2022

PS1 Code Judgement Rubric

ONLY CODE JUDGES SHOULD BE USING THIS VERSION OF THE RUBRIC!

Judge (not shared with students): _____
Judged Team (name or number): _____
Date: _____

Instructions: for each requirement, mark the box that most closely aligns to your impression of the solution given the prompt. Your judgments here should reflect how well the solution satisfies the directions in the problem statement and the team's presentation as well as your own review of the solution's source code. **Aesthetic appeal of a solution is <u>not</u> more important than functionality.** If a solution is not pretty but otherwise meets a requirement, that counts as reasonably satisfying that requirement. Keep your expectations high, teams had **a whole month** to work on this. **DO NOT FEEL OBLIGATED TO AWARD HIGH SCORES!** In the case that a solution attempts to "satisfy" a requirement by making it *look* like it works via the UI/demo code but the required feature is actually non-functional, <u>a score of 0 is appropriate.</u> Also, note that you **do not** have to fill this rubric out in order!

Solution Functionality Review

[total weight: 100]

Requirement(s) to Reference	Score			
	0	1	2	3
	Missing or Completely Non-functional	Attempted But Does Not Meet My Expectations	Meets My Expectations	Exceeds My Expectations
Requirement 1 Requirement 2 Requirement 3 Requirement 6	You **do not have to** login before being able to view questions/answers. **[weight: 2]**			
	Providing random login credentials *does not* allow you to log into the system successfully. **[weight: 2]**			
	Providing the correct login credentials *does* allow you to log into the system successfully. **[weight: 2]**			
	When you are already logged in, you should not be able to create a new account using the same URL/link used to create accounts when you're not logged in. **[weight: 1]**			
	You can create a brand new account via a registration feature; you can login successfully using the account you just created. **[weight: 2]**			

	You can view and create questions, answers, and comments pursuant to the level-based restrictions outlined in the problem statement text. **[weight: 4]**			
	You can give, deduct, and receive points by creating, voting on, and answering questions as an authed user pursuant to the level-based restrictions outlined in the problem statement text. **[weight: 5]**			
	You can attain increasingly powerful “levels” as an authed user by maintaining a certain threshold of points outlined in the problem statement text. **[weight: 5]**			
Requirement 2 Requirement 3	The Buffet view (the page that lists the latest/sorted questions in the system) is pleasant to look at, elegant, and ergonomic. It is pretty simple to figure out and use. It has the appropriate components spelled out in the requirement text and provides the functionality spelled out at each bullet point. **[weight: 5]**			
	You can sort questions by “most recent,” “best,” “most interesting,” or “hottest” as defined in the problem statement text. **[weight: 3]**			
	For each question, you can see the profile picture and other information of the user that created it. **[weight: 1]**			
	The Buffet and/or Q&A views support creating new questions written in Markdown. New questions can be previewed as rendered Markdown before being submitted. **[weight: 3]**			

	You cannot create a question with a title longer than 150 characters or a body longer than 3000 characters. **[weight: 1]**			
Requirement 3	The Q&A view (the page that displays a specific question, its answers, and related comments) is pleasant to look at, elegant, and ergonomic. It is pretty simple to figure out and use. It has the appropriate components spelled out in the requirement text and provides the functionality spelled out at each bullet point. **[weight: 5]**			
	The Q&A view supports creating new answers written in Markdown. New answers can be previewed as rendered Markdown before being submitted. **[weight: 3]**			
	You cannot create an answer with a body longer than 3000 characters. **[weight: 1]**			
	Answers appear sorted by total or “net” votes (upvotes minus downvotes) with the highest appearing first. Comments appear in order of creation (oldest first). **[weight: 1]**			
	You cannot create new answers if your account has less than 1 point. **[weight: 1]**			
	You can accept one *and only one* answer as the “accepted answer” to your question. It is obvious which answer is the accepted answer. **[weight: 2]**			
	Each question and answer shows information about the user that created it, including their profile picture and that user’s total number of points. **[weight: 1]**			

	You can vote to protect, close, and reopen questions pursuant to the text of the problem statement. You may need to use multiple high-level accounts to test this functionality. **If you cannot access such accounts, please contact us in Slack and we'll create them for you.** Also, note that these votes are per-app only (in-progress votes in one team's solution will not be visible in another team's solution). **[weight: 4]**			
Requirement 4	The Mail view (the page that lists all the private messages or "DMs" between users) is pleasant to look at, elegant, and ergonomic. It is pretty simple to figure out and use. It has the appropriate components spelled out in the requirement text and provides the functionality spelled out at each bullet point. **[weight: 5]**			
	Two users can converse back and forth privately with one another via messages using the Mail view. You should create two accounts open in two different browser sessions to test this. **[weight: 2]**			
	Message subjects are limited to 75 characters and bodies are limited to 150 characters. **[weight: 1]**			
	Messages appear in a sensible order in the Mail view. **[weight: 1]**			
	The Mail view supports Markdown. Messages are rendered as Markdown. New messages can be previewed as rendered Markdown before being sent. **[weight: 3]**			
Requirement 5	The Dashboard view (the page allowing users to view information about themselves) is pleasant to look at, elegant, and ergonomic. It is pretty simple to figure out and use. It has the appropriate components spelled out in the requirement text and provides the functionality spelled out at each bullet point. **[weight: 5]**			

