*** is associated with or to describe the type of an ***Item***. Several examples include Woodworking and Electronics in the case of ***Tutorial***. In the case of ***Item***, the two possible categories include tools and materials.
    - **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***). This is set upon tutorial creation.
        - Viewer Difficulty: The difficulty of a ***Tutorial*** rated by ***Registered Users***. Users can rate the difficulty of existing tutorials.
    - **Score:** Used to represent the quality of a ***Tutorial*** based on feedback from ***Registered Users***. Users can rate the score of existing tutorials.

Functional Requirements

Guest User

Priority 1:

1. Guest users shall be allowed to register for an account for each unique email.
2. Guest users shall be allowed to search for tutorials posted by registered users.
3. Guest users shall be allowed to view tutorials posted by registered users.
4. Guest users shall be allowed to view tutorials based on categories.
5. Guest users shall be allowed to view comments posted by registered users.
6. Guest users shall be allowed to view tutorial ratings.

Priority 2:

1. Guest users shall be allowed to sort with difficulty and rating.

Registered User

Priority 1:

1. Registered users shall be allowed to log out.
2. Registered users shall be allowed to recover their account passwords.
3. Registered users shall be allowed to have a verified code to recover their account passwords
4. Registered users shall be allowed to search for tutorials.
5. Registered users shall be allowed to view tutorials based on categories.
6. Registered users shall be allowed to post tutorials.
7. Registered users shall be allowed to include tools and materials lists with their posted tutorials.
8. Registered users shall be allowed to delete their posted tutorials.
9. Registered users shall be allowed to post comments.
10. Registered users shall be allowed to rate tutorials
11. Registered users shall be allowed to view tutorial ratings.
12. Registered users shall be allowed to view comments posted by registered users.
13. Registered users shall be allowed to reply to comments by other registered users.

Priority 2:

1. Registered users shall be allowed to delete their own posted comments.
2. Registered users shall be allowed to provide difficulties to tutorials posted by other registered users.
3. Registered users shall be allowed to post pictures in their tutorials.
4. Registered users shall be allowed to sort with difficult and rating.
5. Registered users shall be allowed to have a profile page.

Priority 3:

1. Registered users shall be allowed to report the tutorials or comments posted by other registered users.
2. Registered users shall be allowed to add tutorials posted by other registered users to a personal favorites list.
3. Registered users shall be allowed to remove tutorials posted by other registered users from a personal favorites list.

Admin

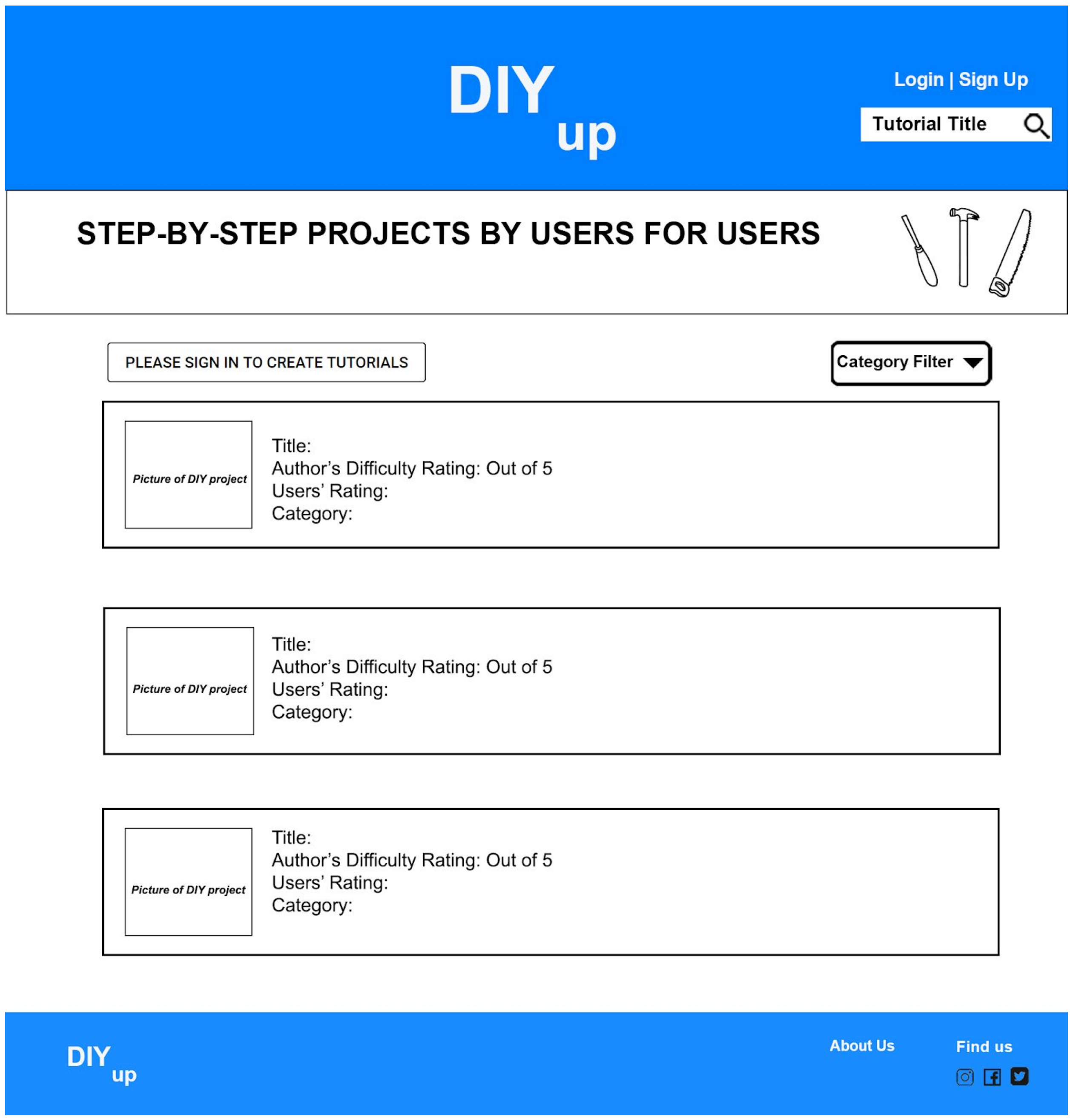
Priority 1:

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

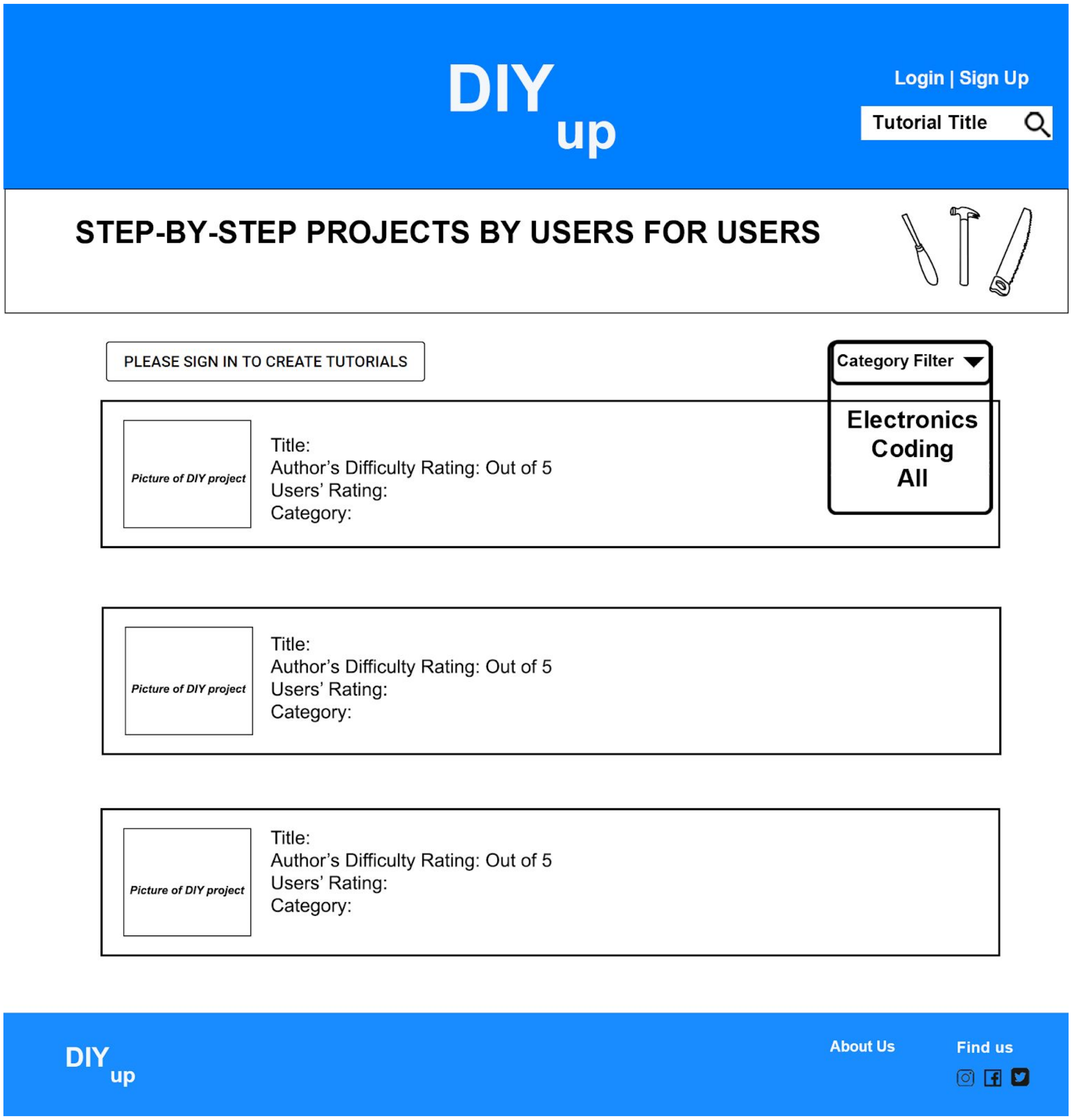
Priority 2:

