Final Updated Scope and Charter Report

# TA Allocation Scheduler

## Team Members

**REDACTED**

**REDACTED**

Brandon Racich

**REDACTED**

**REDACTED**

**Table of Contents**

[**TA Allocation Scheduler**](#_30j0zll) **1**

[Team Members](#_frsf3e5vdxvq) 1

[**Overview**](#_1fob9te) **3**

[Business Case](#_3znysh7) 3

[Project Purpose](#_o1oo93w9g8m) 3

[Goals](#_mt6xzptm70my) 3

[Scope](#_ert6mrrzknxm) 3

[Workflow](#_9kc81gxg2wm1) 4

[Testing](#_e3ufr62ufuhi) 4

[Deployment environment](#_jt0csof307wl) 4

[**Non Functional Requirements**](#_hhvgp1lr6iiv) **4**

[Performance](#_s893cu2ddlc7) 4

[Development](#_dz5780czsh1s) 5

[**Technical Requirements**](#_mbsuatscfynq) **5**

[**Functional Requirements (BUILT)**](#_ife3nh147jh6) **5**

[Admin / Employer Requirements](#_dqvoh6g8hawi) 5

[User/Applicant Requirements](#_3rdcrjn) 6

[Workflow - Stretch Goals](#_9u3t39xvxvqq) 6

[**Functional Requirements (NOT BUILT)**](#_6e073ndhpj) **7**

[Admin / Employer Requirements](#_hs7sbif4p7yf) 7

[User/Applicant Requirements](#_ywdxk454xg06) 7

[Workflow - Stretch Goals](#_lo60lpkrgo0m) 7

[**Additional Functional Requirements**](#_tgyn2m8o1jlj) **7**

[**User Groups and Scenarios**](#_4h7co8oexka9) **8**

[Students](#_qru6aknj726m) 8

[Admins](#_j2wzw3uu2hk7) 8

[Student Scenario:](#_rnqfe9vwbbol) 8

[Admin Scenario:](#_4q29cmr9ykad) 8

[**Use Cases**](#_58gt4sg2fqnw) **8**

[**Test Data from User Testing Event**](#_3pcy9p3gzorf) **8**

[Admin Functions Feedback](#_jksisyo2wbuo) 9

[Student Functions Feedback](#_q88avwwq2jek) 10

[Both Student and Admin Functions Feedback](#_n1w71c39z8pr) 11

[**Automated Scheduler**](#_nft2e3l4f9h9) **12**

[Objective Function:](#_r9wzv2w24fpc) 13

[Applicant Constraints:](#_vnfcqvm1k4s) 13

[Department Level Constraints:](#_uxp8haomkkyd) 14

[Lab Constraints:](#_bpwdfk5pfm63) 14

[Technical Constraints:](#_7xg46fv5x1nt) 14

[**Assumptions**](#_h5z34zev69ph) **15**

[Assumptions](#_tdp86l89lti4) 15

[**Work Breakdown Structure**](#_l9uncy7r3zjy) **15**

[**Conclusion**](#_yh3dva4wxqzr) **15**

[Key lessons learned from the project](#_qu41m3pic607) 15

# Overview

In our overview, we first have our business case which lists our solution to a real problem at UBCO. The

Project Purpose section defines the reasons for why this project is necessary, and the Goals subsection lists the main issues we’re trying to solve. Scope lists what is being made and delivered to the client. Next is our Workflow and Methodology, Use Case Diagram, and finally, Stakeholders.

## Business Case

The scheduler will have three methods for the allocation of TAs to course and labs slots. Our fully automated scheduling will act as a fully integrated system that automates the intake of potential applicants and the appointment of TAs to various labs and courses while taking into account a set of restrictions and preferences. The system will then create a feasible schedule set. The functionality for partial assignment and manual assignment also exist if needed.

The project will allow students to fill out an online application to apply for a TA job. The employer will be able to see all the applicants. The program will give suggested TAs for each lab or lecture slot that are available according to the restrictions and preferences provided.

## Project Purpose

The process of assigning TAs to lab sections for computer science courses at UBCO is currently a tedious process of manually looking through the lab sections and searching for applicants who are available during that scheduled lab time. This system was established well before the department reached its current size. This process is no longer ideal due to the nature of the numerous constraints and preferences that exist within the department, the applicants themselves, and the professors of the courses. It would be ideal to have a system which allows for TA applications to be submitted electronically with its course history, and the schedule of each applicant, will automate the wearisome process of allocating TAs. The system will consider constraints and preferences both set by the applicant such as preferred lab times, and set by the department such as maximum total TA hours.

## Goals

1. Decrease the time it takes to allocate TAs to labs by cutting down on the manual labour, thereby increasing the efficiency of the allocation process.
2. Complete the project in a timely manner meeting all expected functional requirements and security requirements on budget, and on time.
3. Automate all or a portion of the allocation process.

## Scope

The goal of the project is to create a web portal to assist in the assignment of TAs to lab sections for professors of UBC, Department 5 will be the main users of the service with aims to expand and scale the web portal to other Departments if desired. The TA planner will find a feasible schedule of TAs based on hard and soft constraints provided as well as supporting manual assignment. The ability to review TA applications, history, and schedules is a requested feature. There will be no access to UBC servers so data will need to be manually constructed and uploaded, all data will be stored within its own database and server, deployment will be handled by UBC with the deliverable being the software developed. A well tested prototype software is to be finished by April 2019. Deployment and maintenance will not be handled by the developers.

Best efforts will be made to make the program scalable towards multiple departments however the program may need modifications to be used in departments other than Department 5. This is due to the unknown nature of the restrictions of other departments as well as the time constraint of the April 2019 due date.

## Workflow

Our general workflow was assigning ourselves trello tasks and having meeting ensuring our tasks got done. We utilized the Kanban strategy as our project was changing and needed to adapt to new client demands.

## Testing

Testing was done alongside our continuous integration with Travis CI. PHP unit and some unit testing within the web application were used.

## Deployment environment

Our merges to the dev branch on github uploaded our changes on the Digital Ocean live server which utilizes the LAMP stack.

# Non Functional Requirements

In this section Performance, and Development Non Functional Requirements are listed, these cover the technical requirements of the software.

## Performance

Data security

* All data stored in Canada
* Data is accessible only by authenticated users
* Website uses prepared SQL to prevent injection attacks
* Validations of user input by both client side and server side methods
* User data will be stored secured and stored properly and appropriately where applicable
* SSL to encrypt user-to-server communications

Flexibility

* Data can be entered as different methods (manual, automatic). The website receives data through both methods and joins it together

Usability

* Performing database queries will list results in less than 10 seconds
* Automatic scheduling will return a result within the user defined time limit plus 20 seconds to account for overhead of passing data and reading data back
* Automatic scheduling should immediately notify the user if a problem is encountered (infeasible, problem reading data, etc.)
* Easy to read - Well written english, grammatically and punctually correct. Between one and five concise sentences explaining functions if needed
* Easy to use

Maintainability

* Well commented code that can accept modular code

## Development

Design

* Project developed using OOP design principles

Maintainable

* Code structured using an MVC framework
  + Expandable
  + Modular
  + Scalable

# Technical Requirements

The system was developed for Google Chrome and is expected to work on other browsers without any critical faults. The system is hosted on Digital Ocean using a LAMP stack running PHP, Javascript, Python 3, and a MySQL database. The web portal was developed using an MVC framework where PHP was used as the server side language, requesting data from the DB and acting as the Model. On the client side, Javascript was used for the Controllers and handles updating the HTML Views with the data requested from the Model. Google OR tools solves the constraint problem to build the schedule, which occurs on the web server. The user has the option before and/or after the feasibility function runs to make inputs from the client side to the server to assign a preferred TA to a section. When the user manually adds TAs to specific sections, the system uses an algorithm based off constraints to recommend TAs for that section.

# Functional Requirements (BUILT)

Below are the Admin/Employer Requirements and User/Applicant Requirement.

## Admin / Employer Requirements

* Admin can print reports of selected content or all content
* Login in a secure fashion using web safe security protocols
* View all applicants as a list with their year level, currently assigned hours, minimum hours, maximum hours, and preferred number of hours, and how many hours there are left available to allocate to them
* Modify TA info (in case they entered the wrong info)
  + Set minimum and maximum number of hours for each TA
* Filter the view of applicants according to the criteria such as degree and year level
* View individual course and its lab slots as list
* View a list of recommended TAs for a given lab or lecture slot
* Assign TAs to lab and times manually, via the automated feasible schedule function or a combination of both
* View a list of courses with filter options
* Assign TAs a total number of hours that is broken down into different categories. (Prep, Marking, In Lab, and Other)
* Define the scheduling restrictions and prerequisites associated for a given course in order for an applicant to be eligible to TA a course
* Filter a set of courses for scheduling
* Upload a course schedule
* Add/Remove sections
* Create additional admin accounts

## User/Applicant Requirements

* Students can create accounts
* Login in a secure fashion using web safe security protocols
* Upload their transcript in the application as CSV file
  + The user will copy paste their transcript into a CSV file and will upload their transcript via a secure file upload
* Upload their course schedule with their application to indicate their availability with an .ics file
* Set their unavailable times
* View past assignments

## Workflow - Stretch Goals

* TAs can register, apply for TAship by filling out a digitized version of the current Unit 5 application form
* Admin can email TAs with their assignments

# Functional Requirements (NOT BUILT)

Below are the Admin/Employer Requirements, User/Applicant Requirement and Stretch Goals that were NOT implemented into the project.

## Admin / Employer Requirements

* View individual course **lecture** slots as list
* View individual course. its lab, and lecture slots **as a calendar**
* Assign TAs to **lecture** times manually, via the automated feasible schedule function or a combination of both
* Assign TAs **coordination hours**
  + The system will give a **default or suggest hour break down** for the TA depending on the course
* Assign a **coordinator** to a course

## User/Applicant Requirements

* Request their **preferred courses**
* View **applications**

## Workflow - Stretch Goals

* Admin can **email contracts** to TAs
* TAs can sign and upload their **TA contracts**
* Admin can **approve or modify assignment, and resend** the amended contract
* Admin can **email reports to professors** of each course with their assigned TAs

# Additional Functional Requirements

Below are additional functional features we implemented to the project.

* Account holders can reset their forgotten passwords

# User Groups and Scenarios

## Students

Students want a fast and easy way to send their offers to UBCO. Most students prefer electronic submissions for its speed and the lack of dealing with papers. Viewing their assigned courses should be a one click process with all the information being displayed on a single page.

## Admins

Admins want a fast and efficient method for managing and listing TAs. They want to the process of TA selection to be easy and fast as possible. The ability for manual assignment along side automatic assignment is a must, as flexibility in this function will save the admin a lot of time.

## Student Scenario:

Student Bob would like to apply TAship for Computer Science. He asks his professor how to apply and is given a website link. He heads on over to **REDACTED**

and creates an account. He logs in and is greeted with the TA portal screen. He would then submit an application with his prefered hours, transcript, and schedule then await for his offer via email. After receiving his emailed offer, he could view his account to look at his assigned courses and hours.

## Admin Scenario:

Admin Alice wishes to accept Student Bob to TA for COSC 101. Alice would first log into her admin account. Upon loading the Admin Portal, she’d click on Applications to find Bob’s new Application offer. She would click on “Accept” for Bob’s offer status. Then, assign Bob to TA one of the sections for COSC 101 on the Course List page. After that, she would click on the view more info button to assign some hours to Bob. Once that is done, she would head over to the List TAs page to email the offer to Bob.

