2023

PS2 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 not 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, a score of 0 is appropriate. Also, note that you **do not** have to fill this rubric out in order! If you have *any questions at all*, please reach out ASAP!

Solution Functionality Review
[total weight: 100]

Requirement(s) to Reference	Score			
	0	1	2	3
	Missing or Completely Non-functional	Attempted But Does Not Satisfy Requirement	Reasonably Satisfies Requirement	Exceeds Requirement Expectations
Change 2	When creating a new user, a full name is required. When accessing a user's profile, you can see their full name if they have one in the system. You can login as a user and change the full name to something else, which should be reflected in that user's profile immediately. **[weight: 5]**			
Change 3	The Feed view (the page that lists all of the articles a user can see) 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]**			
	Users can create their own articles, which appear in their own feed. **[weight: 5]**			
	Along with their own articles, users can see articles from users to which they're connected. Other articles cannot be seen by users. **[weight: 5]**			

	The Article view supports Markdown. Articles are rendered as Markdown and are pleasant to look at. Users can create new articles and edit existing articles they own. **[weight: 5]**			
Change 4	You can filter which articles are shown in the Article view UI by keyword. **[weight: 20]**			
Change 5	When viewing an article, the title of the page (listed in your browser's title bar) is changed to match the title of the article. **[weight: 5]**			
Change 6	Authenticated users can change their email and full name from within the Profile view. **[weight: 5]**			
	`administrators`, when impersonating a user, can change that user's email address or full name from within the Profile view. **[weight: 5]**			
Change 7	`Administrators` can demote `staff` accounts back into `inner` accounts. **[weight: 5]**			
Change 8	The Opportunity view now shows, along with the total number of active viewers/sessions, how many of those active viewers are authenticated vs how many are not. For example, something like: "10 active viewers (4 authenticated, 6 unauthenticated)" **[weight: 5]**			
Change 9	From the Admin view, you can see an overview of all activity within the system pursuant to the text of the requirement. **[weight: 10]**			
	`administrators` can sort and filter data when interacting with the overview of all system activity. **[weight: 5]**			

	System activity updates asynchronously/automatically (give it 60 seconds) without you having to press a refresh button or take any other actions. You can test this by creating multiple users across multiple tabs that are viewing and interacting with the same opportunities/profiles/articles; you should eventually see the changes from one tab reflected in the other. **[weight: 10]**			
Change 10	`administrators` can delete users via the Admin view. Any user can delete their own account. Deleting an account does not delete the opportunities associated with that account, but it does delete the articles associated with that account. **[weight: 5]**			

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! If you have *any questions at all*, please reach out ASAP!

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: 20]**			
Security	The solution should use modern software engineering practices that protect from common XSS, SQL injection, and other security vulnerabilities. Specifically: form input should be properly sanitized and should not be vulnerable to SQL injection attacks (example). User-generated outputs should not be vulnerable to XSS attacks (example). **[weight: 20]**			
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: 20]**			
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: 20]**			
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.