Repliance Functional Documentation

April 10 2015

James Desjardin | Matt Fischer | Rohan Kapoor
Kavya Krishna | Colin McGinnis | Kalina Nedkova

The following explains the technical design of our application broken down by each different view, and how those views' elements will operate with the application's back end and database. We have also included a section describing the main navigation bar, available from every view once logged in, and its function throughout the application.

Login & Signup View

This is the default view which a user encounters when they are not logged into Repliance. The view will contain a box centered on the homepage which includes a login component and a sign up component. Each component will exist within its own div element, login on top, center aligned with a form in each to collect the user data. By default, the signup component will be hidden, with a "Sign Me Up" button visible underneath the login prompt. The application will use jquery to expand the signup link and collapse it when this button is clicked. The login form will ask for a username and password while the signup form will ask for a username, password, optional first and last name, and an additional password confirmation field. Both of these components will have a button at the end of the password field to be pressed and trigger the login or signup process.

On attempting a login, there is a login event that queries the database with the username and password entered. If the password entry in the database for that user entry does not match the password passed in through the form, it returns to the login page. On success, a route handler will direct the user to the home page view. A cookie will be generated to store user login information following a successful login.

On creating an account, we query the database for the username entered in the form to see if the username already exists. If it does, we return to the login page and display "invalid

username". We will also return to the login page if any of these fields are not suitable for their respective entries in the database (such as not writing anything for a password), which we will check using node before the data is passed into the database query. If the name is not in the database and all fields are valid, then a new entry will be created in the database and the user will be redirected to the home page view, with a cookie generated to store the new current user information.

Navigation Bar

The user will navigate through the application via a vertical navbar on the right hand side of the webpage. The navbar will be a full window height div element approximately 250 pixels in width containing the Repliance logo at the top. This image will function as a link, which redirects to the home view. Beneath this image link will be links titled "Account" and "Logout". The Account link redirects to the Account view. The Logout link will redirect to the Login view and delete the cookie generated for the login event. Underneath these will be three more links, "My Questions", "My Answers" and "Ask a new question". "My Questions" will redirect to the My Questions view, "My Answers" will redirect to the My Answers view, and "Ask a new question" will redirect to the Ask Question view. This navbar will also display the current user's name and "Score: X", where X is the number of points a user has achieved for their written comments. This aspect of the navbar will require a database query for the username and score elements of the row for the user in the Users table, whenever the web page is reloaded. This navbar is visible on every view except for Login.

Home

The home view is the main page of our application, featuring any questions which do not yet have the full amount of answers or completed time limit. These questions will be displayed in a list feed. Questions that need the fewest replies or have the shortest time left before they disappear from public view will be displayed toward the top. This sorting will be handled by JavaScript after the data is pulled from the database before populating into the page.

This question feed works by querying the database for questions that have not yet reached their allotted time limit or number of responses limit and displaying those questions in the

feed, which is easily identified by the status attribute. The user would be able to reply to any of the posts that are displayed on the home page. Each post will be displayed in an expandable div element with a button titled “view post”. These boxes will show the title of the question and its picture, scaled down, and the first few characters of the body text. Upon clicking the “view post” button, the box will expand and the reply form to enter a text reply will be revealed, as well as the rest of the body text and a full size of the image. By typing a response and selecting ‘reply’, the text that the user has written as a response in the reply field will be stored in the Answer table of the database, and the number of replies in the repliesTotal attribute of that question’s entry in the Questions table will be incremented. At this level of updating the Questions table, a check will be performed to see if the repliesLimit is equal to the repliesTotal. If it is, then 0 will be inserted into the status of that question in the Question table to show the post is closed from replies, preventing it from returning to the Home view’s feed. The database’s QA table will also be updated to link that new question with that new answer.

In order to check whether or not the time limit on the question has been reached, we will use the JavaScript object Date(). We will find save the amount of time that the user sets as the time limit for their question. We will then add that amount of time to the current Date() at the moment it is posted and save this as the “time of expiring” time. From there, we will check every second if the current Date() is greater than or equal to the time of expiring. If not, keep checking; if so, mark the question as expired.

Once a question has expired, the post div for that question will disappear for every user, including the user who had originally posted the question. The original poster would still be able to access his or her questions using the My Questions view. To make the post div for questions that have expired disappear, the database will be queried for all the questions associated with that user’s answers, and then make them hidden to the user.