# Use Cases

\* In the updated design document

# Test Data from User Testing Event

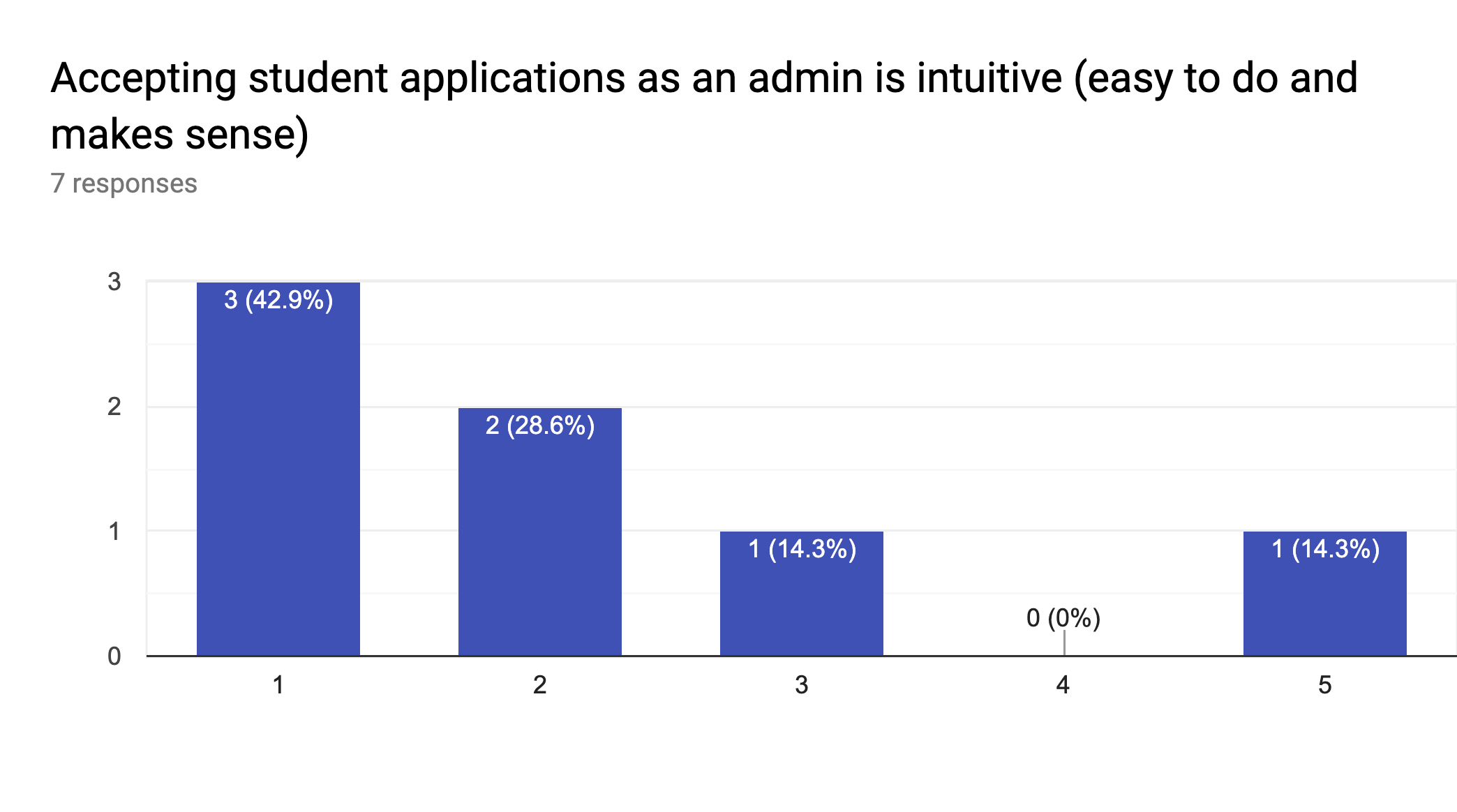
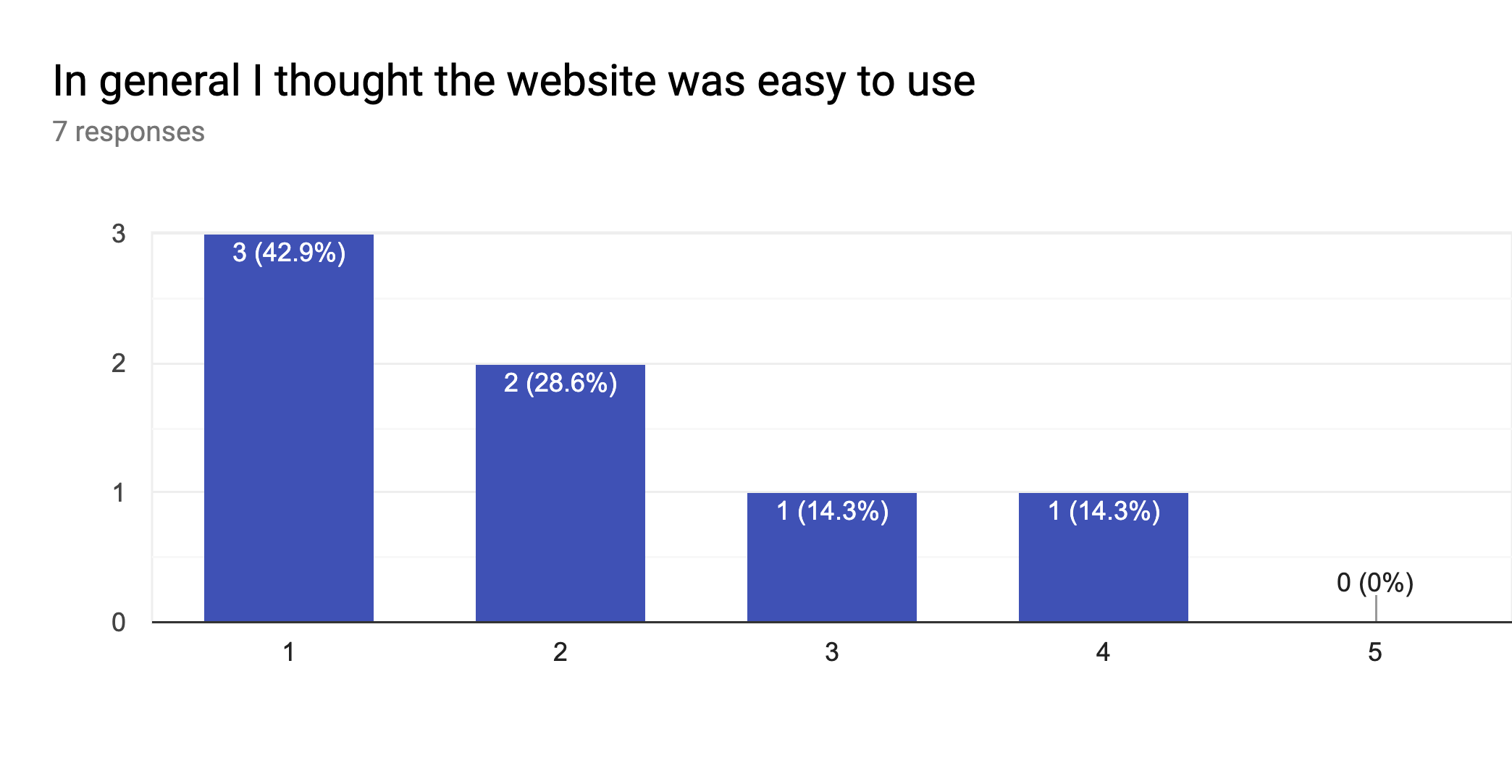
We had 19 users test our project. Of those user's, 10 completed admin functions, 7 completed student functions and 2 did both.

The scale we used:

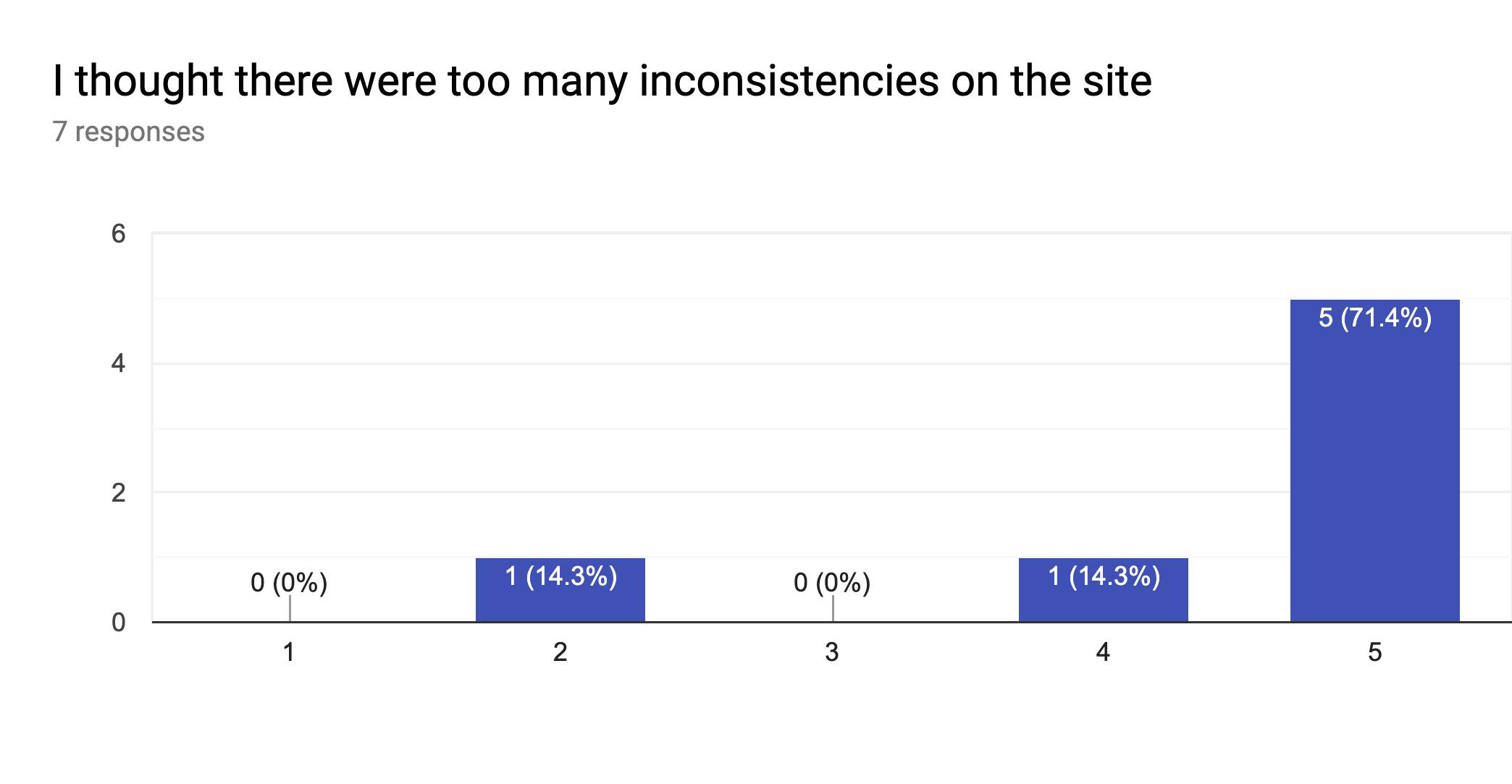
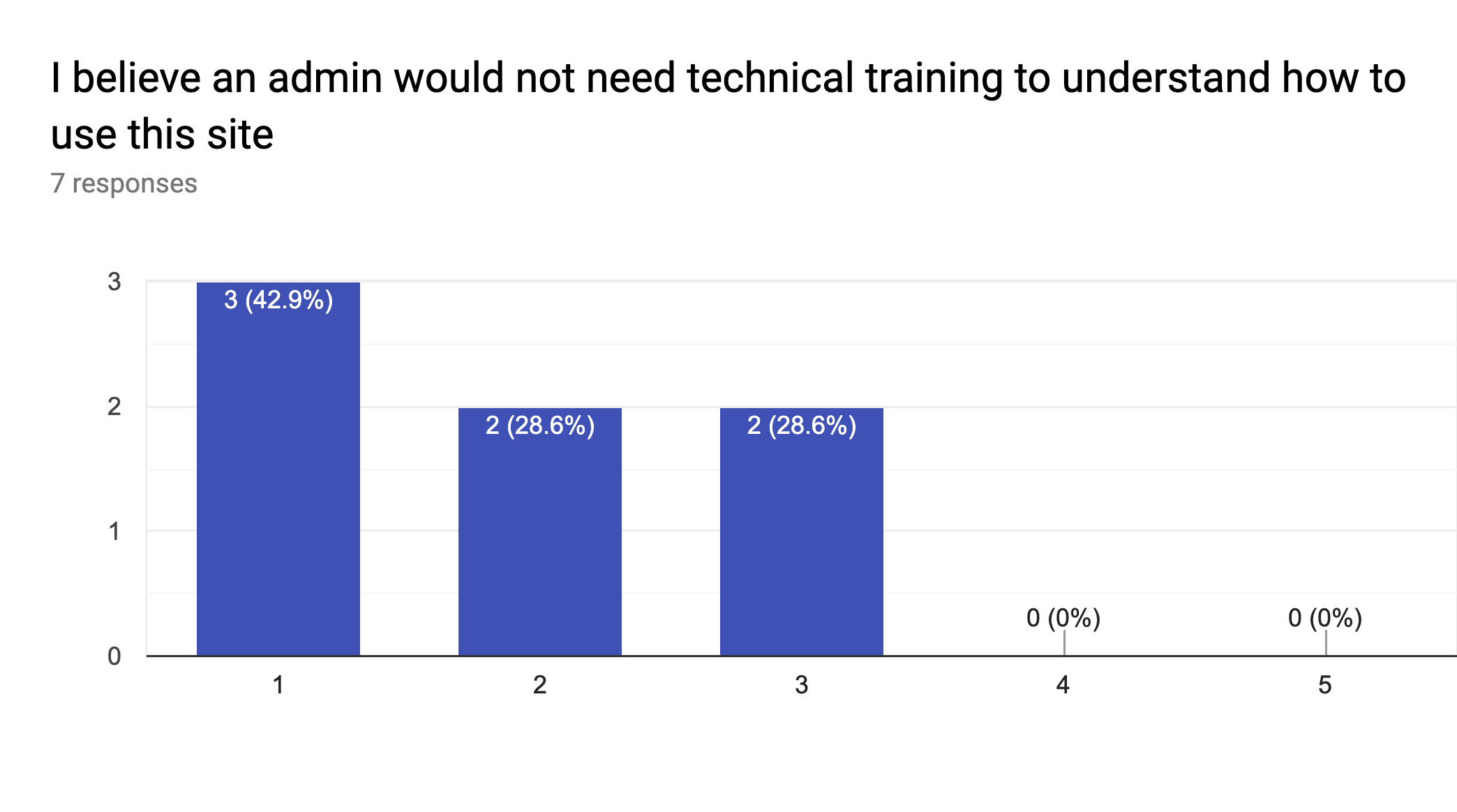
1 - Strongly Agree 2 - Agree 3 - Neutral 4 - Disagree 5 - Strongly Disagree