1. Admins shall be allowed to delete any comments.

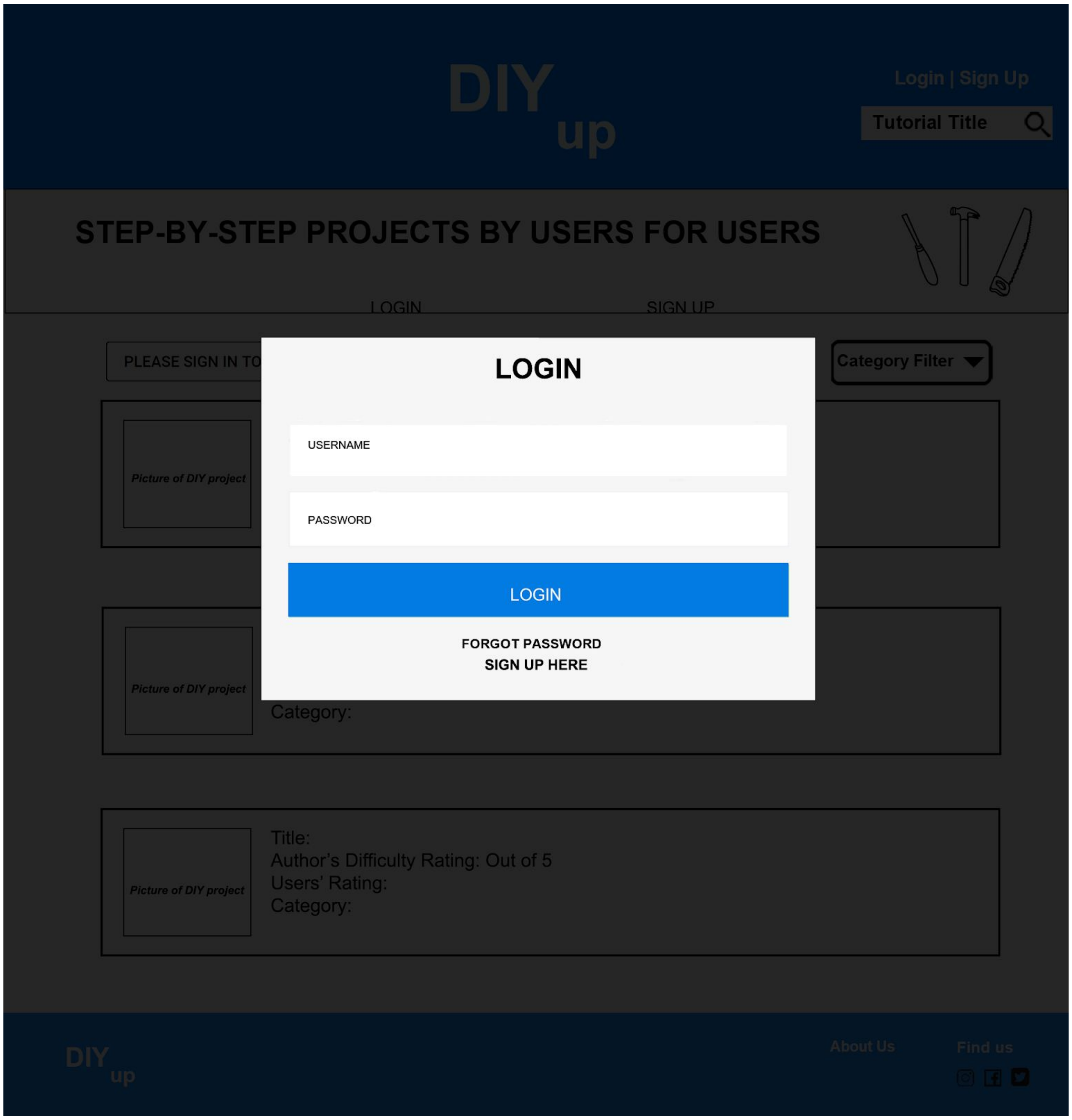
UI Mockups and Storyboards



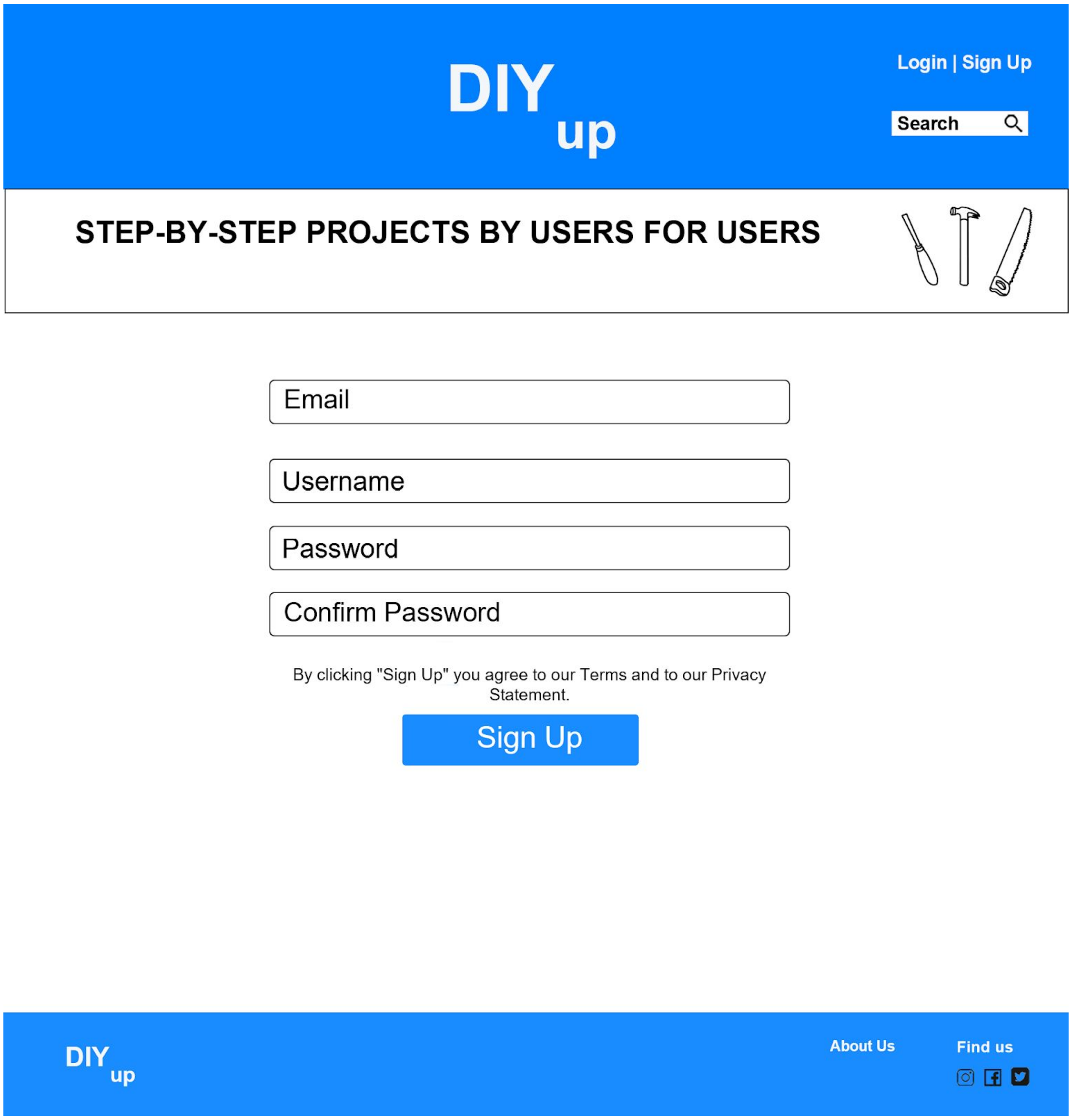
Main homepage for our website mockup where users can view, sort and search for DIY projects and view projects in different categories. Projects are presented with a small picture of the project and a small description of what the project is.