Requirement 6	The Auth view (the page that allows you to login and/or register) is pleasant to look at, elegant, and ergonomic. It is pretty simple to figure out and use. It has the appropriate components spelled out in the requirement text and provides the functionality spelled out at each bullet point. **[weight: 5]**			
	You cannot use the same email address or username to register more than one user. That is: usernames and email addresses must be unique. **[weight: 1]**			
	Registration requires a CAPTCHA. Users are restricted from logging in for 1 hour after 3 failed login attempts. Users can see how many attempts they have remaining. **[weight: 2]**			
	When logging in, you can use “remember me” functionality to stay logged in. That is: you can login using “remember me,” completely exit the browser, reopen it, and still be logged in afterwards. When you disable “remember me” functionality (i.e. log out, then login again without using “remember me”) and you exit and relaunch the browser the same way, you should no longer be logged in. **NOTE: ensure the browser’s session restoration “features” are disabled if you’re having trouble with this one.** **[weight: 2]**			
Requirement 7	You can use the password recovery feature to regain access to an account without the password. This should be available through a “forgot password” link of some kind when visiting the Auth view. **You MUST be forced to set a new password and should NEVER be shown your old “forgotten” password which would be a dire security failure.** **[weight: 2]**			
Requirement 8	The app has a navigation element with the BDPA logo, app title, and a subset of user data (for authed users) visible in all views. **[weight: 2]**			
Requirement 9	Any user (including guest users) can search for questions in the system pursuant to the			

	criteria listed in the problem statement text, including sorting results. **[weight: 1]**			
Requirement 10	The upvote/downvote counts of visible questions, answers, and comments in the Q&A and Buffet views *eventually* update on the screen automatically when changed in the backend without you having to press a refresh button or take any other actions. You can test this by creating multiple users who view, upvote, and downvote the same questions, answers, and comments. **[weight: 4]**			
Requirement 12	When viewing lists of questions, answers, comments, etc., all of these results are *paginated* in an aesthetically pleasing way. See the text of the problem statement for more details. **[weight: 1]**			
Requirement 14	The app fails gracefully when errors happen. You don't lose all your progress in the registration form when you press submit with invalid data, for instance. When loading a lot of data, the app does not freeze or present a broken UI but instead shows a spinner icon or some other loading indicator while waiting. You are never waiting too long for something to happen in response to user input. The app never "breaks," never displays internal error messages or warnings anywhere on the page, the backend server hosting the solution never crashes, never any HTTP 500 errors, etc. **[weight: 1]**			
Requirement 15	The app looks good when you view it on a small smartphone-sized screen. **One way to simulate this is by using your mouse to drag the right edge of the browser window to the left until it cannot shrink anymore. This shrunken column represents the size of a phone screen.** Is the app still usable in this state? Is it visually pleasant? **[weight: 2]**			
	The app looks good when you view it on a medium tablet-sized screen. **To simulate this, make your browser window half the size of your screen (try windows key + left arrow). This simulates the size of a tablet screen.** Is the app still usable? Is it visually pleasant? **[weight: 1]**			

	The app looks good when you view it on a normal desktop/laptop screen. **Ensure the browser window is maximized.** Is the app still usable? Is it visually pleasant? **[weight: 2]**			
	Overall, you feel this solution is easy to use. **[weight: 1]**			
	Overall, you feel this solution is beautiful to look at. **[weight: 1]**			
	Overall, you feel this solution functioned well. App functionality that worked fine when the browser window was maximized did not suddenly break when you shrunk the browser to simulate a smartphone or a tablet. **[weight: 1]**			

Instructions: for each of the five criteria below, mark the box that most closely aligns to your impression of the solution's source code after your review. Does their source code meet your expectations of quality? **If you notice source code that seems strangely out of place, overly complex, extremely nonsensical, suspiciously unformatted or very badly organized syntactically, or you otherwise infer that students copy-pasted/plagiarized part of their solution's source code from the internet or some illegal source, inform the chief judge immediately.** Also, note that you **do not** have to fill this rubric out in order!

Solution Source Code Review

[total weight: 100]

Criterion	Score			
	0	1	2	3
	Does Not Meet Expectations	Partially Meets	Meets Expectations	Exceeds Expectations
Naming Conventions	Variables should be well-named (e.g. **accountSum** is good while **acsu** is bad). There should be few "magic numbers" floating around. The chosen variable naming scheme should be consistent and self-documenting. **[weight: 5]**			
Organization	The code should be well-organized, with a consistent logical file/directory layout, encapsulation, and proper separation of concerns. Similar units of code should be grouped together in a logical way. If object-oriented language features are used, they should be used properly (i.e. SOLID). **[weight: 30]**			
Maintainability and Extensibility	The solution should be implemented such that a future group of programmers would not have to rewrite large portions of the source code to add new functionality. Source code should be DRY where reasonable. Pay special care to inflexible and hardcoded values. **[weight: 25]**			
Readability	How easily readable is the source code? Did you have a hard time going through it? Solution code should be as intuitive and self-documenting as possible. Confusing sections should be documented with comments. There should be few "WTF?!" moments. **[weight: 25]**			

Efficiency	Any implementations should not be too inefficient. Does some piece of code waste cycles looping or otherwise doing something unnecessary? Is the application wasteful with system resources? Does it make many more network requests than necessary? **[weight: 15]**			

Instructions: please provide any additional comments you have below. Note that your comments are shared directly with the students. Please be thorough, encouraging, and fair.

Positive Comments

Please share any positive comments you have for this team.

Constructive Criticism

Please share any unanswered concerns or comments you have for this team.

Two Questions For This Team

Please share the two most pressing questions you'd like answered about this team's solution.

Shared Thoughts (visible to all judges)

Please add any thoughts or concerns you believe other judges should be made aware of.