## 

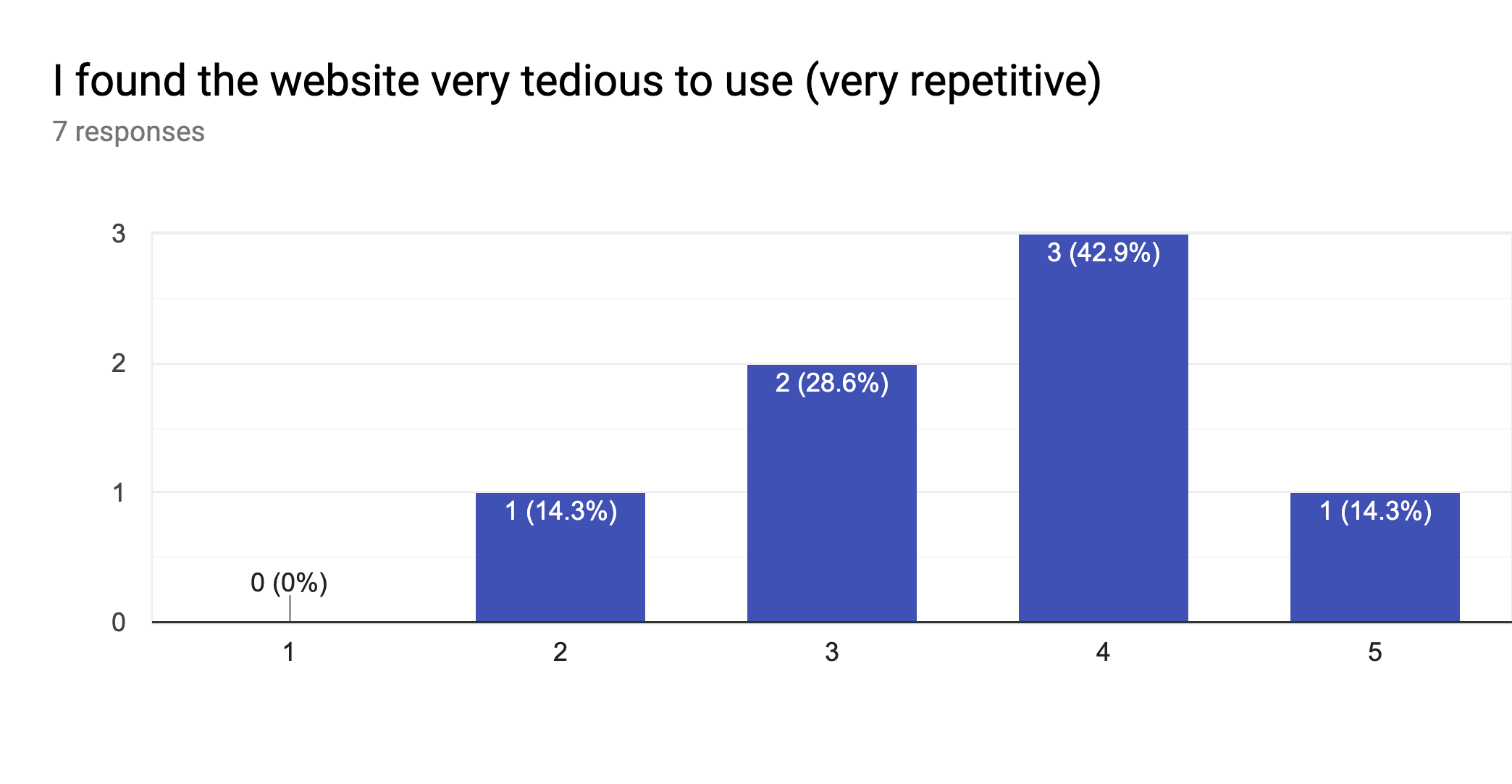
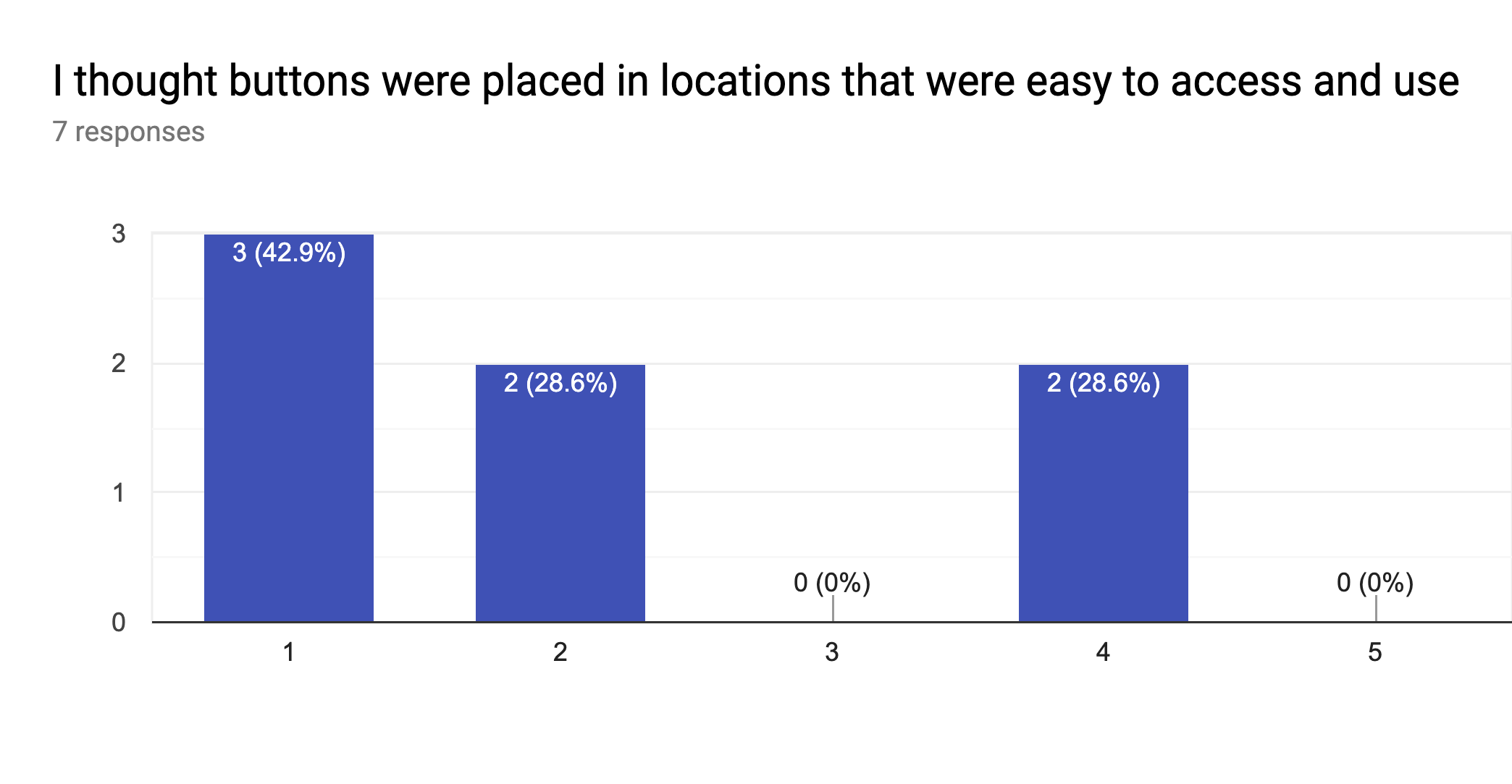
## Admin Functions Feedback