Users can sort through projects on the home page according to their liking of category. This makes it easy for users to be able to quickly find a DIY project that they like.



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.


DIY up

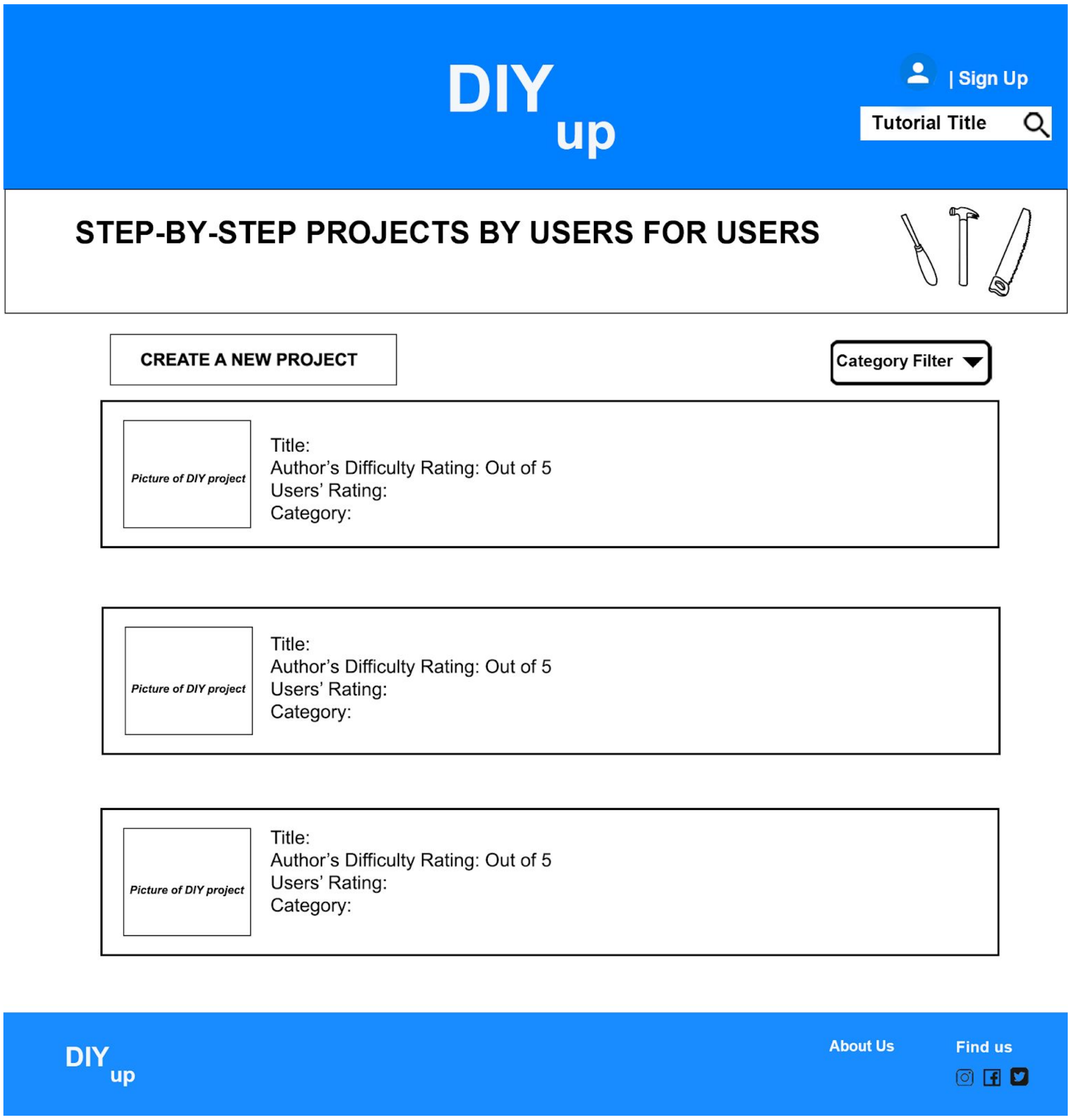
Login | Sign Up

Search

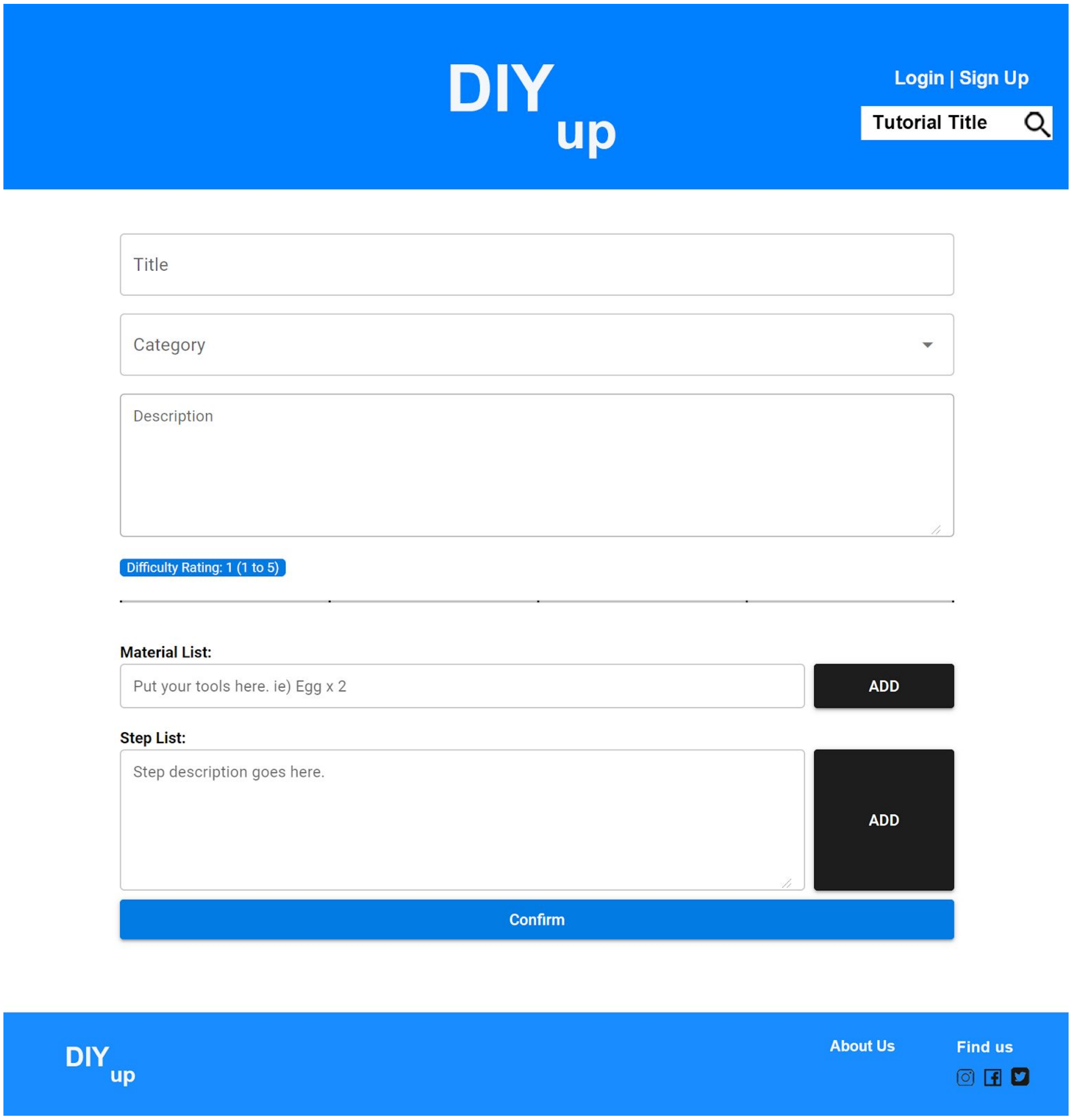
STEP-BY-STEP PROJECTS BY USERS FOR USERS

By clicking "Sign Up" you agree to our Terms and to our Privacy Statement.