Account

This view displays all information about the current user. There will be a list of information on this page, including “First Name”, “Last Name”, “Username” and “Score”. This information is generated and populated into the template with a query into the Users table for the current

user. Beneath each of these will be an option to update those fields. For first and last name, we will handle verifying that the input is valid with node before the data is attempted to be inserted into the database. Username updates will function similar, however first the database must be queried to determine that the new desired username does not already exist. Each of these updates will have a single form entry with a submit changes button. Upon completion, the page will reload, and the latest information about the current user will populate the display. This is also where a user may change their password. The last item on this page is a form for password updating, which contains three entries. The first is “current password”, followed by “new password” and “confirm new password”. The database will be queried for the current password. If this matches the newly entered “current password”, and the two fields in “new password” and “confirm new password” match, then the password will be updated in the database for the current user.

Post Question

This view allows users to ask a new question on Repliance. It will prompt the user for several pieces of information displayed in a form. The first is the title of their post, that is, their main question. The second field is the body text, where they can clarify further information. We will implement a character limit on these form fields, 30 characters for a post title and 500 characters for body text. There will also be a button to allow for image upload. At this time our intention is to implement the JavaScript middleware “multer” for this process, while storing the image in the database with the rest of the question data. Finally, there are two options of which the user must select one; number of replies needed and time limit. Both of these options will be selected from drop down menus, allowing for from 10 to 100 replies or five minutes to one hour for time. The user may select both, but must choose at least one. The “submit question” button available at the bottom of this page will begin the process for sending this data to the database to generate a new question. This functionality in JavaScript is where we will check to make sure that the user chose “replies needed” and/or “time needed” as an option. If there is a problem, the page will not redirect and an error will appear showing that something went wrong. When a post succeeds, it is added to the database and the user is redirected to the “My Questions” page to see the post, which will also be available from the home page. The Questions table in the database will hold all of the information provided in this form for each question entry in the table.

My Questions

This view will display all questions ever asked by the current user. Each question will be displayed in a div element, including any original text or images they had included when it was posted. These will be taken from the database with a query to the Questions table pulling rows matching the id of the current user. In the “status” field of each entry in Questions, there will be a 1 or a 0. An entry of 0 means that the question is “open” and still gathering replies, while “1” means that a question has closed and is no longer gathering replies. Sorting this result from lowest to high based on this field, we display open questions at the top with closed questions below.

Within each individual question’s bounding box will be a “Show Replies” button in the lower right. This triggers a query to the database to look up all answer id’s in the QA table which match the question id of the relevant question clicked, and then gather answers from the Answers table possessing the answer ideas found in QA. This gives us all answers pertaining to that particular question. Clicking the “Show Replies” button will expand the window of the bounding box to double its height, and load up to five replies underneath the question information. Beside these answers, there will be two left and right buttons which change the display to five other additional answers, allowing the user to click through until all answers have been shown in groups of five. This presents the answers to a particular question in a more accessible pattern rather than displaying potentially dozens of replies in one long list the user would need to scroll through.

When these answers are loaded, each will be displayed in its own div element. This includes the text of the answer, and three buttons. The buttons will be for rating the reply, with “Best Answer”, “Thumbs Up” and “Thumbs Down” available for the user to choose from. “Best Answer” is worth three points, “Thumbs Up” is worth two points, and “Thumbs Down” is worth zero points. Each button will send an insert to the database’s Answer table, finding that particular answer and updating the score value to “3” “2” or “0” for that answer (the default is 1 if it is never voted on). To avoid querying the database for a tally of answer scores every time the user’s Score is displayed in the navbar, there will be a function associated with each of

these button clicks that causes an update to the score of that particular user. This way there is only an update to the score when one of that user's replies was voted on, summing the score entries in the answers table with a database query and inserting the result into the user's score value in the Users table.

My Answers

This view will display a list of every comment the current user has ever written. The list will be generated by a database query to the Answers table, collecting every answer associated with the uid of the current user. The text of the answer, and its score, will be included in the display results from the query. Our database is designed such that even if a user deletes their question, the entry for each answer is not removed from the Answers table, allowing users to save their comments and thus their total points for their comment score.

Repliance Database Schema

foreign key:	*								
primary key:	dark color								
not null:	^								
unique:	$								
USERS									
int	varchar(20)	varchar(50)	varchar(50)	varchar(50)	int				
uid	username^$	password^	firstName	lastName	score^				
5978	JamesD	whatever	James	Desjardin	14				
ANSWERS									
int	int	int	text						
aid	uid*^	score^	text^						
200	5978	2	"Yeah totally."						
QUESTIONS									
int	int	int	int	int	int	bytea	text	text	int
qid	uid*^	repliesTotal^	repliesLimit	timeTotal	timeLimit	image	bodytext^	title^	status^
10	5978	3	10	:04:55:00	:13:00:00	10.jpg	"Never tried it."	"Is this fun?"	0
QA									
int	int								
qid*	aid*								
10	200								