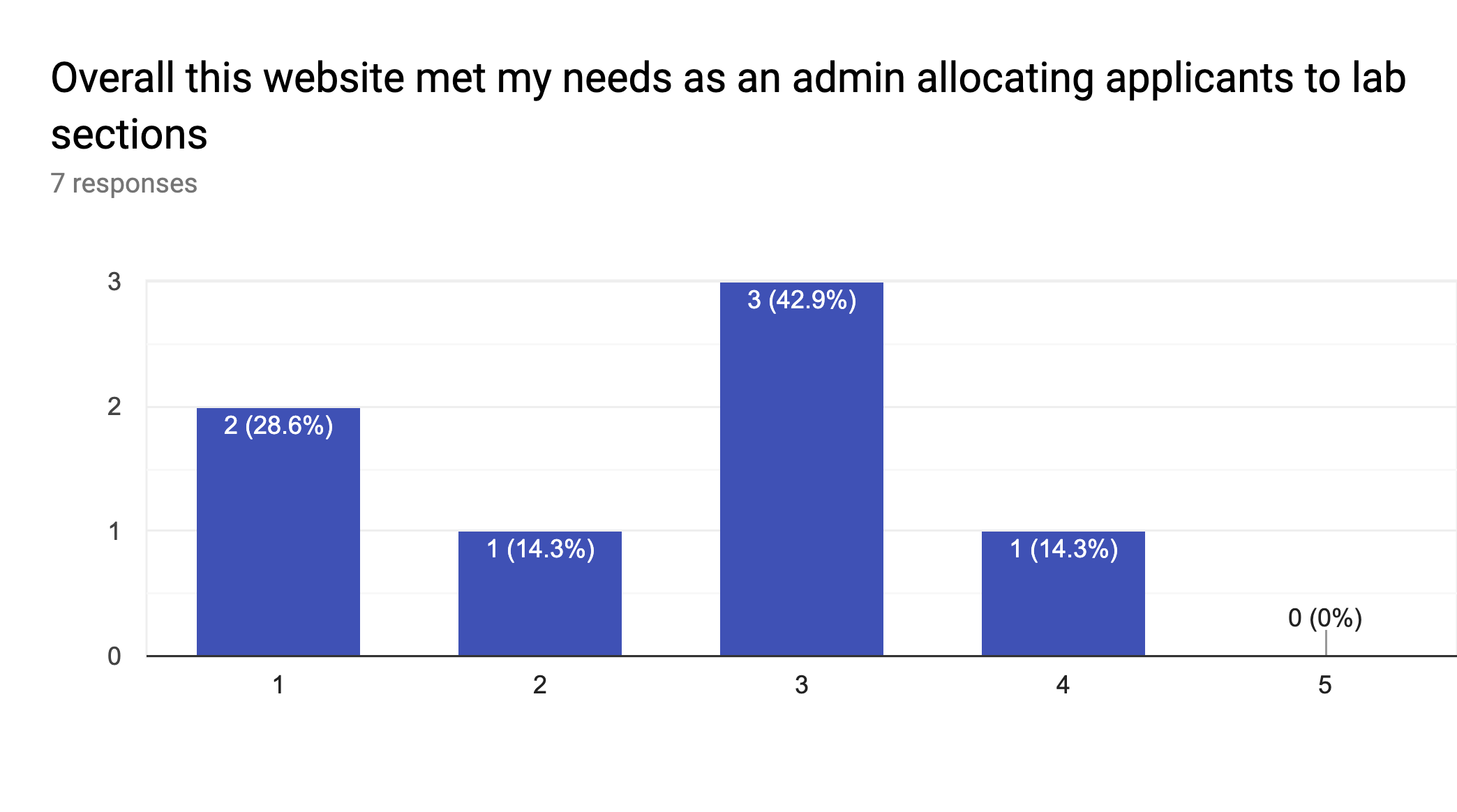


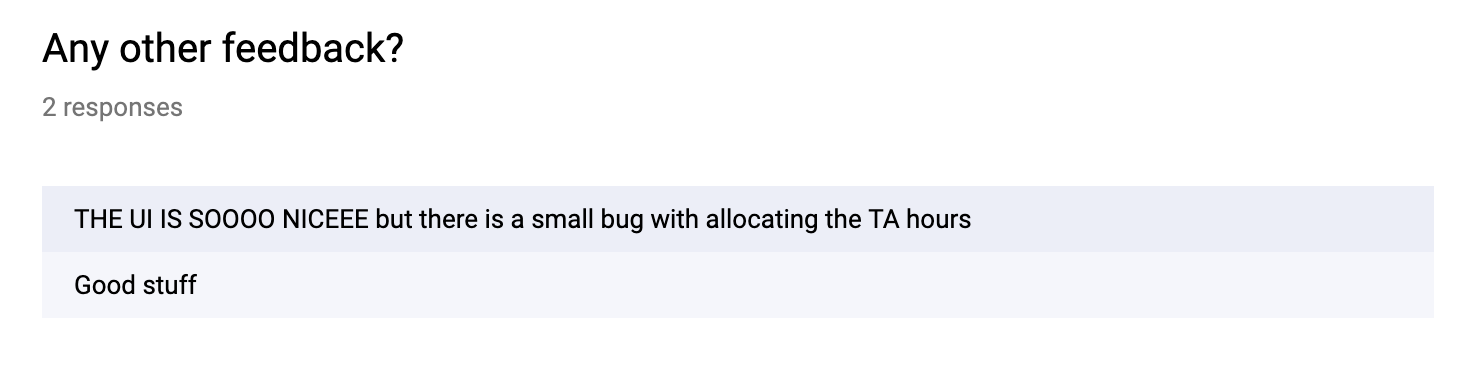
## 



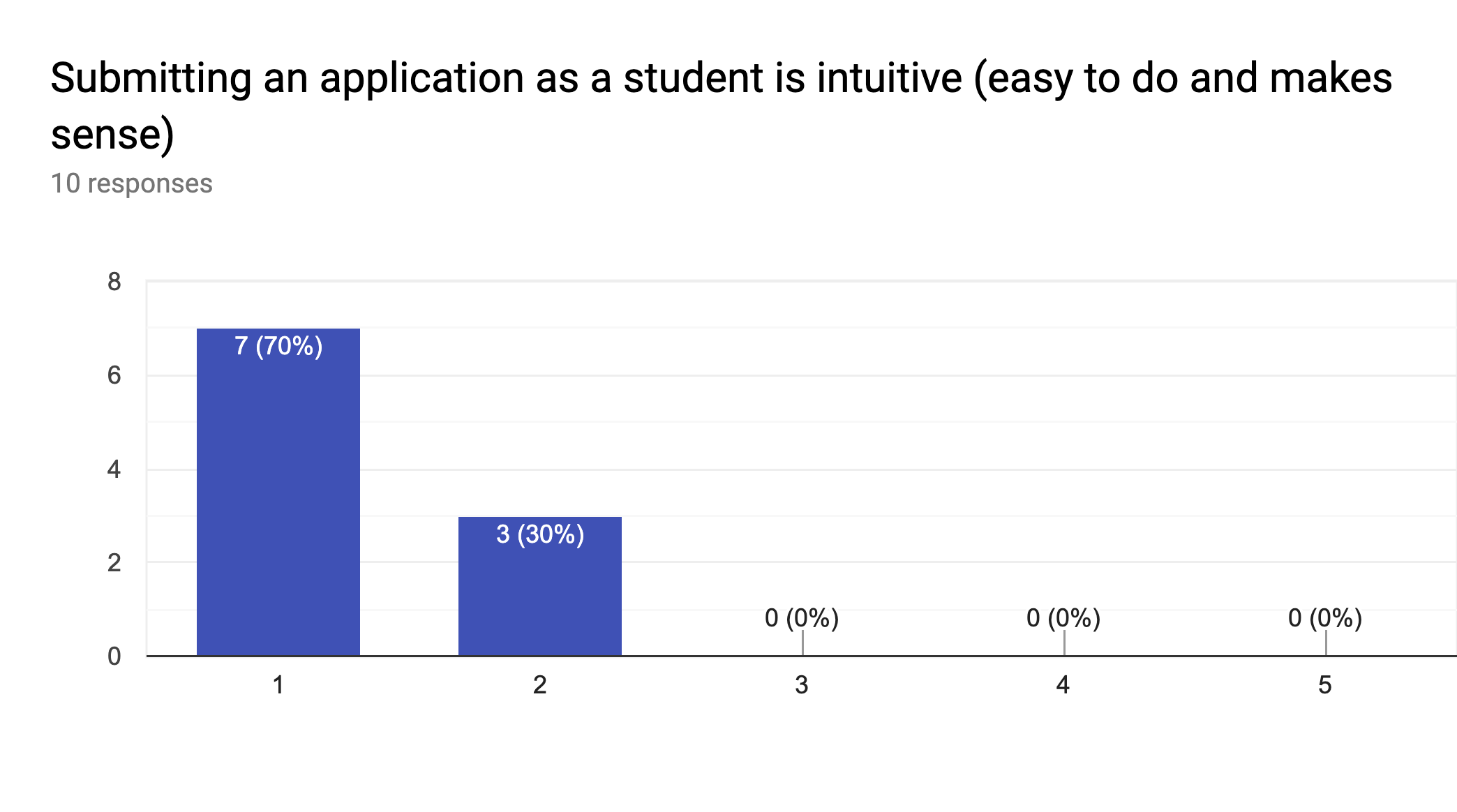
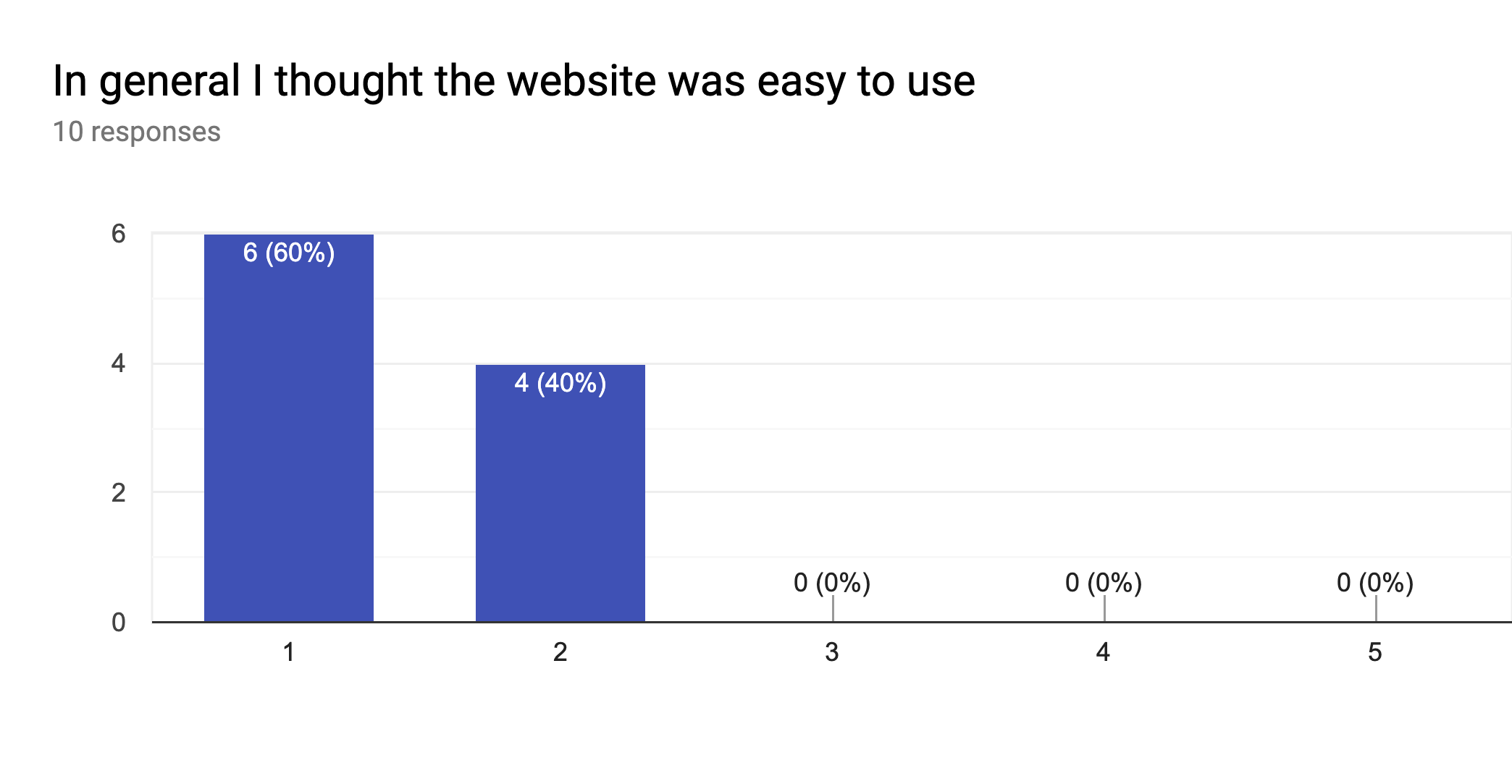
## 

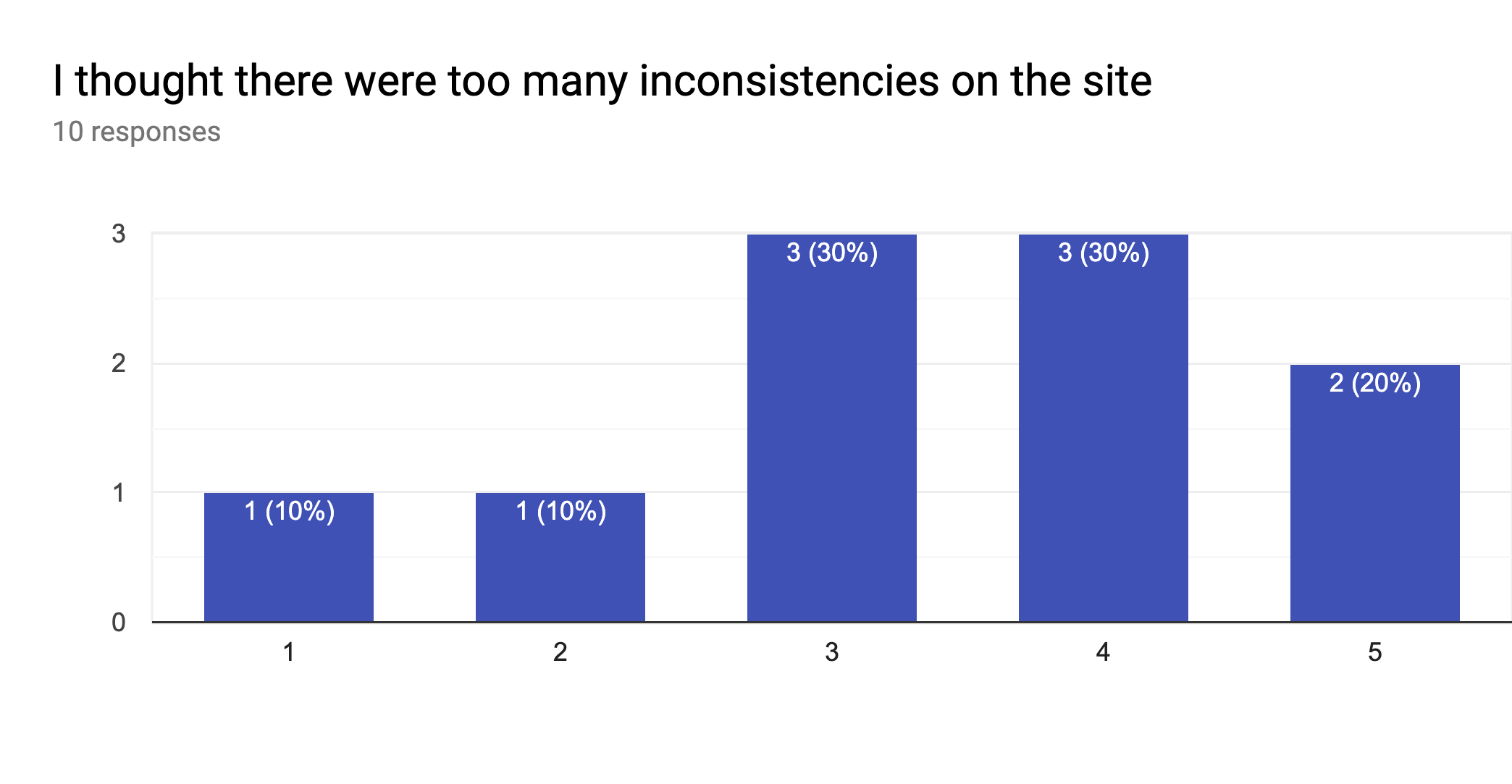
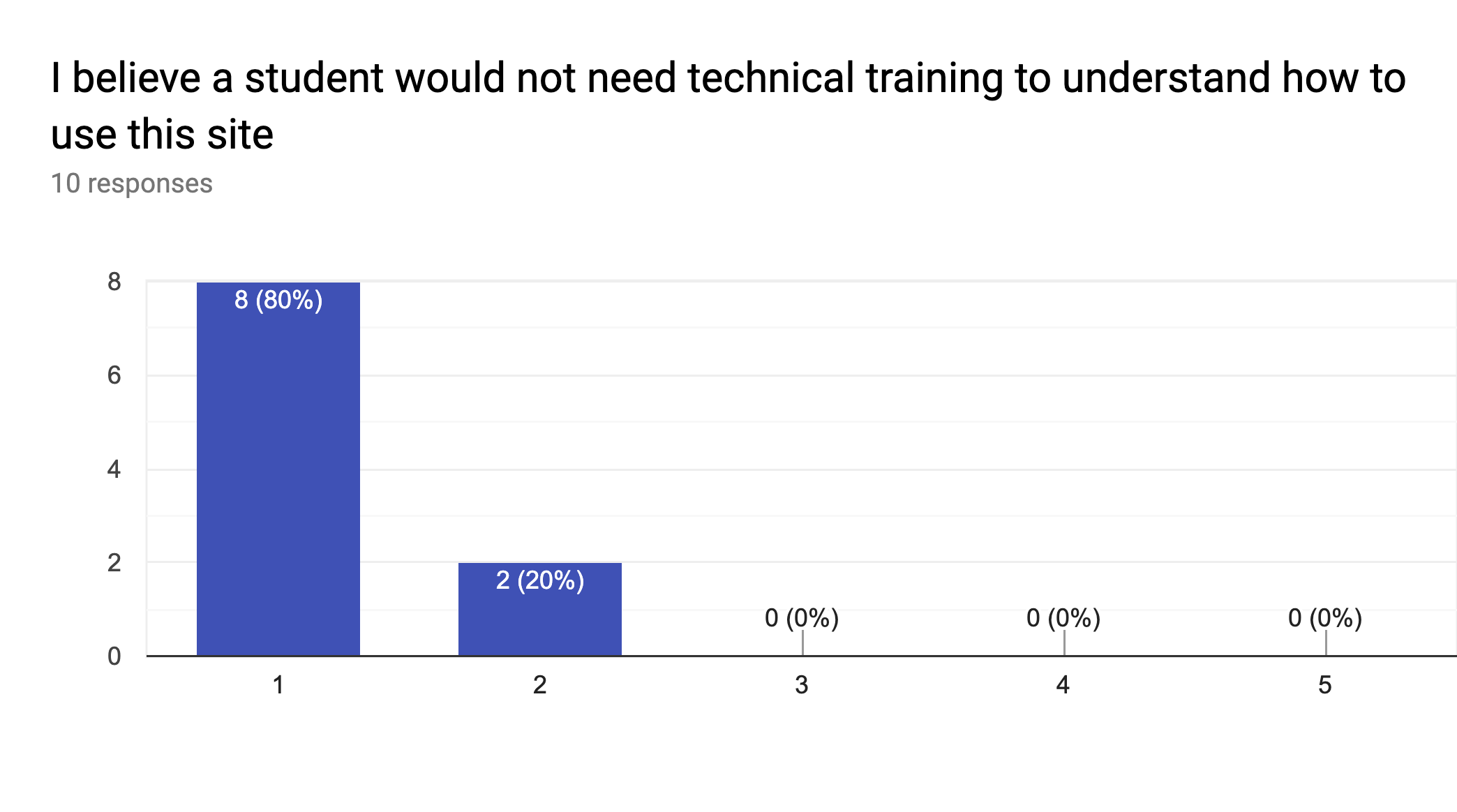