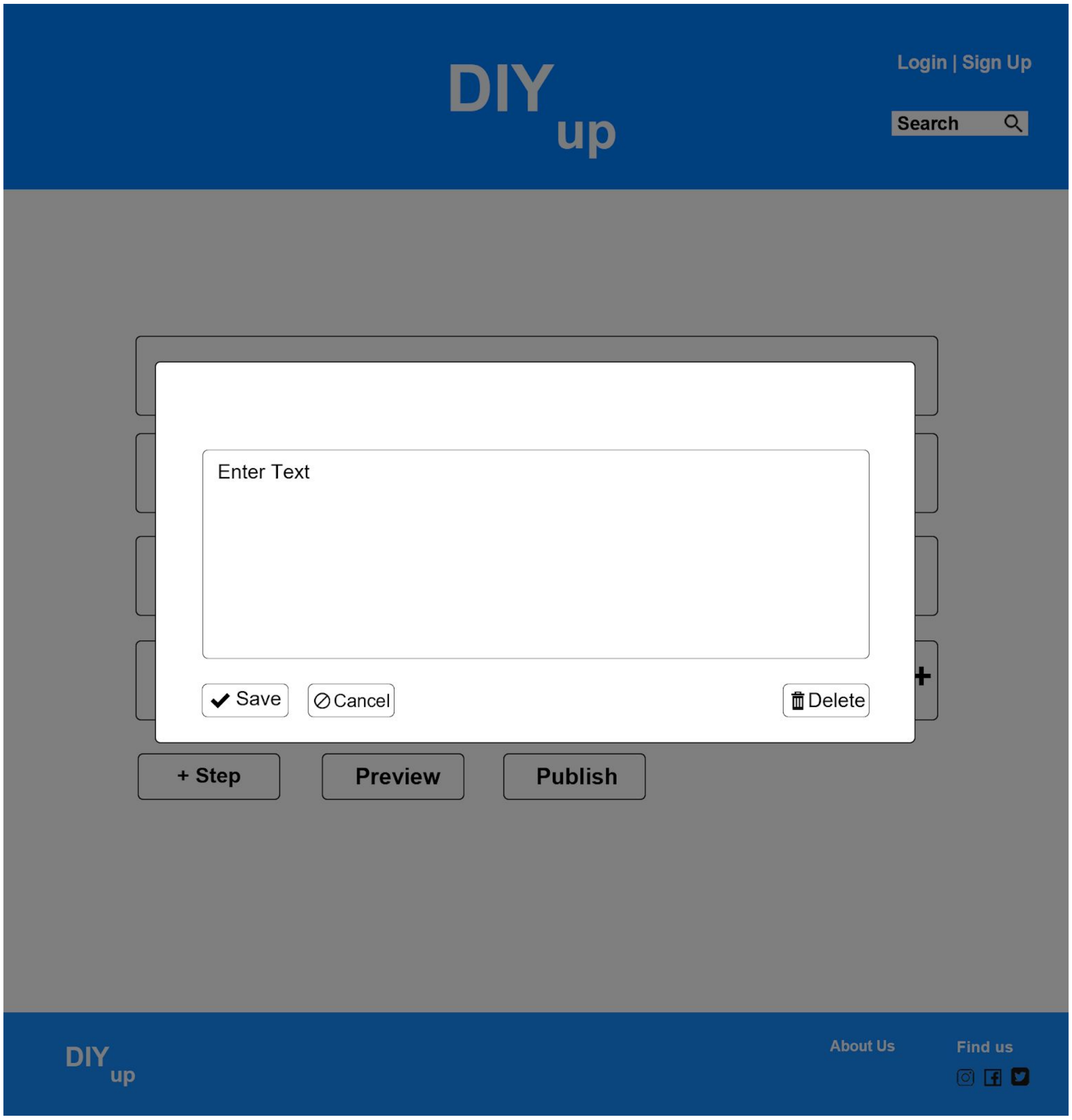
Page where users can sign up for an account



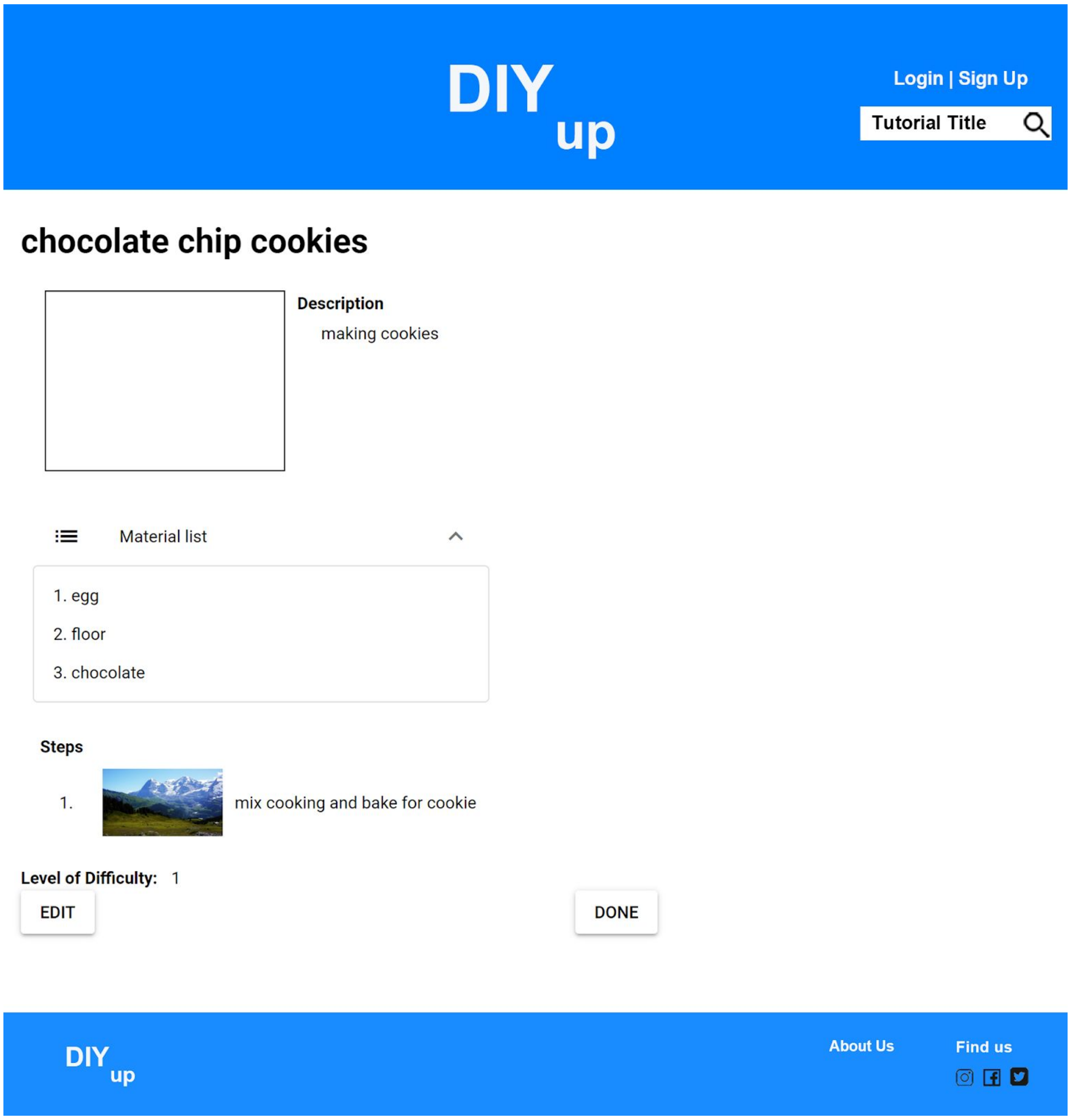
Once logged in users can click the “create a new project” button to create a new post.



