Specifications: The server is run using a Python Flask server. This is a very easy and reliable way to host a server on a computer to be accessed by other devices. The UI is coded using HTML, CSS, and JavaScript (JS) to get a simple and yet responsive interface. The UI is controlled via an Xbox controller which commands which type of request will be sent. The requests are sent using XMLHttpRequests which are built into JS. The UI and the server will pass along data in a JSON format which is a very organized and easy to read way of passing data along. The Raspberry Pi's GPIO is managed using Python. The RPi code will connect to the server via Python's requests library. This library is simple to use, yet powerful. The robot's usage data will be passed along onto the server in JSON objects.

Usability without specialized training: The user interface should be easy enough to understand for the average user. It displays which part of the robot is moving with each button press. The only part that may require getting accustomed to is using the controller that is paired with the interface. The robot is controlled with an Xbox controller, which for users who have not had any experience using the controller, it may take some getting used to before using all the inputs instinctively. The controller scheme is very simple. Only 8 buttons will be used out of the controller's 16 inputs. The D-pad, which is the cross, will be used for moving the robot around. The triggers and LB, RB buttons will be used to control the bin and excavators.

Flexibility: The software can handle changes to the robot in many ways. If the button mappings need to be changed, it is as easy as changing a few lines of code (under 15 lines). The software does not need any major redesign when there are changes to the robot's hardware. The only thing that may require a change to the software would be the addition of extra components that need to be activated via a button. This means that additional button mappings would have to be coded in. If any existing components are removed, there is no need to make any changes to the code. The

server should be capable of communicating with the UI and the Raspberry Pi so long as there is some sort of LAN they can all connect to. Any device capable of providing such a thing, from a router to a phone, can be used.

Scalability: The code can handle additional components being added. The only thing is that as more components are added that will require a button mapping, these mappings have to be added to the UI as well as added to the JSON object that is passed back and forth. This does not take a lot of code and should be simple changes.

Maintainability: The code should continue to work with very minimal maintenance. The only thing that may cause the code to break in any way is the changes made to the APIs for the libraries that are used. If any of the methods in use are removed, it will cause the code to break. These things can be fixed by simply making the appropriate changes that would be specified in the documentation for the library's API. Another thing that may break the code is if any bugs are introduced into the platforms that are being used like the Raspberry Pi OS or the libraries. Some changes may be needed to work around them.

Reliability: The software will work under any condition. The robot should be capable of communicating with the UI so long as the robot, the device running the server, and the device running the UI are within range of the common access point (in our case, it's the router). If some part of the robot breaks, so long as it's not some vital component of the Raspberry Pi, everything should work well.

Redundancy: In case of failure of the UI code, the best thing to do it try to restart the UI and avoid calling the command that made it break. E.g. pressing B on the controller causes the JS code to break for some reason, then just restart the server and UI, then avoid pressing B until the

issue in the code is fixed. This kind of issue should be very rare to encounter since button inputs and requests to the server are thoroughly tested. In case that the requests are not going through and any HTTP (400|401|402|403|404) code is returned, this might mean that the server is not up. If the server is already up, check the IP address, the port, and the route of the server and be sure that they match the URL on the browser search bar. To make it easier to find these constants in the code, they have been defined as variables at the top of each script file.

- URL format: **http://{ip-address}:{port-number}/{route}**

Security: As far as safety goes in the code, the only layer of security is in the SSID and password for the network router. The server can be tampered with if you gain access to the network. In the case that the LAN network goes down, this does not mean that the server and all else is useless. You can still get the server back up and running so long as you can find another device to used as a soft access point to host another LAN. This can be your phone or a laptop or anything that supports hotspot, if not another router. The thing about the LAN is that we rely on it to be used as a direct intermediary and as a common place for all the devices to communicate so that you could be running the server on one device, the UI on another, and the robot code in another and everything should still work fine.

Performance speed: n/a

Accuracy of system output: ?

Resource utilization: The UI memory usage is recorded at a low of 15.1 MB while running in a Microsoft Edge tab. Memory usage seems to stay constant or just increase very slowly when not sending any request. While requests are being sent, memory usage increases at a rate of ≈ 3.0 MB/min after running a few tests. These tests consisted of sending requests at the maximum rate

of speed the UI can send them which is one request every 500ms which is the cap I set on it to prevent too many requests from being sent. Yes, it introduces a bit of lag between pressing a button and the robot reacting, but this will significantly reduce the usage of resources as well as prevent the usage of too much bandwidth. In the Python server, the program stays at a constant 54.4 MB of memory usage even with requests being sent into it.

Bandwidth: Maximum bandwidth would be at around 1.5 KB to 2 KB. This is the absolute highest recorded request size that was sent. At the time of this, the true expected average packet size should be at around 1 KB per request.

Wi-Fi Speed: If speed is the maximum bandwidth at our disposal, then it depends. We are using a router which supports the 802.11ac networking standard as well as devices which all support the same standard. This means that out theoretical maximum speed lies at around 1300 MB/s. Now, this is under perfect conditions, since we will be connecting to the router remotely, this will reduce the bandwidth slightly, but nothing to be too concerned about since the amount of data transferred is minimal in comparison to the bandwidth available even in less-than-optimal conditions.

Response time: Response times between the server and the UI interface lie at an average of ≈ 5ms for 20 requests sent. Request times tested at a low of 3ms and a high of 11ms. So, response times are almost immediate and there should be almost no lag between input and server response. The time it takes to get this information from the Raspberry Pi should be around the same if not a little higher. I would estimate that the final input lag between pressing a button and the robot reacting should be under 100ms. This is just a very rough estimate.