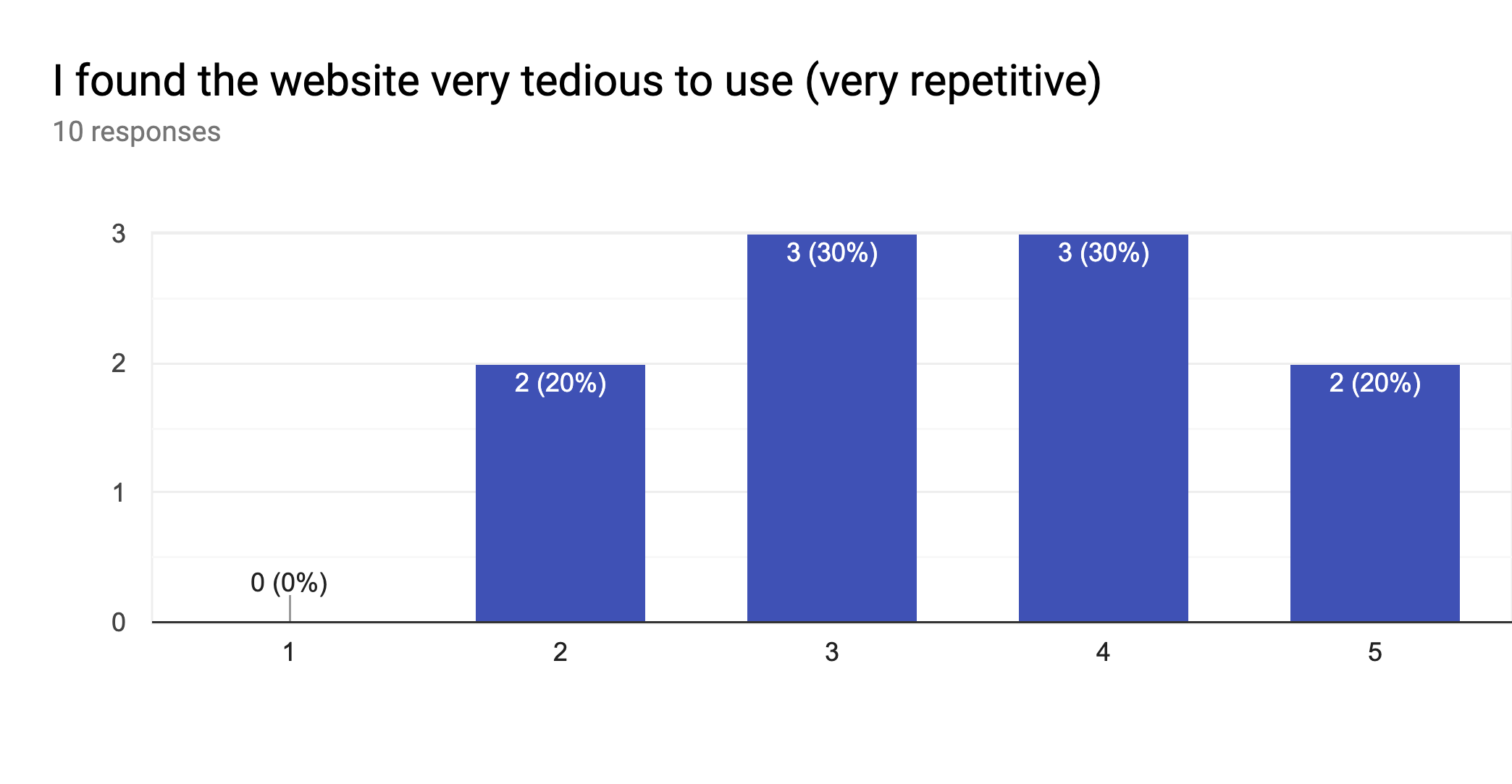
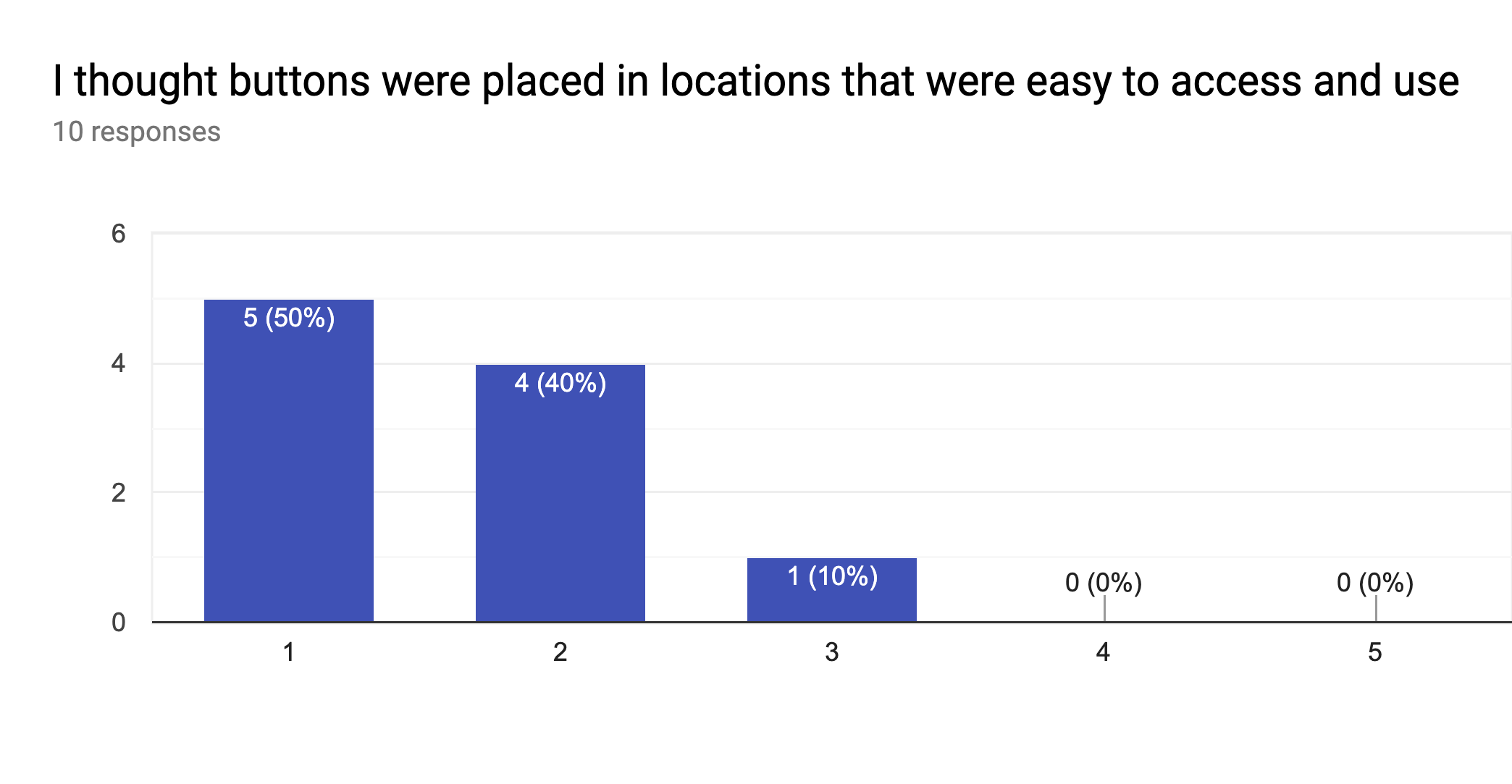