Once logged in users can create a project by adding a picture and entering what the materials were used and a description of the project. Users can also add steps along with pictures for each of the project process.



Clicking on the description, materials or steps box will bring up a pop up text box where users can write whatever they like and save it for publishing.



DIY up

Login | Sign Up

Tutorial Title

chocolate chip cookies

Description

making cookies

Material list

1. egg
2. floor
3. chocolate

Steps

1. mix cooking and bake for cookie

Level of Difficulty: 1

EDIT

DONE

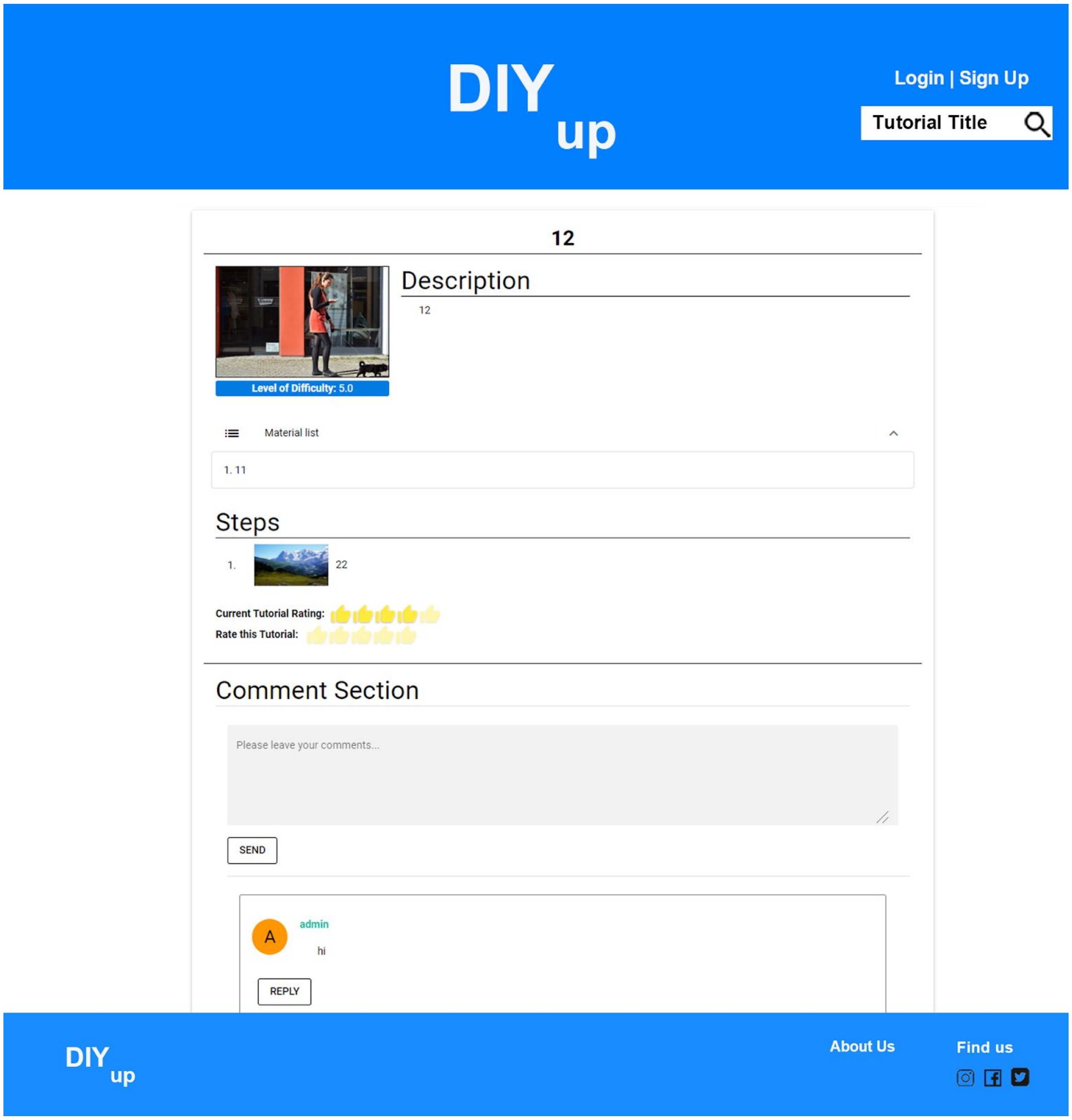
DIY up

About Us

Find us

5

Clicking on the preview will show you a preview page of the post and clicking edit will take you back to edit your post.


DIY up

Login | Sign Up

Tutorial Title

12

Description

12

Level of Difficulty: 5.0

Material list

1. 11

Steps

1. 22

Current Tutorial Rating:

Rate this Tutorial:

Comment Section

Please leave your comments...

