2017-2018 Fall Semester
CS 353 Database Systems

CASESWITCHERS

PROJECT PROPOSAL

Group 2:

Fırat SİVRİKAYA
Eren BİLALOĞLU
Gülce KARAÇAL
Afra DÖMEKE

Section 1 - Group 2

Available online : https://djcedrics.github.io/CaseSwitchers/

CONTENTS

1. Introduction

CaseSwitchers is a collaborative hypertext dictionary where people can find contents or definitions of computer science and informatics related topics. Unlike traditional dictionaries where users cannot write,edit or comment, users have a control on this web-based dictionary. They can write posts, rate and comment on them. People can find entries about almost every computer programming topics and concepts that are written by the registered users.

Simply it can be considered as a user based dictionary that provides exchange of information among computer engineers or software developers and keeps its users up-to-date.

2. Description

Users of CaseSwitchers can sign up and act as the authors of a dictionary. They are able to start new topics, to post under the topics, to upvote or downvote the posts of other users, to make comment to other posts and to send messages to each other. A post that is committed under a topic is called “entry” and entries are the basic elements of the CaseSwitchers.

Entries consist of only sentences (no pictures, videos etc.). There is no strict limitation for entry writing format. Everyone can write however they like (in a formal manner or in a daily speaking manner), what matters is content. As far as entries are useful for someone looking for decent information (it can be an explanation, an experience of any other user, or opinions of other users etc.) about a computer science related topic, users are free to post them. As the only constraint for the writing format, users have to write their entries in the form of definition.

An entry example for the topic named Computer Science:

“A scientific discipline where mathematics, logic, creativity, eye disorders and sleep deprivation meet.”

As it can be seen from the example, entries do not have to be formally written strict mannered sentences. They can be humorous, but they have to be written in a definition format and they need to convey some message.

Users can communicate each other via direct message or commenting to each other’s entries. According to their ranks, which are formed by karma of the votes they get for their entries, users can be at different levels. These levels are displayed near users’ nicknames so that everyone can see whether the user is proficient or a rookie.

Topics are listed on the left side of the screen in a frame. Topics to which users recently posted an entry are sorted above the other topics. In other words, topics are listed according to their current popularities so that users or visitors can see what is hot at the moment.

CaseSwitchers serves as a source for information and it grows with its users since users create the content. Users can be from anywhere around the world and they can share their knowledge with anyone. The more they post entries, the more CaseSwitchers grows. This makes CaseSwitchers an up-to-date dynamic information platform.

3. Significance of Database Systems

As mentioned in the description section, CaseSwitchers is a multi-user web platform that enables users to perform many actions such as posting entries under a category, following other users, favoriting entries, commenting on other posts etc. Entries, posts, users, comments, categories are all accepted as entities in our context. By the project’s nature, the platform should support multiple users that can write multiple entries, comments under multiple categories, and follow other users and be able to send them direct messages. This requirement brings the need of

storing and organizing data in an efficient way. Storing and organizing data in text files as we did in CS102 course is not sufficient for this platform. The data will be in huge amounts and in a complex structure, so it is nearly impossible to maintain the data in such simple text files. Thus, an another system with more capabilities should be used in our context. Database, a well known data storage system, satisfies the purposes of the system with its functionalities. Database system is going to be used in our project for the following reasons:

- The content of a database is easy to manage and maintain with the usage of queries. When the user wants to create a new entry, or update/delete his/her own entry, a simple query will be executed at the background and the data related to the entry will be manipulated accordingly in the database.
- The data stored in a database can be accessed and manipulated efficiently and quickly. When the user wants to display the comments under an entry, the comments should be loaded nearly instantly. Searching operation should also working fast, so the time elapsed for listing the search results of an entry search query should be minimum enough.
- Database supports the establishment of relations between entities, by doing so, stores the data in a more connected and organized way. The platform requires the usage of multiple entries with multiple relations. For instance, comments are always related to posts, since a comment which is not attached to a post does not exist. Moreover, a comment can only have one post related, whereas a post can have multiple comments related. We can implement such relations by using databases.
- Database system knows how to handle huge amounts of data and how to scale itself accordingly. The platform will support the registration of new users and creation of new posts, without an upper limit other than hardware constraints (disk space). Thus, our data is very likely to scale up and the growing data should be handled perfectly to avoid any problems to occur.
- Database can easily detect and avoid recurrences in data stored. A simple example can be given for this case. If a user registers for a new account, he/she should not select a username that is already in use. Otherwise, the

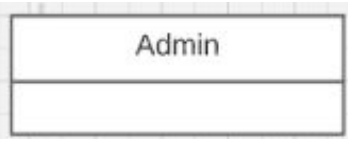
login operation would fail since the system would try to match the entered username with multiple passwords. Similarly, an email address should also be attached to only one user, otherwise, the same issue would arise. Thus, in this instance, usernames and email addresses should be unique and the database system is already able to maintain the uniqueness.

4. Functional Requirements

4.1. User

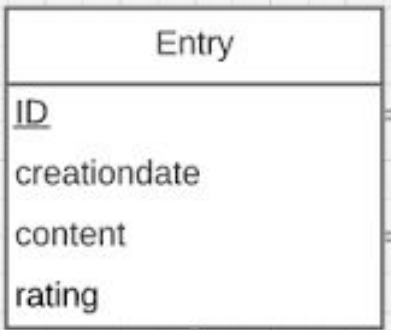
- The user should be able to write posts.
- The user should be able to comment on posts.
- The user should be able to rate posts.
- The user should be able to label entries as favorites.
- The user should be able to follow other users.
- The user should be able to send direct messages to other users.
- The user should be able to modify its posts.
- The user should be able to delete its posts.
- The user should be able to change its profile information.

4.2. Admin



- Admin should be able to do anything that a user can.
- Admin should be able to delete an entry.
- Admin should be able to ban users.
- Admin should be able to update users' personal information.

4.3. Entry



- An entry can be either a post or a comment.
- An entry can be rated by users.

4.4. Post

- A post can be upvoted or downvoted by users.
- A post can be added to favorites by users.
- A post can have multiple comments.
- A post can have multiple categories.

4.5. Comment

- A comment can be added on a post.
- A comment can be rated by users.
- A comment can be added to favorites by users.

4.6. Category

- A category can have multiple posts.

5. Non- Functional Requirements

- **Usability:** Since CaseSwitchers is a dictionary in which users play active roles, the system should not have complicated functionalities. Instead, we will focus on the ways which make our dictionary easy to use. That is, an

user-friendly interface will be one of our concerns while working on this project. In this way, users can spend their time enjoying posts rather than struggling to learn how to use the dictionary.

- **Extensibility:** Adding new features to the dictionary to maintain the excitement and interest of the user is significant. Hence, our system design will provide an environment in which adding new functionalities and entities to the existing system is possible.

- **Portability:** Due to the fact that, our dictionary will be implemented in a web-based environment, it should run on many browsers and mobile platforms. This feature will make CaseSwitchers portable on any device, namely computers, mobile phones and tablets.