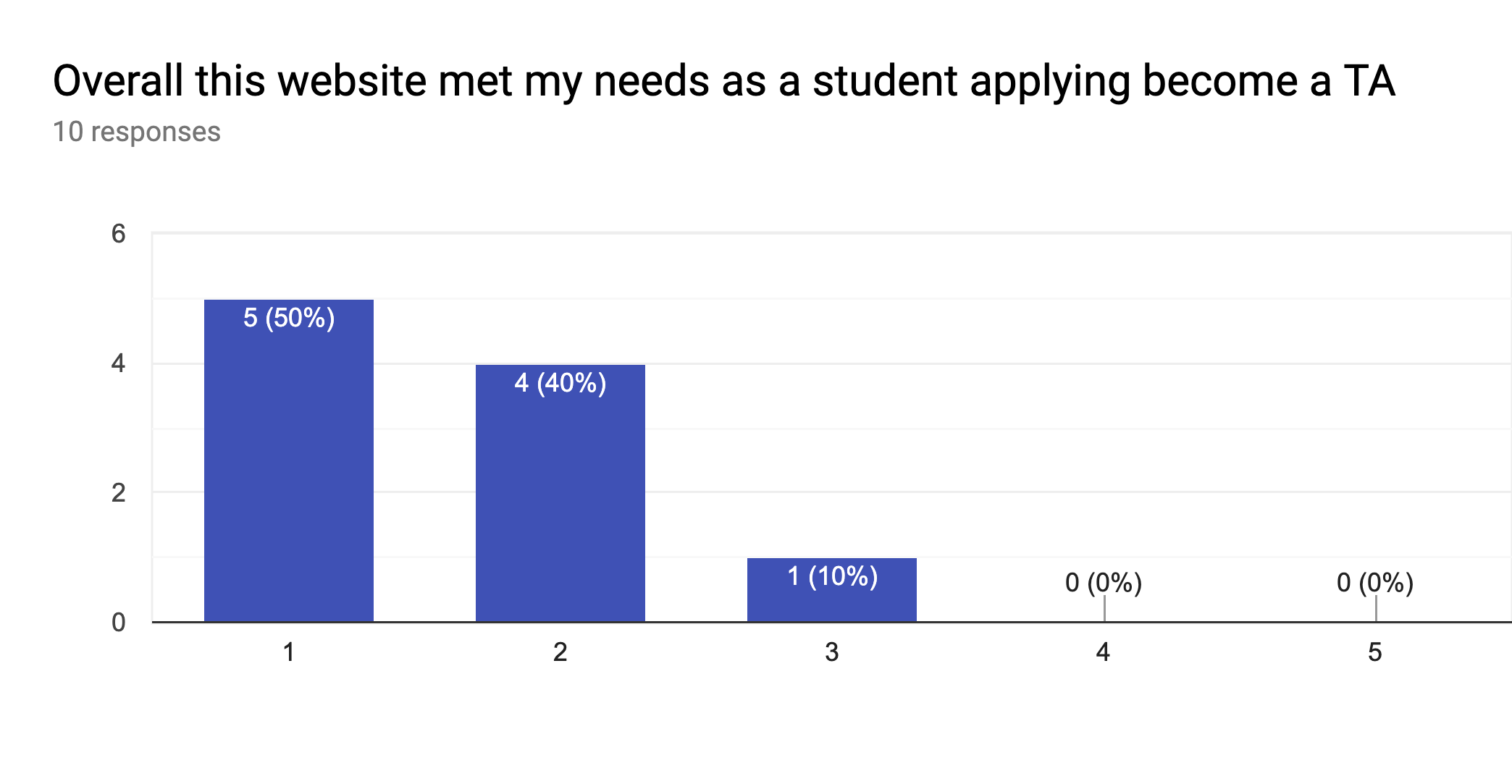


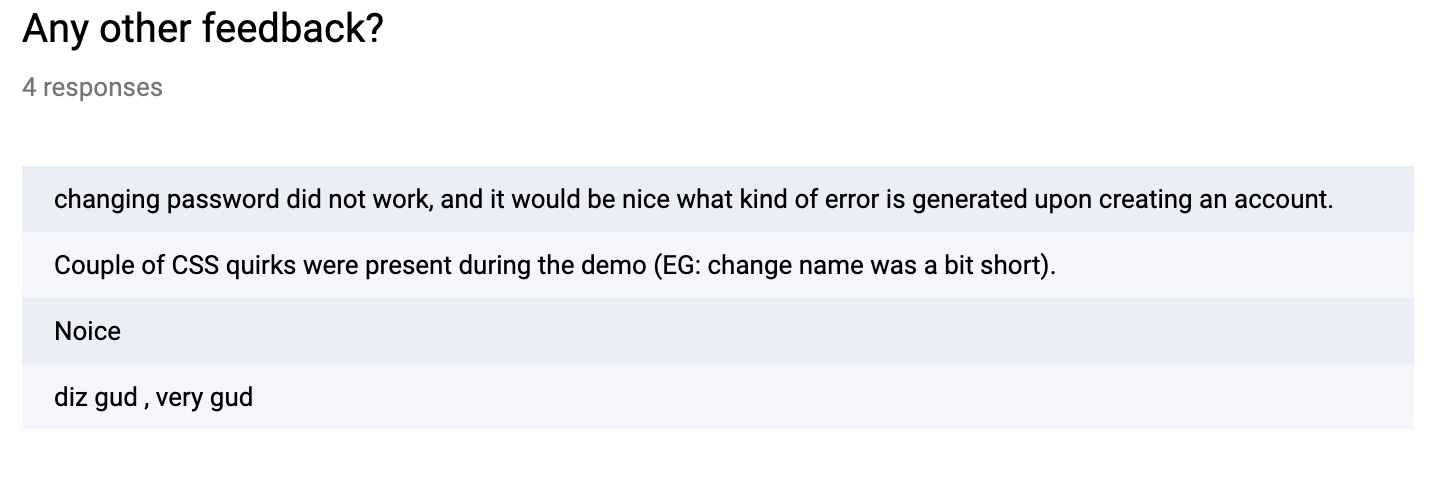
## Student Functions Feedback



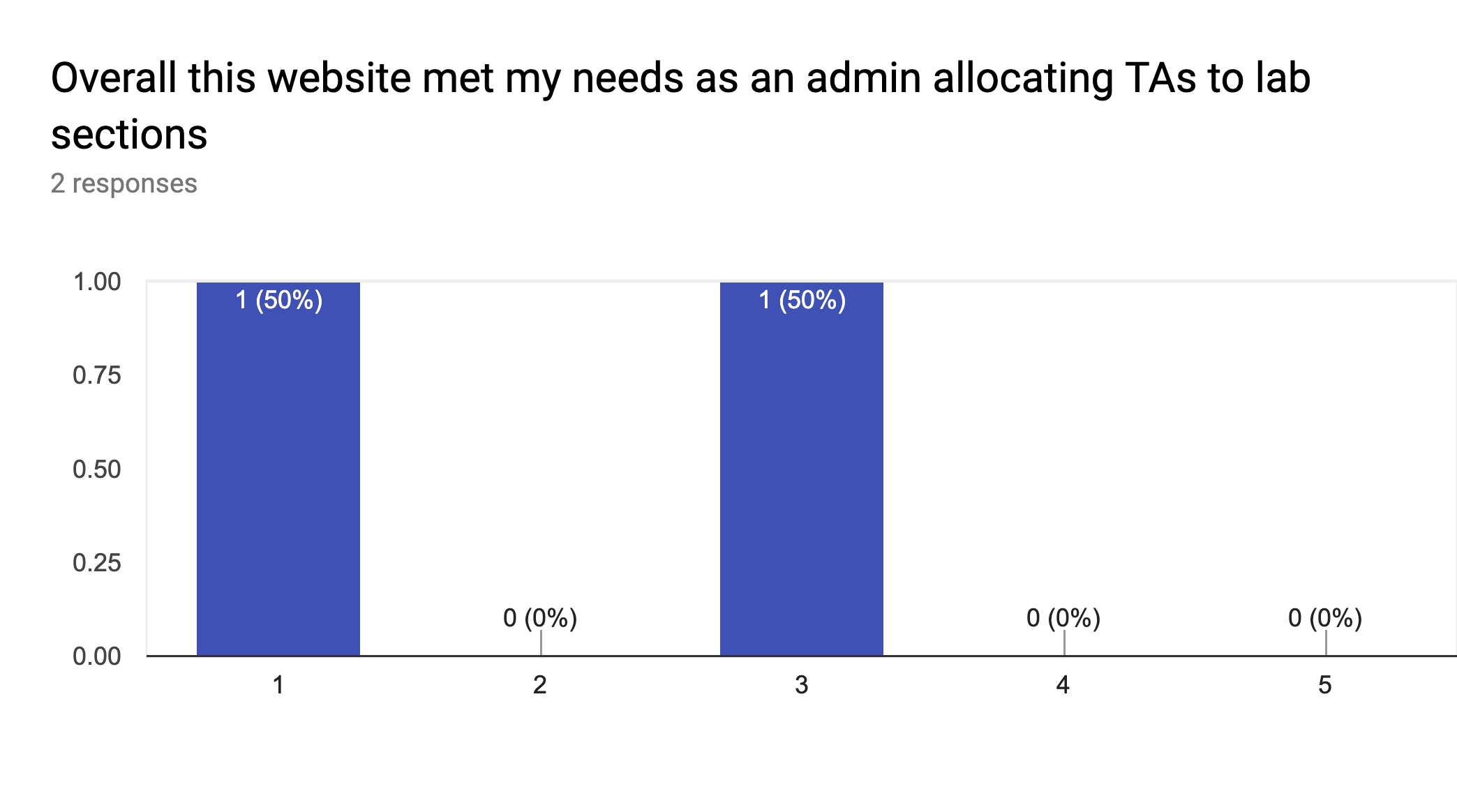
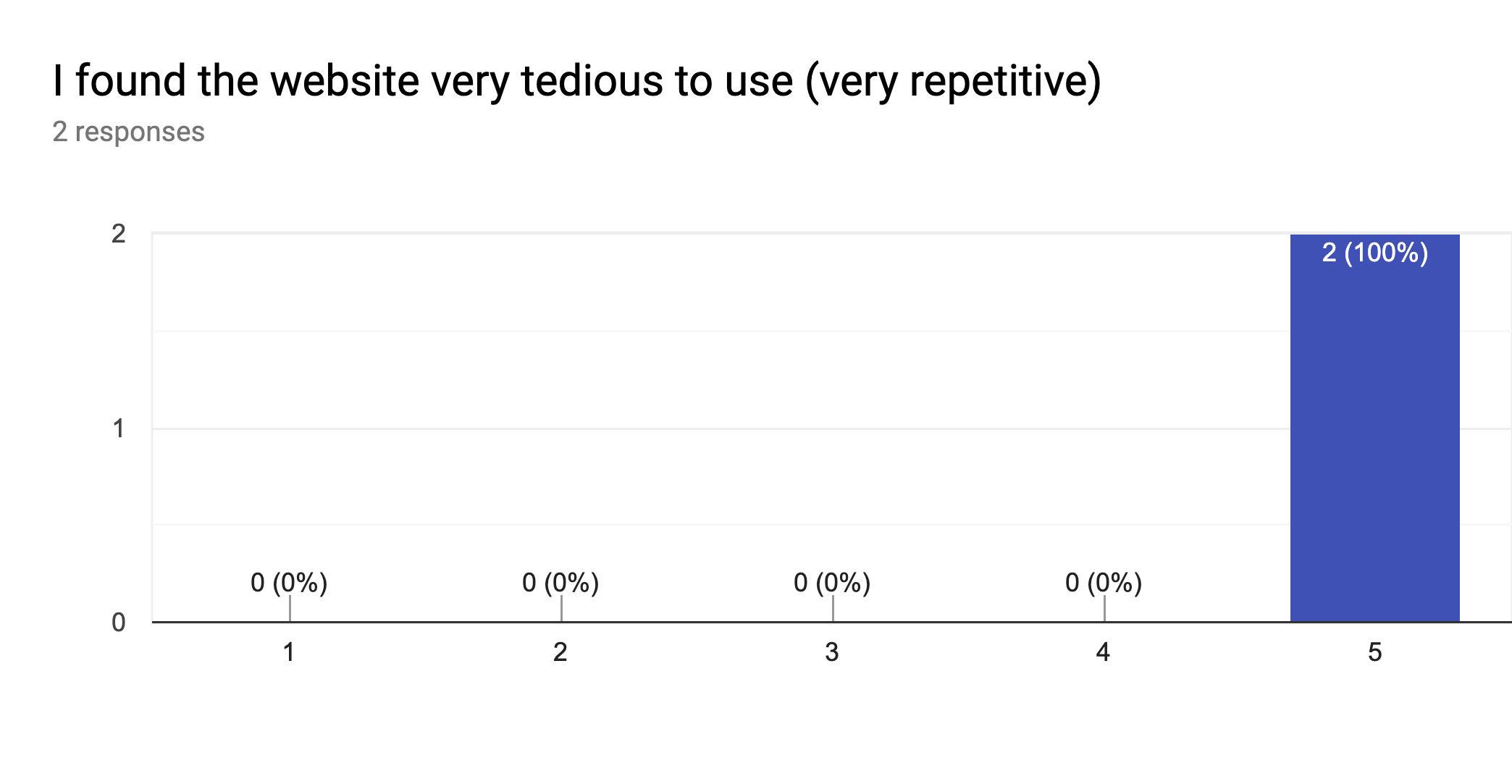
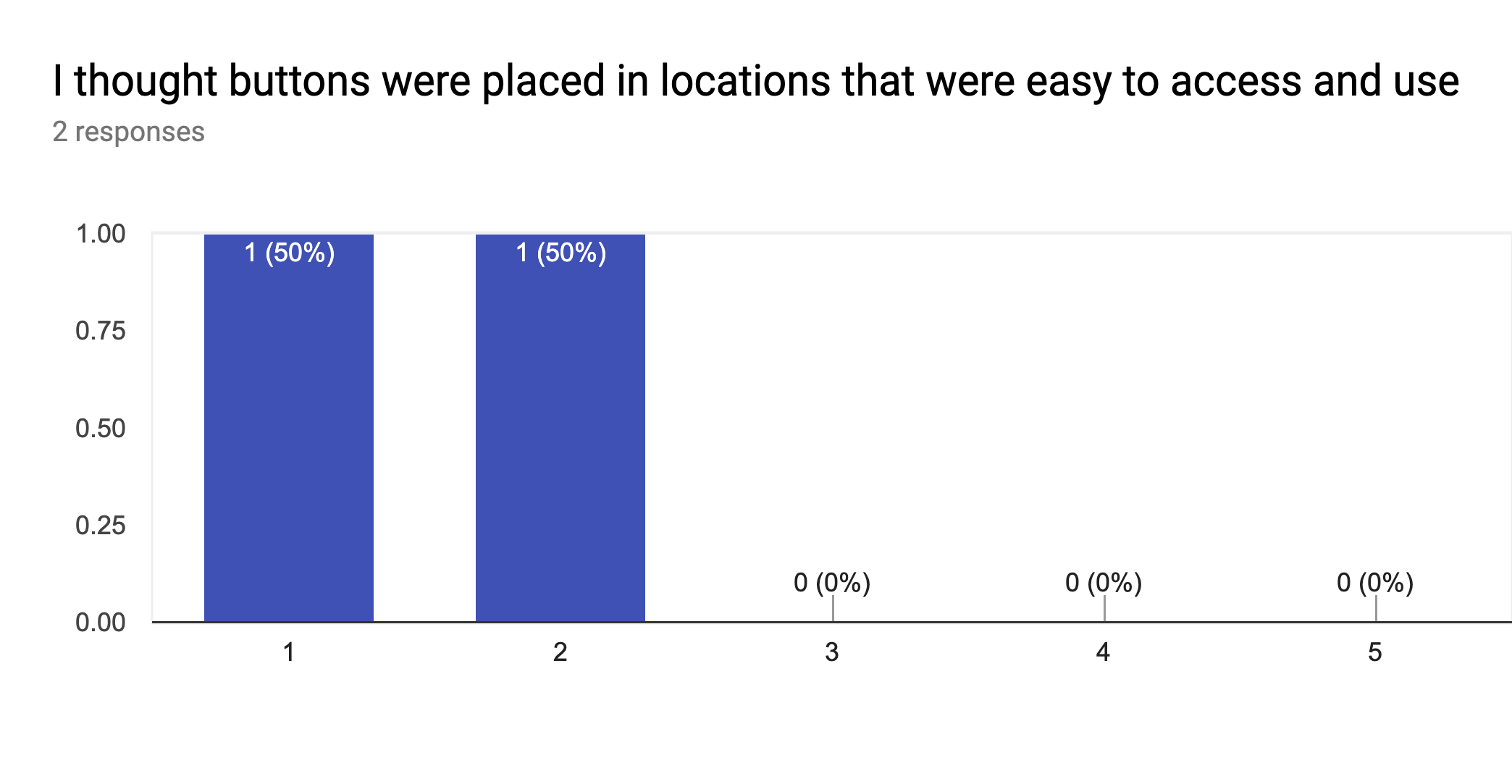