SEND

admin

hi

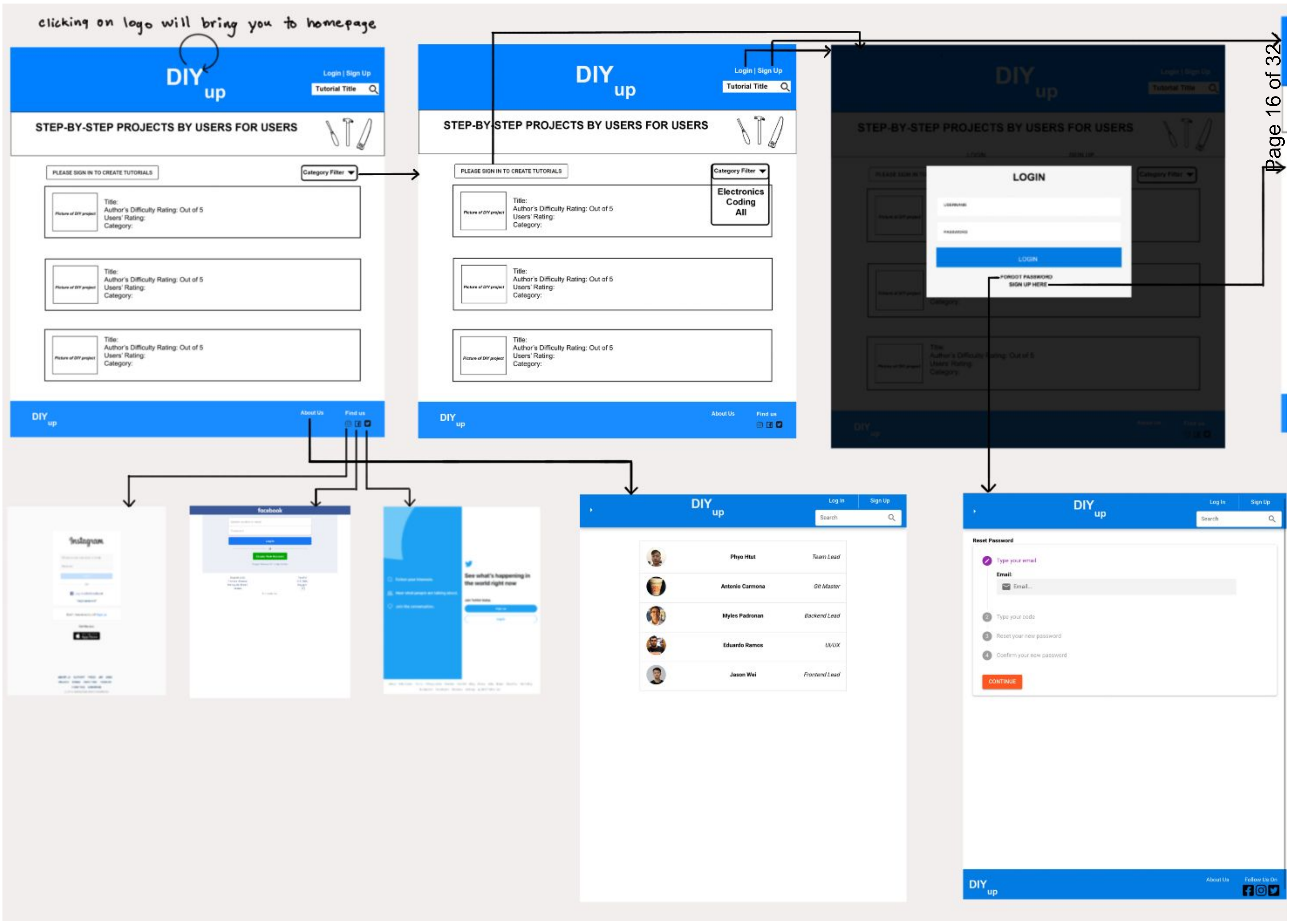
REPLY

DIY up

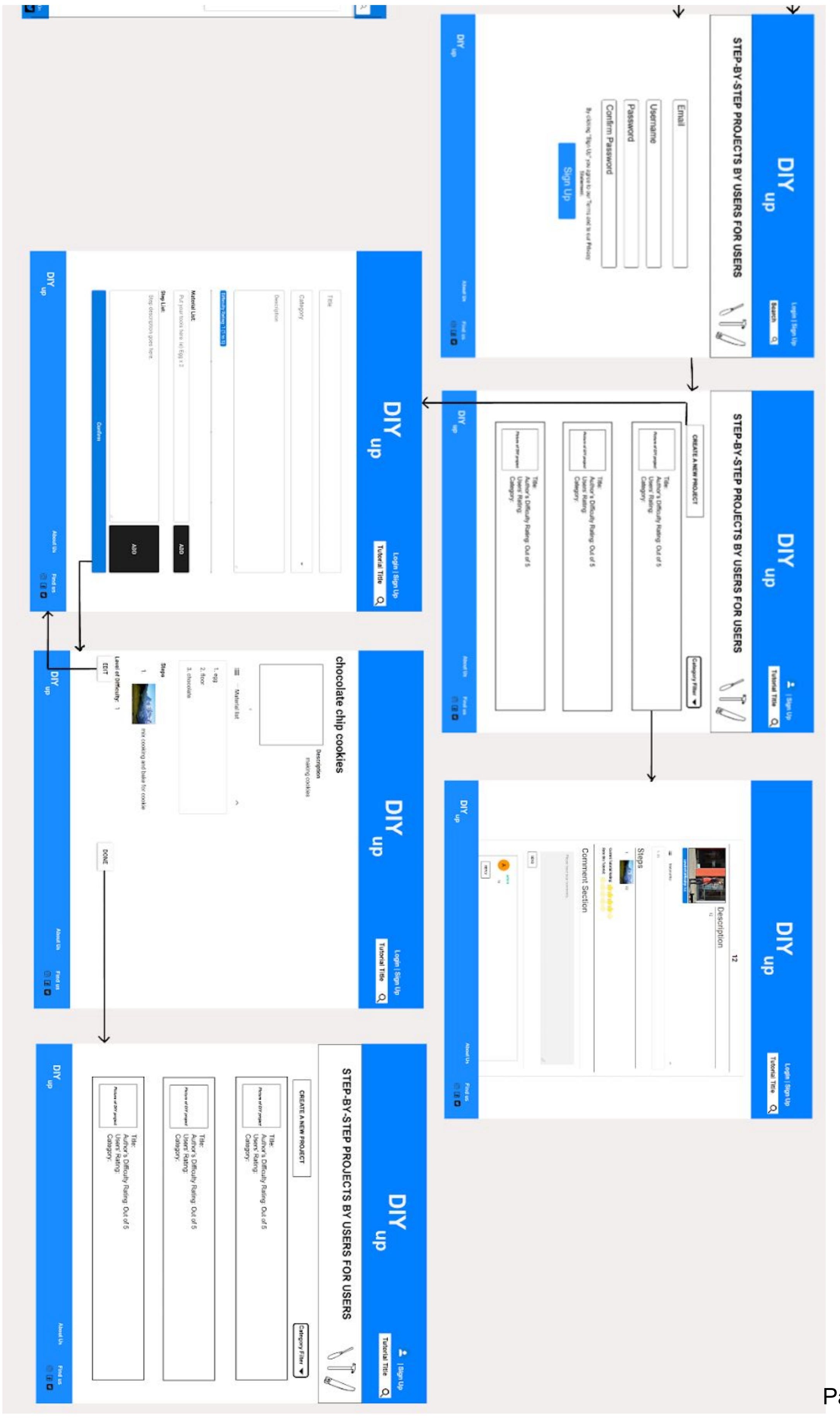
About Us

Find us

Clicking on the publish button will take you to the page where your post has published.



Phyo Htut	Team Lead
Antonio Carmona	Git Master
Myles Padronan	Backend Lead
Eduardo Ramos	UI/UX
Jason Wei	Frontend Lead


DIY up
STEP-BY-STEP PROJECTS BY USERS FOR USERS
Login | Sign Up
Search
Email
Username
Password
Confirm Password
Sign Up
CREATE A NEW PROJECT
Category Filter
Title:
Author's Difficulty Rating: Out of 5
Users' Rating:
Category:
Tutorial Title
Description
Steps
Comment Section
Material List:
Step List:
ADD
Confirm
chocolate chip cookies
EDIT
DONE
About Us
Find us

High Level Database Architecture and Organization

Business Rules

1. Users can post many Tutorials.
2. Users can post many Comments.
3. Users can create many Ratings.
4. Tutorials belong to one User.
5. Tutorials can have many Comments.
6. Tutorials can have many Ratings.
7. Tutorials can have many Steps.
8. Tutorials can have many Items.
9. Tutorials must have at least one Step.
10. Tutorials must have at least one Item.
11. Comments belong to one User.
12. Comments are made on one Tutorial.
13. Ratings must be associated with one User.
14. Ratings must be associated with one Tutorial.
15. Steps are associated with one Tutorial.
16. Items are associated with 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 (TEXT)

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

- uuid (TINYTEXT)
  A user's unique ID. This is used to generate an email verification link for a user's account upon account creation.

- is_verified (TINYINT)
  Indicates whether or not a user has verified their email address. This is never null and has a default value of 0.

- password_reset_code(TINYTEXT)
  A user's password reset code. This is generated and emailed to a user when they make a password reset request.

- Tutorials
  Tutorials that are posted by users.

  - uuid (VARCHAR)
    The unique ID used to identify a tutorial. This is unique, is never null, and is generated as new tutorials are created.

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

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

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

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

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

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

- Comments
  Comments made on Tutorials by Users.

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

- tutorial_uuid(VARCHAR)
  The UUID of the tutorial that a comment was made on. This is never null.

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

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

- created (VARCHAR)
  The timestamp for the time at which a comment was created at. This is never null.

- timestamp (VARCHAR)
  The timestamp for the time at which a comment was last edited at. This is never null.

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

- reply_to (INT)
  The unique ID used to identify the comment that a comment is replying to.

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

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

- Items
  Items required to follow a Tutorial.

  - tutorial_uuid (VARCHAR)
    The UUID of the tutorial that an item is associated with. This is never null.

  - index (TINYINT)
    The index of an item in the list of items for the tutorial that it is associated with.This is never null.

- name (TINYTEXT)
  The name of an item. This is never null.

- category (ENUM)
  The category that an item belongs to (either “tools” or “materials”). This is never null.

- link (TEXT)
  The link that can be visited to purchase an item.

- Ratings
  Ratings made for Tutorials by Users.