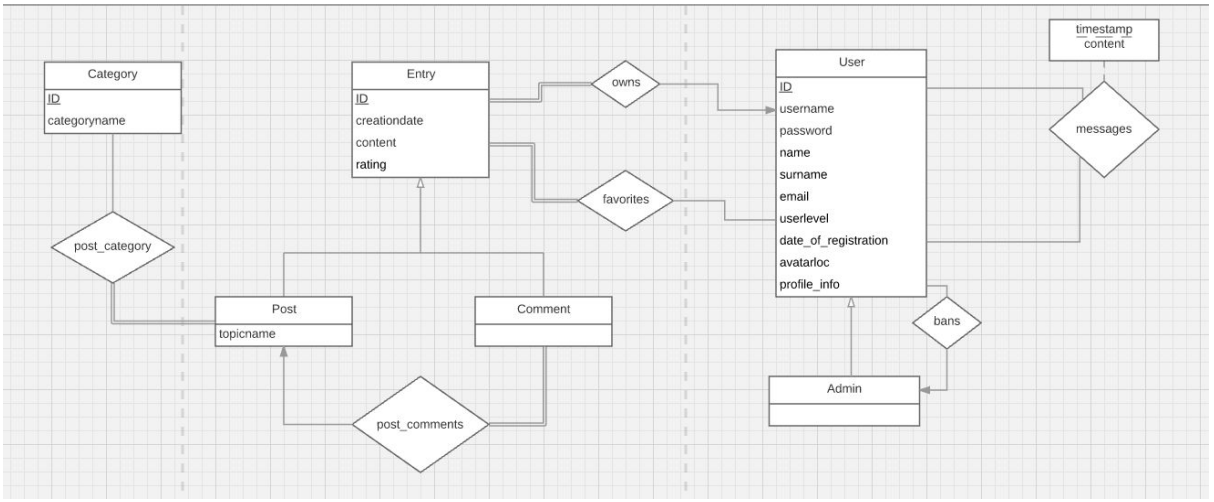
- **Performance:** It is crucial how quickly the system reacts to input from users. Our dictionary will provide an immediate response to users' actions in order to maintain users' interests.

6. Limitations

Users can read entries, write, rate, comment on posts and label entries as favorites. They can follow other users and send messages to each other. Also, they can modify or delete their own posts but cannot edit others' posts which are not posted by them. They can update their own profile informations.

Admins can edit or delete any articles and comments unlike users. Also, they can delete or ban the users and update all users' personal information. They have the ultimate control over the system.

7. E/R Diagram



The diagram above sums up the relations between the entities in our system. In this section, we are going to give more details about the relations.

7.1. post_category

Posts have to belong to categories in the system. Post_category specifies the many-to-many relationship between Category and Post entities. Each category can have multiple posts, likewise each post can be categorized under multiple categories. There is also a total participation of posts in this relationship. A post must have at least one category, otherwise it cannot be created.

7.2. post_comments

Comments are the responds made to the posts by users. Post_comments relationship clarifies the posts which have comments. There is a many-to-one relationship between Comment and Post entities; a post can have multiple comments; however, a comment can be related to only one post. Furthermore, all of the comments must participate in this relationship with a

total participation for the following reason. A comment cannot exist without a post attached, as comments can be published only under posts.

7.3. owns

By the core requirement of the system, users are able to create entries. To establish the relationship between User and Entry entities, owns relationship is created. There is a many-to-one relationship in this context, since an entry can belong to only one user and a user can create multiple entries. There is a total participation of entries in this relationship for a trivial reason. An entry must owned by a user, otherwise it cannot exist. This total participation also holds in favorites relationship.

7.4. favorites

Users can have favorite entries in our context. This brings the need of storing the favorite posts of each user. Favorites relationship tracks the posts that are added to favorites by the users. There is a many-to-many relationship between Entry and User entities; users can add multiple entries to favorites and an entry can be added to favorites by multiple users.

7.5. bans

Admins are able to ban users in our context. To store the banned users, we specified a relationship called bans. We could instead have added the ban status as an attribute to User entity; however, we would not be able to keep the track of banned user together with the admin who banned the user in this case. Bans relationship makes the tracking of both user and punisher admin possible. The relationship is many-to-many, which means that every user can send messages to another multiple users.

7.6. messages

By system requirement, users should be able to send messages to each other. This requirement specifies a relationship between users. Messages relationship will keep the track of which user has sent a message and to whom the user sent the message, together with the timestamp and message content. By doing so, the message receiver user will be able to read the message content, sender name, and timestamp of the message.