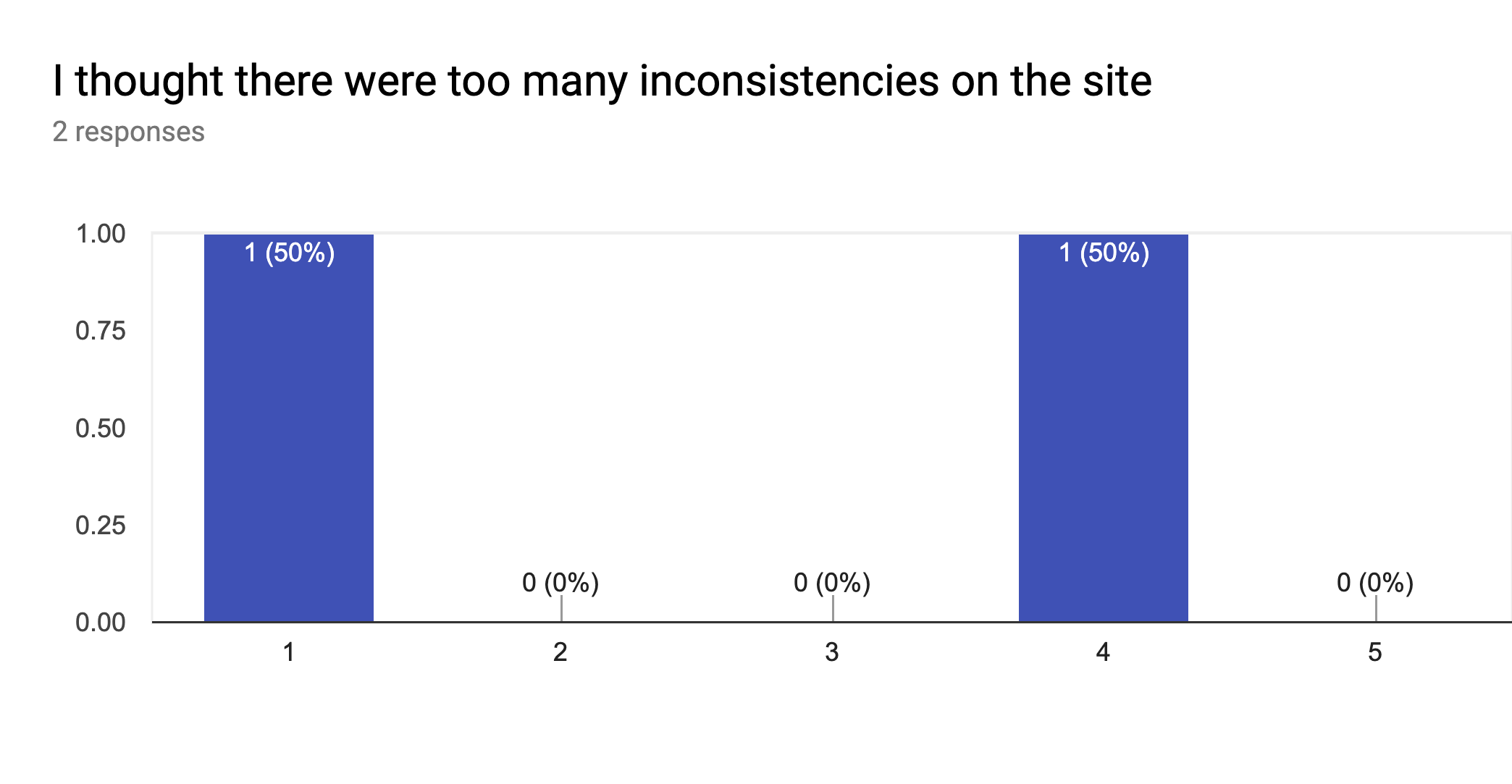
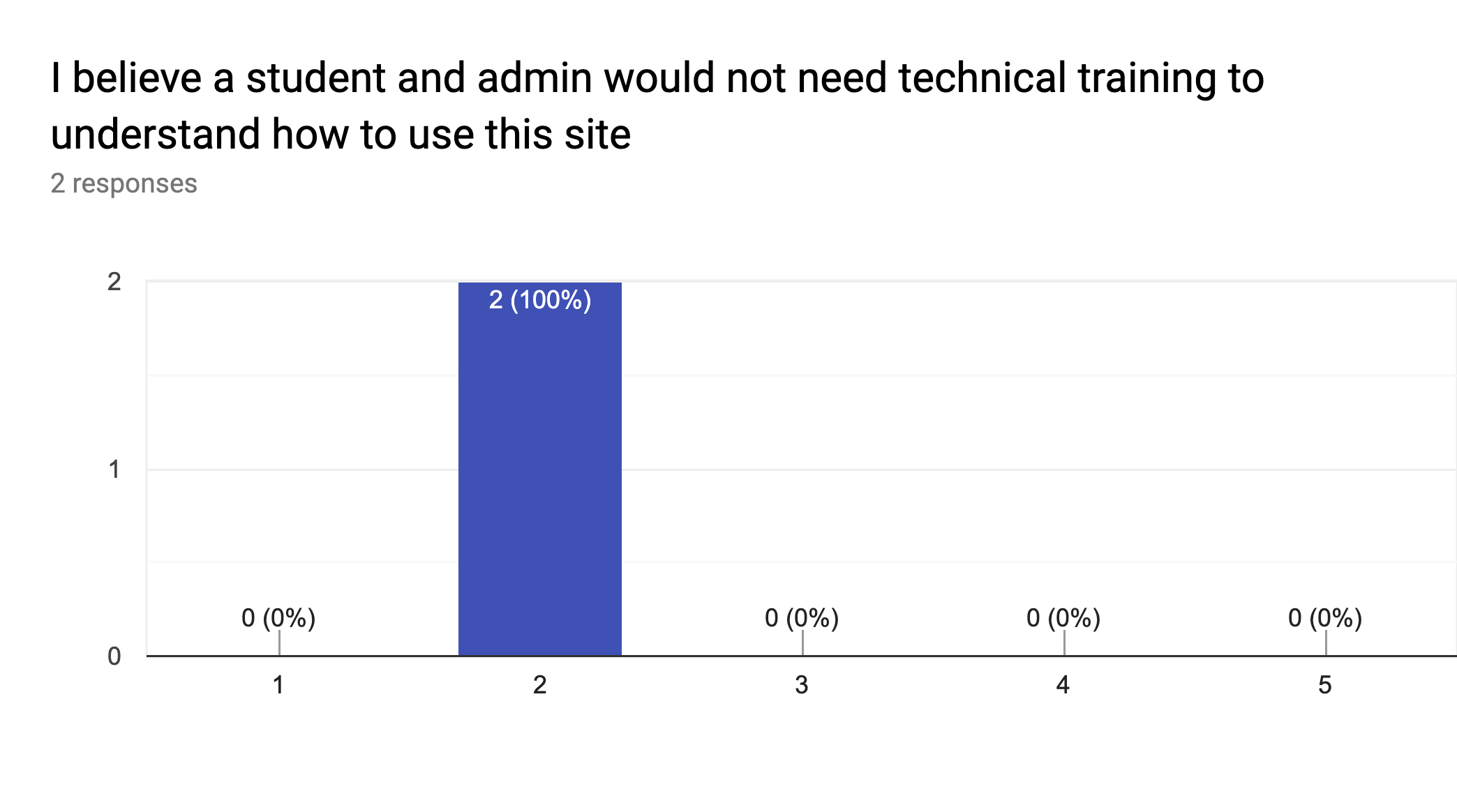
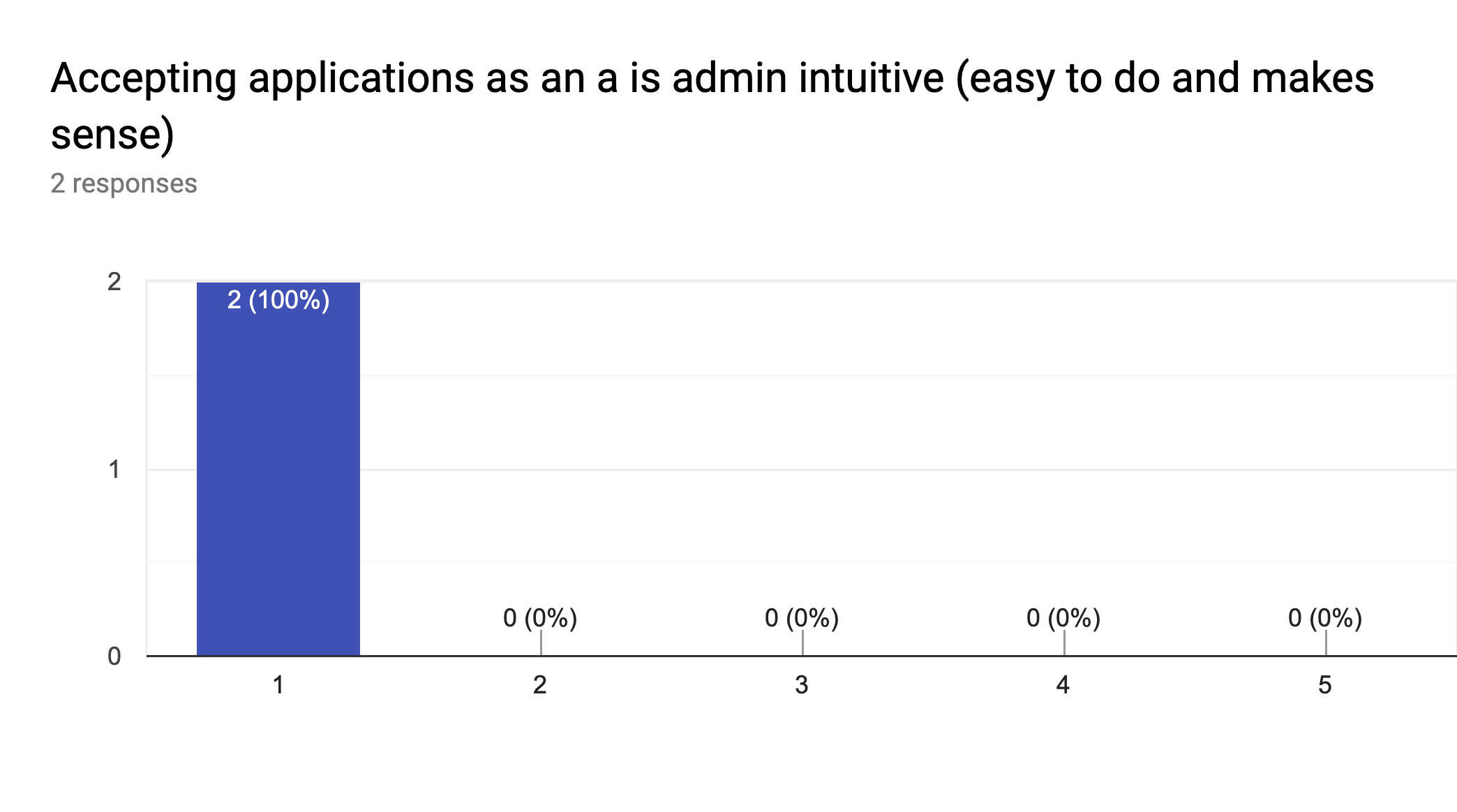
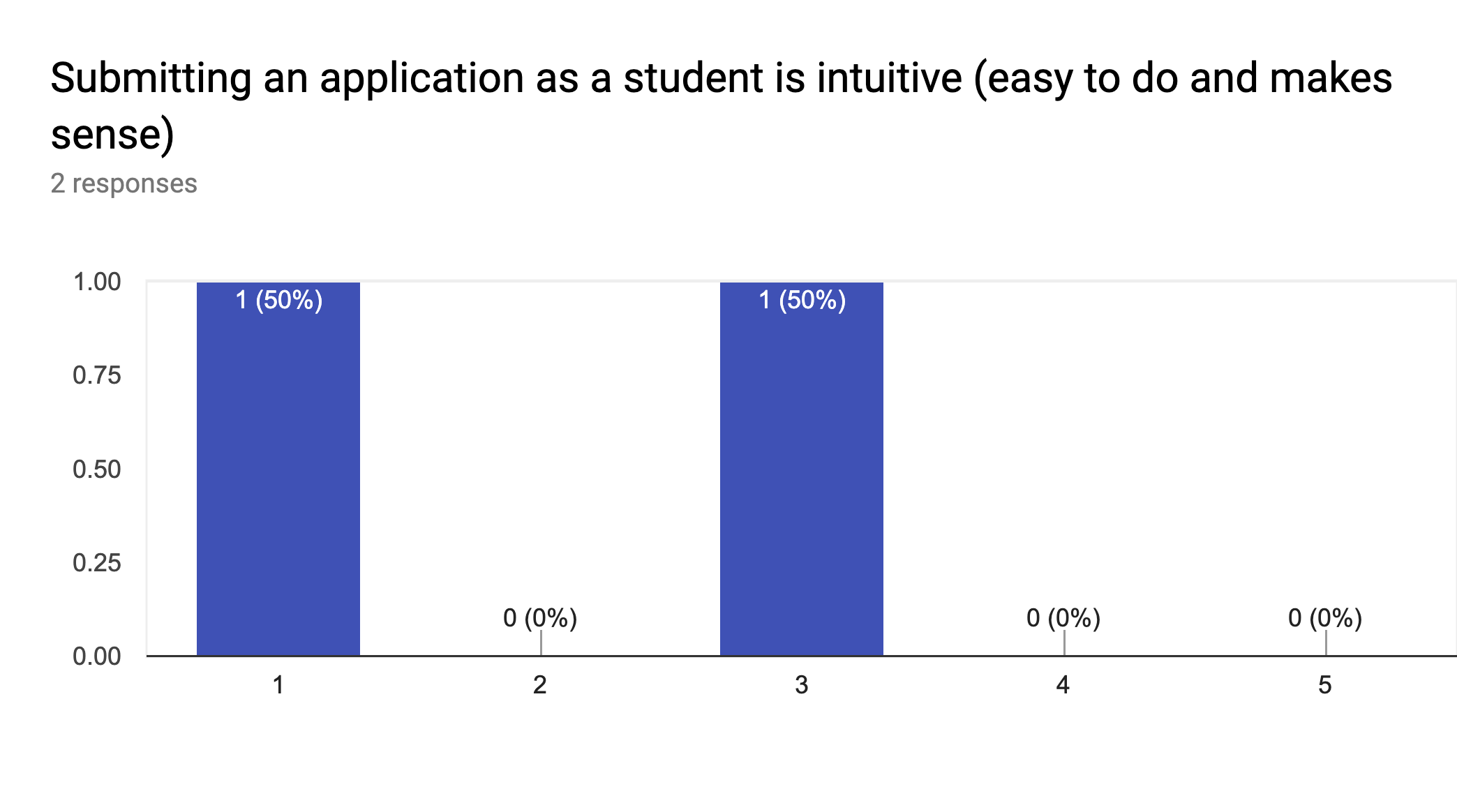
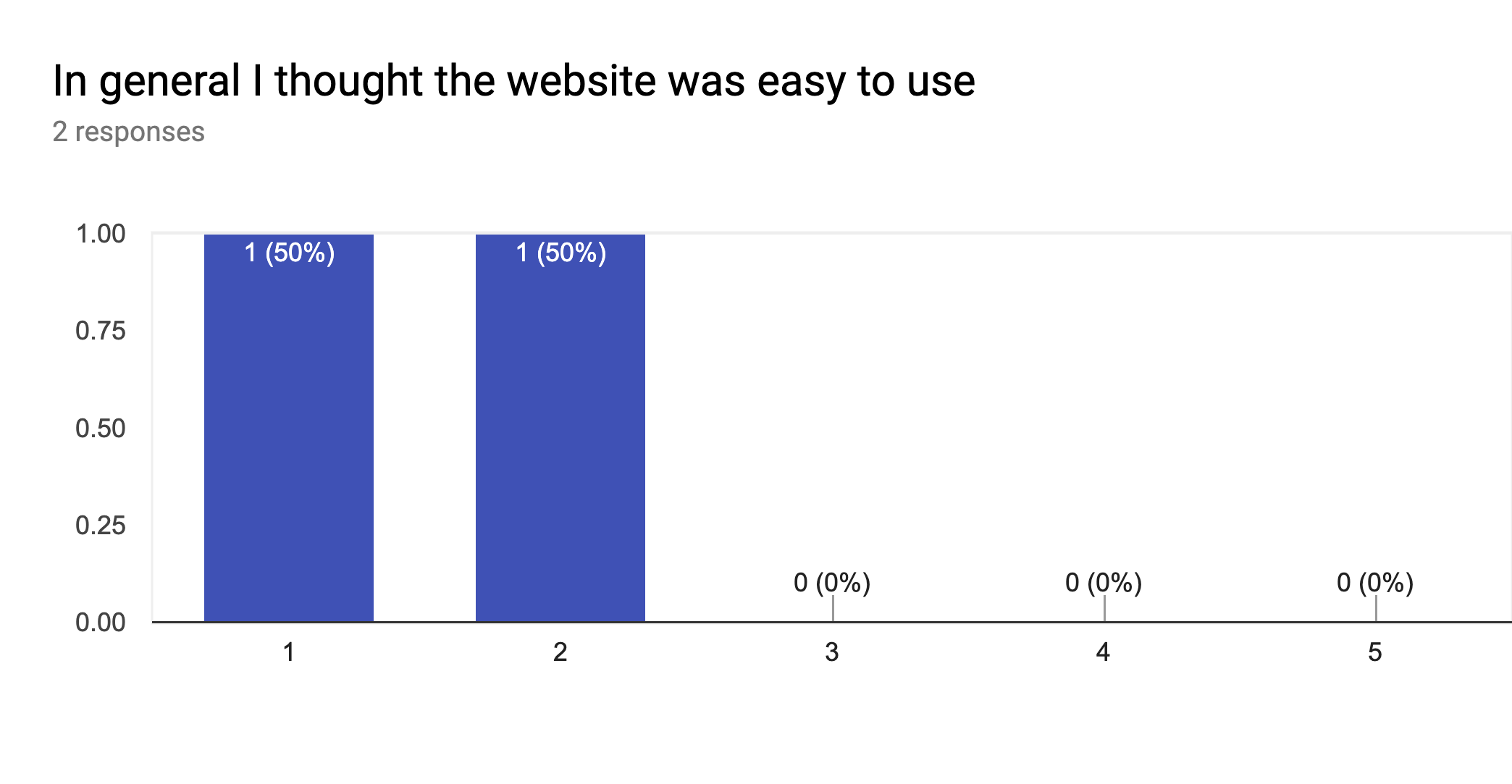