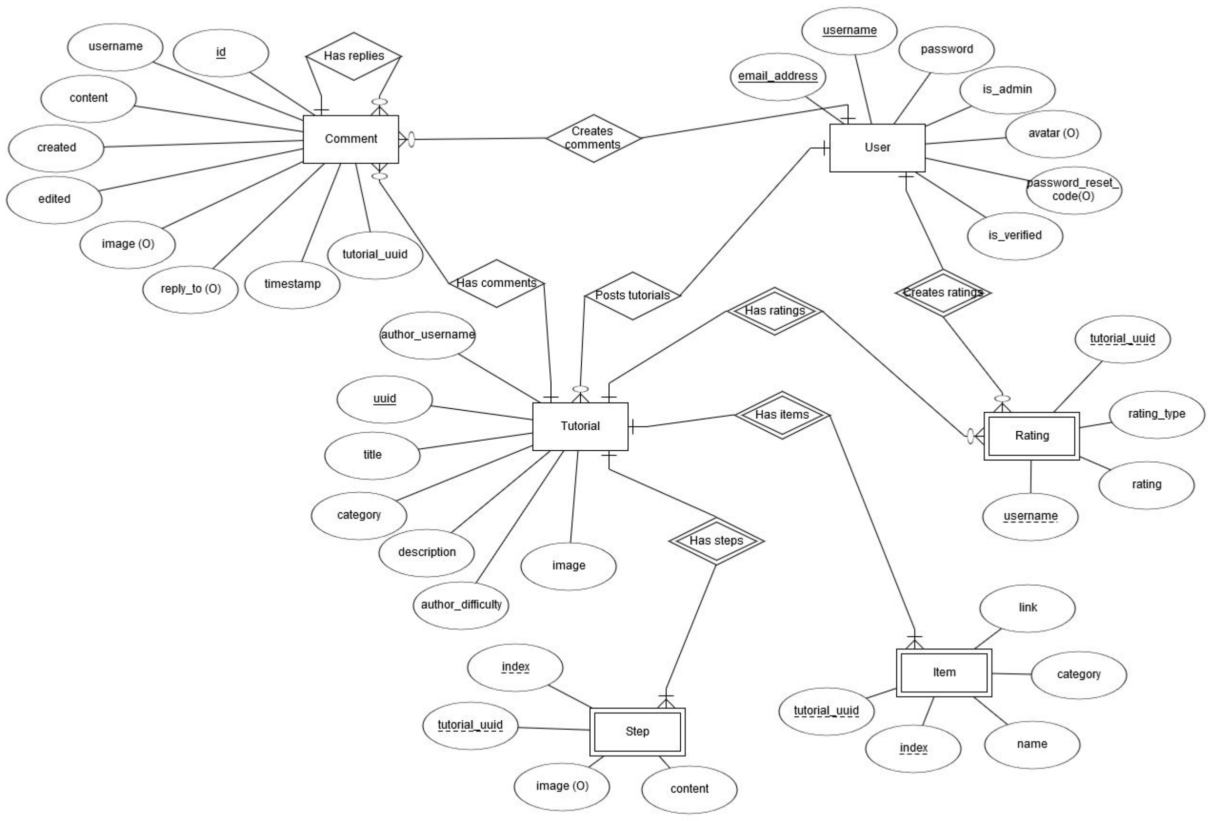
  - tutorial_uuid (VARCHAR)
    The UUID of the tutorial that a rating is associated with. This is never null.

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

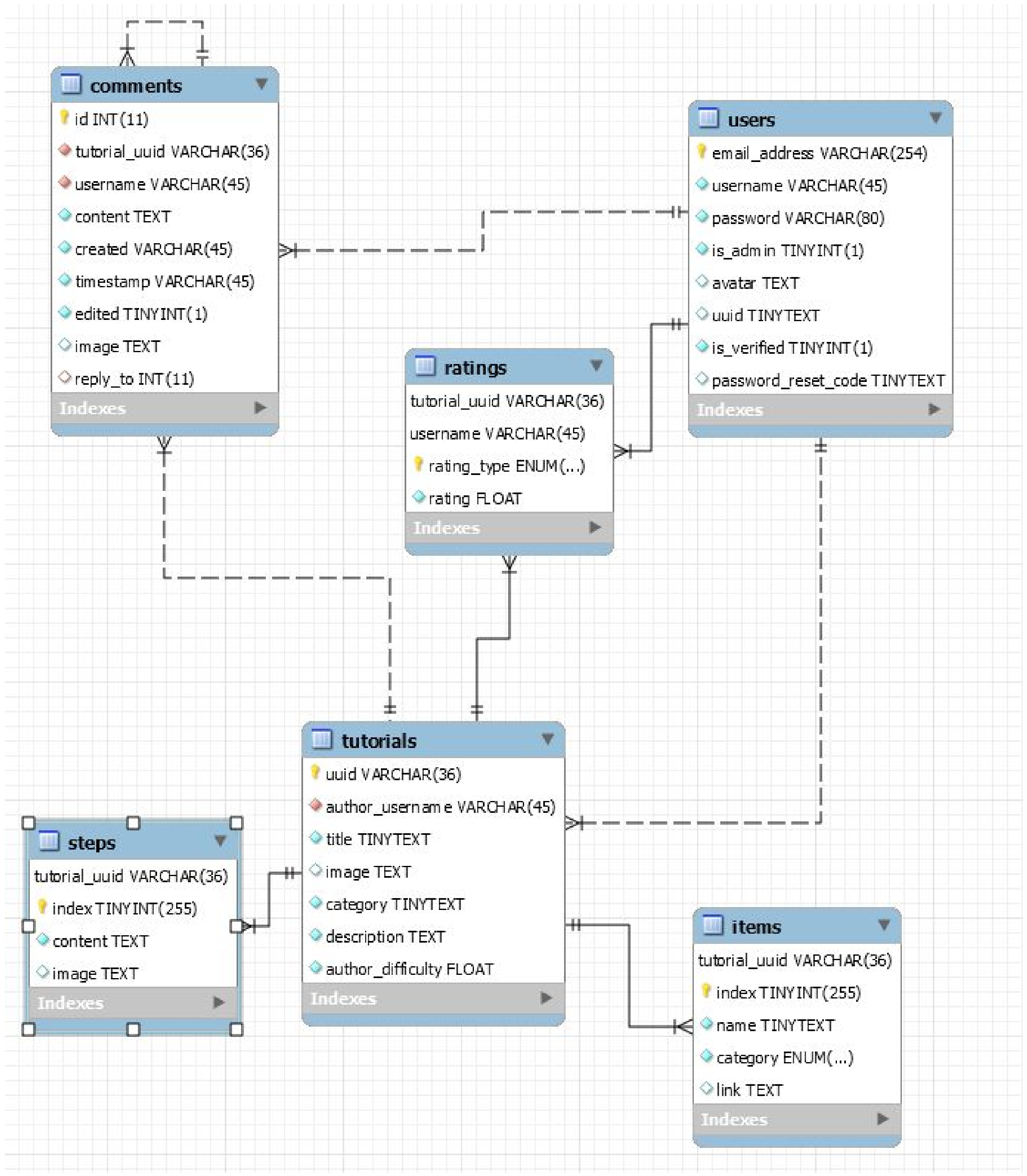
  - rating_type (ENUM)
    The type of rating that the rating is (either “score” or “difficulty”). This is never null.

  - rating (FLOAT)
    The value of the rating. This is never null.

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

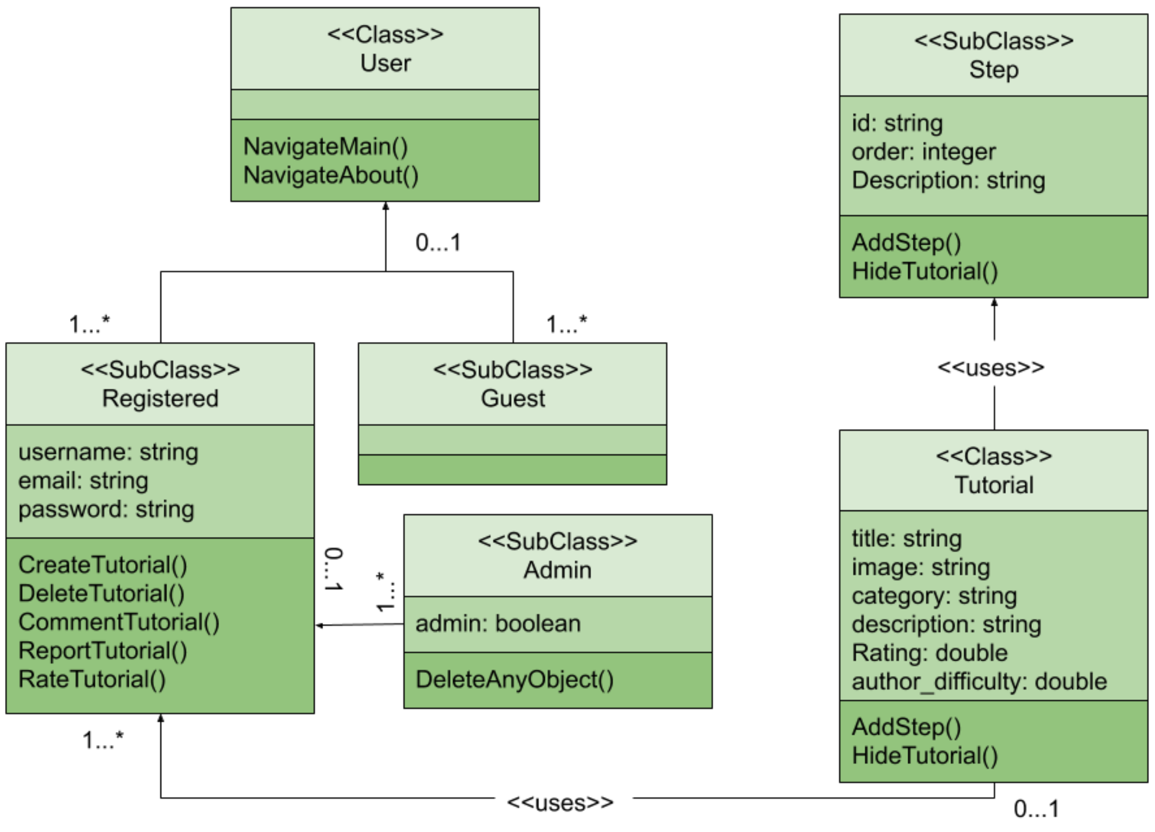
- @app.route('/api/tutorial/<username>/<tutorial_id>/step/<step_index>', methods=['GET'])
    - get_one_step
        - Gets one step of a tutorial by a user from the database
        - Variables needed
            - User username
            - Tutorial ID

