Athena

Athena is a Q&A website designed for teachers and students in high school. It is written using the Django web framework on the backend Javascript and its associated libraries on the front end. Django works by dividing the components of a website into separate and individual applications; within every app there is a model, view, and template. This README will walk the reader through the different apps and explain their functionality as determined by the model, view, and template.

Forum

The forum is a the part of the website that contains and displays all the questions and answers.

Models.py:
The main objects in our Forum model are the Question and Answer classes. These two classes contain all the information necessary in the class fields.

class Question:
This class contains a question_text field that is essentially the “title” or initial “blurb” of the question - this field is intended to not be long. The “body” field is the actual in depth explanation of the question. “pub_date” is the time at which the question was posted. Each question also belongs to a unique subject, but these subjects are limited. In the actual UI we have buttons that allow the user to select a subject from a list of choices - we chose this list to consist of Math, Physics, Chemistry, and Biology. Lastly, each question is associated with some User and some Group (will be discussed later). These are treated as Foreign Keys in the database. Both Question and Answer class have class methods meant to abstract a lot of the backend data processing. For example, Question has methods that return all the answers to that particular question that are from a teacher. This way, the Forum view can effectively display all the teacher answers first, sorted by upvote, and then the student answers below. This supports the idea that teacher answers are more credible and thus should be the very first answers to a question that are seen.

class Answer:
This class is very similar to the Question class in that it also contains a text field and foreign keys to a User. An answer also foreign keys to a Question object, since a Question can contain many Answers. An additional field that the Answer class has that Question does not is an “upvotes” field. This integer signifies the number of upvotes a particular answer has received - it can be either positive or negative. The Answer class contains methods that alter the upvotes on that particular question. As we will see in the User class, Users have a many-to-many relationship with Answers, and thus these methods initially detect whether the up-voter has already upvoted the answer, and if so, prevents any action from occurring.

Views.py:
The forum views are responsible for interpreting a user request to a particular endpoint and returning the appropriate HTML that is the UI.

def index():
This function takes every question in the database that has been asked in the universal feed, and sends it via the context map to the template which is then rendered on a browser. This is the “homepage” for the forum.

def filter_index():
Used to take questions from the database that are associated with certain subjects; for example, when the user filters the global feed to all Biology questions, this is the function that extracts all the Biology questions from the DB.

def detail():
Displays a question in detail so that it can be viewed outside the list of questions in the feed.

def upvote():
This is the function AJAX methods on the frontend will use to communicate in order to increase the counter for votes on the UI. Essentially calls the upvote() method on an Answer, and returns to the UI the number of upvotes an Answer now has.

def downvote():
Similar to upvote.

def answer():
This function is called whenever a answer is typed on the UI and the “Enter” button is pressed. The function goes through the process of generating an Answer object and subsequently saving it after all parameters have been validated.

def add_question():
Uses a try-except clause to determine whether a question’s fields are valid (contains user, title, body, subject, etc) and if so then then it is saved to the database and can be rendered into the forum by index().

Templates:
The templates involved in the Forum app are all seeking to display the questions and answers in a reasonable fashion.

base.html:

This is the base for most pages of the website. It consists of the background and nav bar, both of which stay constant for every page but the login and signup pages. It also imports all css, javascript, jquery, etc, and has a few custom javascript functions for the voting, answering, and affixing of objects.

forum_base.html:
This is the base html for a forum, which means that every time a user is viewing a list of questions (forum) then it will contain the same layout. This layout consists of a box that indicates the UserProfile in the top left corner, along with school, teacher status, username, profile image, etc. On the right hand side there is a space reserved for the actual questions and question box. Bootstrap is used to have the Userprofile box in the top left float down as a user scrolls through the questions.

index.html
This template is given a context map that contains all subjects a Question can be, as well as a list of questions. The template allows for a User to ask a question and subsequently submit it into the database. The list of questions are also iterated over such that each new question is displayed right below the other. Javascript / Bootstrap is invoked when a user clicks on a question - the answers to that question are displayed right below.

detail.html
This is used only when a user seeks to view a question independent of other questions (i.e., outside the forum). It is nothing complicated; it merely displays the question and the list of answers below it, without having to click for the answers.

Static:
Django provides easy referencing for “static” files, these generally include any images, css, javascript, etc. that are associated with the app. This folder contains the files for all apps related to Athena.

Forum:

bootstrap, jquery:
These are downloaded jquery and bootstrap to reference (downloaded and added to make them editable rather than the standard files available online)

my-java.js:
This contains the custom written AJAX for sending requests related to voting and answering, and the json data it sends back to the template.

flag/orange-square:
Static image files used as defaults for not logged in users.

style.css:
Custom css for the templates relating to base.html in the forum folder and all templates which extend it.

User:
pic1-7.jpg:
These are the pictures which make up the changing background of the login and signup pages.

user-style.css:
style for the pages which extend login_base.html in the user class. Mostly this is concerned with css for overlaying forms on the changing background without having the forms fading as well and customizing it for various widths (for mobile etc.)

Groups

models.py

class Group: models the “study group” (joined with users in the UserProfile model)

group_name - name of the group
create_date - date of group creation
creator_username - username of user who created the group
topic - group topic (more of a free description)
group_type - (one of 3 types of groups)

views.py

def index(): The homepage for the groups app (that can be found in the navigation bar)
def detail(): detail page for each group (requires an additional group id parameter)
def new(): the page containing the form to create a new page
def add_group(): the function (doesn’t have a corresponding page) that handles the POST request for creating a new form

urls.py

routes the html request to the corresponding controller function.

<u>Templates</u>

index.html

has a list of the groups that the logged-in user is a part of

detail.html

detail page for a group that contains information about the group members and the questions asked inside the group’s forum

new.html

contains the form to create a new group and allows a user to add a group name, topic, type of group, and select any and all group members.

Users

The users app contains all the classes and pages associated with users including the Django-bundled user class, an extended user profile class, logging in, registration, logging out, and editing a user's profile.

Models.py
class UserProfile:
Implements a UserProfile class that extends the Django-provided User class. Includes fields for picture, school, first and last name, whether the user is a student or teacher, lists of upvoted answers and downvoted answers. Includes several definitions that restricts a user to only giving either one upvote or one downvote per answer.

Views.py
def register():
Takes user-inputted values from UserForm and UserProfileForm and adds them to a new user-userprofile pair. Once values are inputted, authenticates the given profile and logs the user in so that the user is redirected to the homepage already logged in.

def user_login():
Uses the provided credentials from the request and attempts to log in the user.

def user_profile():
Brings up either the currently logged in profile (if id = 0), or someone else's profile (id != 0). Brings up a sorted list of questiosn that the user has answered.

def edit_profile():
Brings up the EditProfileForm and passes in the current profile as a parameter to the form so that the fields can be initialized with the current profile values.

def user_logout():
Logs out the current user.

Forms.py
This file contains all the form classes specific to registration and editing a profile.

class UserForm:
Leverages the standard Django User class and allows a user to input values for username, email, and password.

class UserProfileForm:
Implements a profile form class and allows a user to input values into the various profile fields.

class EditProfileForm:
Implements an edit profile class that allows the user to change certain values in his or her profile. Specifically, a user can change their first name, last name, picture, or school through this class. Fields in the form are initialized with the current profile values, which is done by extending the __init__ method for the class and passing in the current profile values as parameters to the edit profile form class. When the user submits the form, the values in these fields are written as the new values for the given profile.

Contact

Mark Valentine markolas@stanford.edu
Camilo Arevalo carevalo@stanford.edu
Divya Garg divyag@stanford.edu
Alessandro Sanchez sanchez7@stanford.edu