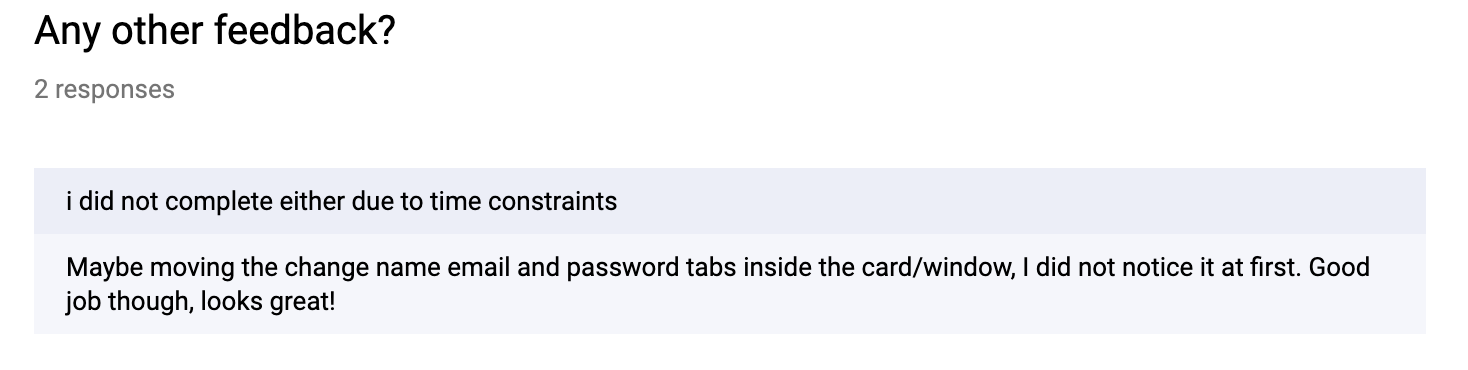






## Both Student and Admin Functions Feedback

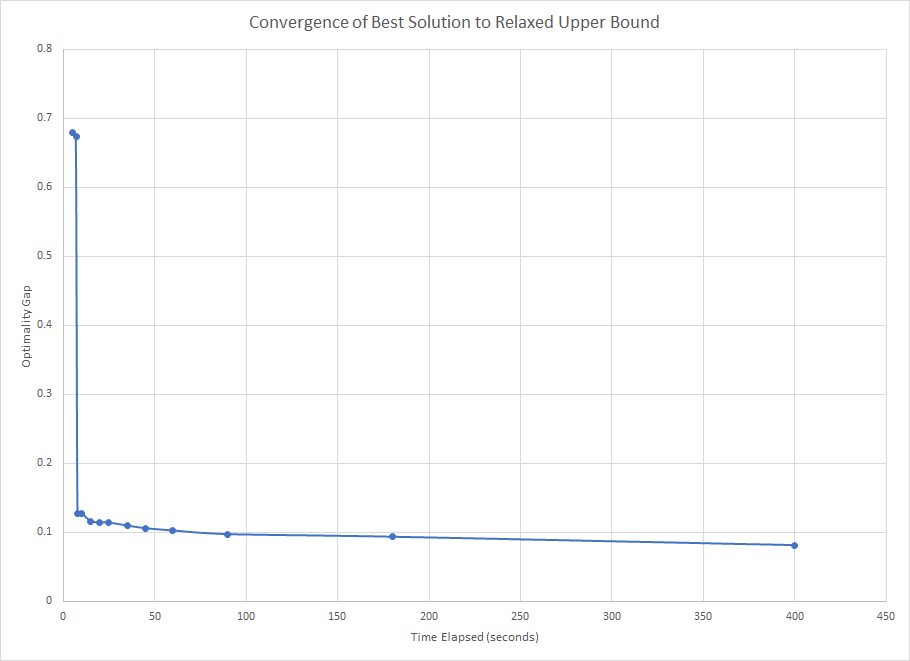




# Automated Scheduler

The automatic assignment of applicants to lab sections is modelled as an integer program (IP) wherein decisions are made on (i) who teaches a particular lab section, and (ii) who marks a particular lab section. These decision variables take on a value of either 0, meaning not assigned, or 1 meaning the particular applicant has been assigned to that section. The allocation of other hours, such as preparation, follows from these decisions and thus does not need to be explicitly modelled as a decision. On the course list page, there is a form to begin automatic scheduling which requires parameters specifying a time limit in seconds, as well as hour budgets. The default time limit, 45 seconds, will yield a simple feasible solution, while a longer run works to respect soft constraints and minimize such occurrences where undesirable situations take place. A short run, of less than 30 seconds is not recommended as the optimality gap is unlikely to converge in this time. *Plot 1* demonstrates this idea. A longer run will shrink the optimality gap further, but the rate at which this occurs diminishes rapidly. For a very good schedule, and provided there are sufficient applicants, a run of 90 minutes (5,400 seconds) is recommended.

When using the automated scheduler, it is vital to ensure that manual assignments do not render the system infeasible, as manual assignments do not prevent the administrator from exceeding the applicant’s maximum hours, which, if exceeded, renders the system infeasible with no explanation. These manual assignments are sent to the model as constraints forcing the user specified applicant to be assigned to a particular section. This keeps that applicant open to being scheduled into other sections, if possible, while respecting their total hours assigned.



*Plot 1: Demonstrates initial poor optimality gap of short 3 second run which rapidly improves.*

Upon starting the automated scheduler, data from all unblocked sections, as well information on all accepted TAs for that session is compiled and pipelined to a component where the IP model is defined. The website probes for updates through a status code, which stores what state the solver is in. This can range from idle, feasible solution found, infeasible, to errors on reading the data. This also ensures only one scheduler instance may run at a time, regardless of session and year, even if multiple administrators are operating the system at once. Here is a description of the model:

## Objective Function:

Maximize the number of labs filled while penalizing certain undesirable outcomes (to indicate preferences). The objective function penalizes (or awards) the following situations:

* Penalty for each TA added to course (to give preference to TA already assigned to course)
* Penalty for hiring UTA with no experience
* Award for hiring TA of higher year level
* Penalty for exceeding an applicant’s preferred hours
* Penalty for situation where TA teaching lab is not the same as the TA marking the lab

There are coefficients on each listed situation to allow for these to be prioritized in some fashion. Department concerns will have a higher coefficient - such as a penalty where the TA marking is different from the TA teaching a lab - while concerns of other stakeholders such as applicants will have a lower coefficient.

## Applicant Constraints:

* Applicant’s preferred weekly hours not exceeded (can be relaxed up to maximum with a penalty on objective function for relaxing)
* Applicant availability constraint - force a 0 on decision variable for this section if unavailable
* Applicant qualification constraint - force a 0 for all sections in course if applicant not qualified for this course
* GTAs: minimum weekly hours
* Preference to applicant with experience
* Preference to applicant with higher GPA
* Applicant cannot be assigned to teach two labs that take place at the same time

## Department Level Constraints:

* Hours for UTAs and GTAs do not exceed department budget
* TA can only get hours in a course if they are assigned prep hours

## Lab Constraints:

* A lab can only have a single TA assigned to it
* For a lab to be scheduled, both marking and teaching hours must be assigned
* A lab with a TA already assigned is sent to the model as a constraint forcing that TA be assigned

## Technical Constraints:

* Problem will be solved as an IP, with decision variables for both marking and teaching a section
* Indicator variables to handle changing cost (ie. adding on only one preparation block

As mentioned above, a TA assigned to too many labs such that their maximum hours is exceeded will yield an infeasible solution. There is nothing in place to denote what caused an infeasible result, and it can be difficult to determine, as it can be a combination of factors in some cases, so provided is a list of possible causes:

* TA manually assigned to too many sections causing their personal hour maximum to be exceeded
* TA assigned to course/section they are not available or qualified to teach\*
* GTA does not receive their minimum hours (could occur if not enough sections to schedule to, or too many promised hours and not enough working hours)

\*Although our system will not recommend a TA be assigned to a course/section if they are either not qualified or not available, it is important to recognize that this is a hard constraint which could be violated if for instance it was possible for a student to change their course schedule, and this change caused infeasibility.

The best tool to prevent infeasible results is to be aware of such causes. If the scheduler returns a feasible solution when no TAs are locked in already, but an infeasible result with TAs locked in, it could be that a TA exceeds their hours, or too many labs are filled with UTAs such that GTAs cannot achieve minimum hours, or a combination of both. If the scheduler fails to return a feasible solution when there are no TAs assigned to labs already, the cause is likely related to minimum hours for GTAs not being possible to meet. Overall, the automated scheduler has been designed to be robust - filling as many labs as possible given both the pool of sections and applicants, while giving the administrator the option to specify a time limit, which indirectly enables the administrator to specify how optimal of a solution is required, and give an expectation of when the schedule will be ready.

# Assumptions

## Assumptions

* The transcript a student uploads is honest and accurate
* Accurate course and lab list data is uploaded to the system
* Students will use a web platform form rather than the paper one implemented currently \*
* A student’s academic record is uploaded in a supported format
* A secure server within Canada will be provided (plus a domain to host on)
* UBC IT will deploy this

# Work Breakdown Structure

**REDACTED**

# Conclusion

The optimization component of the project was initially our largest risk and highest difficult component of the project. We are very proud to have finished that section. Overall it was a great experience that ran fairly smoothly all term largely in part to our excellent communication as a team.

## Key lessons learned from the project

Internal Communication: Strong internal communications led to a more pleasant work environment. When there is a strong sense of trust and harmony within the team it makes the team work harder and more willing to put in long hours.

Requirements Specificity: Many of the requirements felt sufficiently specific in the fall, but when reading them over in March we were often confused. If we had been more careful during the design and requirements phase we could have saved some time and redundant efforts.

Test Writing Maintenance: It is always difficult to maintain test writing during the peak development phases. However we learned our lesson about being diligent about test writing as it helps discover bugs and missed corner cases more quickly.

Time Estimation: We are lucky to not have a real budget, however if we had we would’ve been very over budget. As time went on we became better at estimating a task’s complexity. During the next project we will have a better idea of how long tasks take.

Client Communication: We were often apprehensive to approach the client without a major change to show them. However we’ve now learned more frequent check ins, despite a client’s busy schedule leads to a more detailed logs of what the client is looking for. Ensuring that we develop a product that is in line with what they expect and want.