        - Step Index

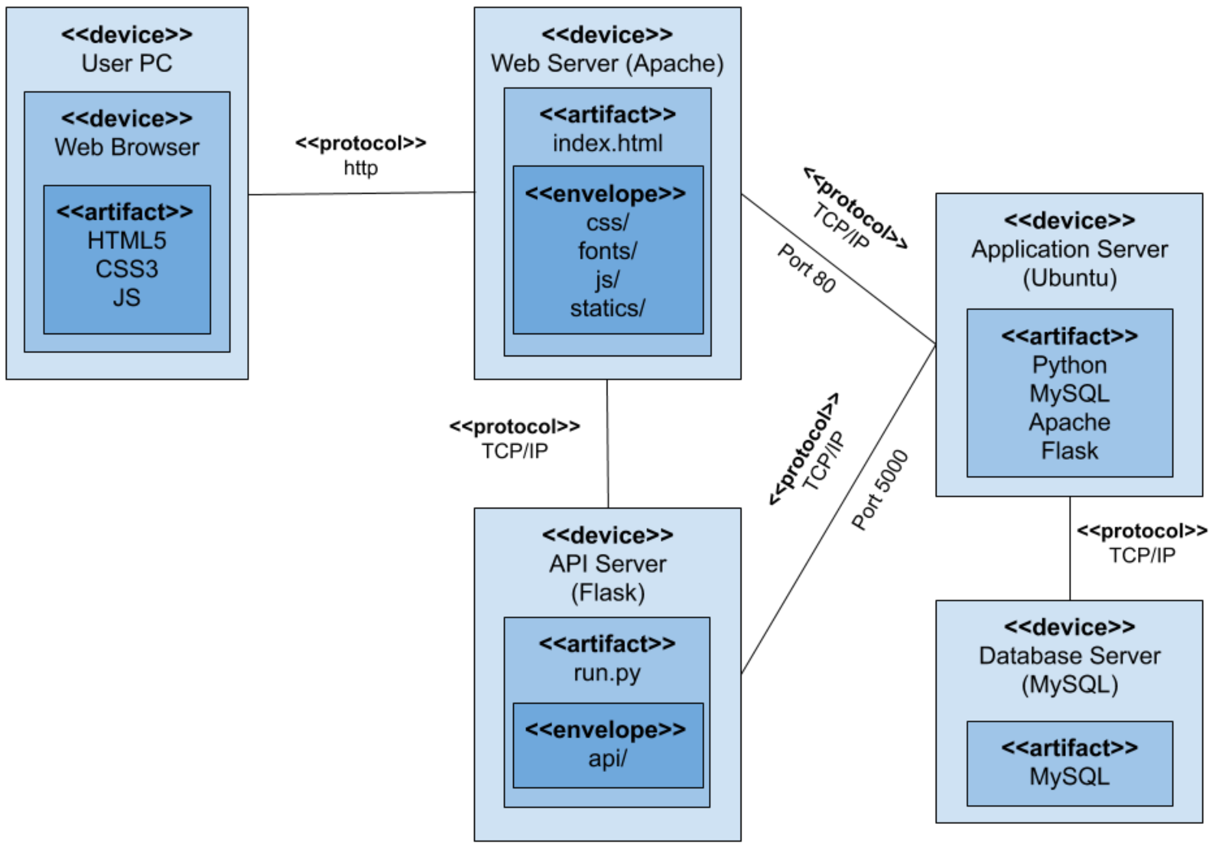
- @app.route('/api/tutorial/<username>/<tutorial_id>/step/create', methods=['POST'])
    - create_tutorial_step *
        - Registered user access required
        - Creates a step for a tutorial by a user
        - Variables needed
            - User's username
            - Tutorial ID

- @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

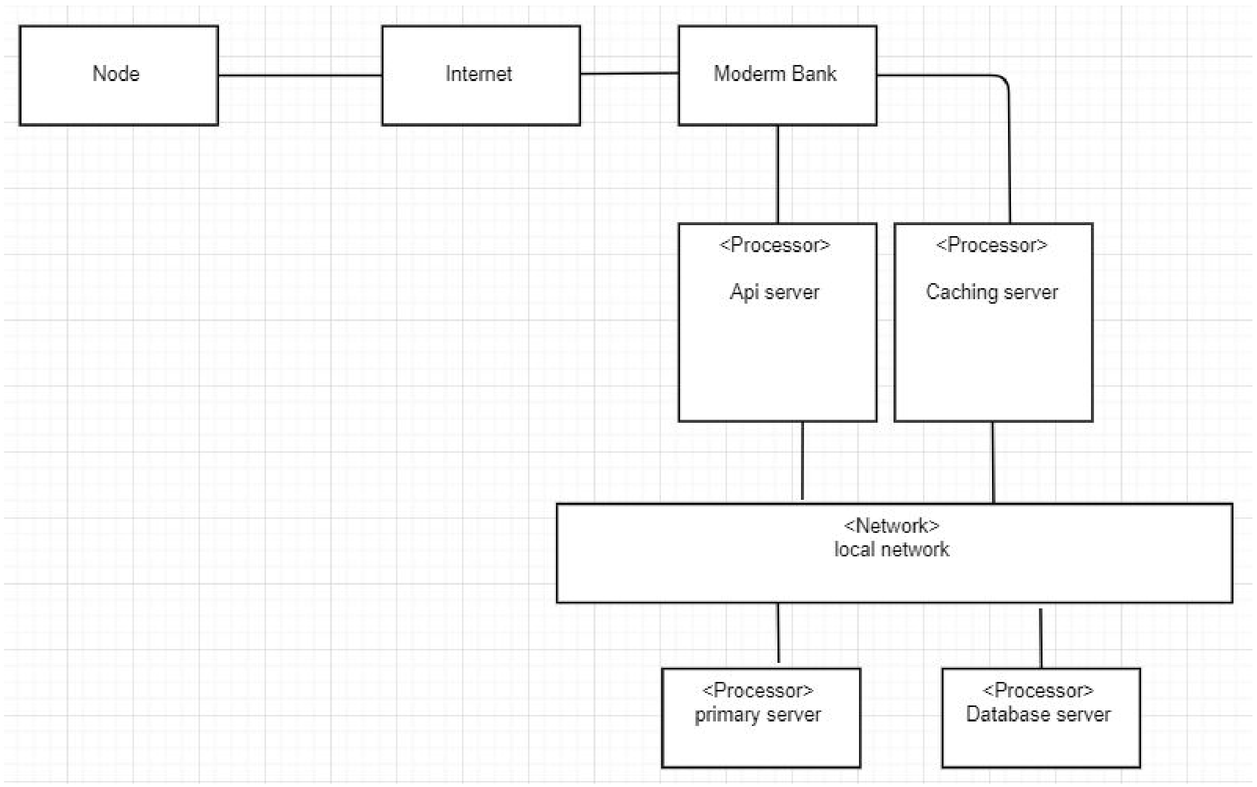
High Level UML Diagrams



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.