Tech Feasibility Report

ViceGrips

Western Digital Corporation

Mentor:

Volodymyr (Vova) Saruta

Team Members:

Ian Ambos, Ethan Baranowski, Tayler Skirvin, Joshua Melo

10/22/2021

Table of Contents

Introduction 2

Database Structure 4

Hosting Options 6

Secure Login Profiles 10

Virtualization 14

Front-End Technologies 19

Reporting Systems (Jira, BitBucket, Confluence) 23

Technology Integration 27

Conclusion 28

Introduction

In a technology driven society, people rely heavily on the operation of their devices. The public eye does not pay close attention to the specifics of firmware failures, most relying on a successful operation. Just like any other software, firmware will have bugs, and because firmware runs more critical parts of the system, the consequences of failure are larger. This is where validation comes in, a series of tests are performed to ensure that a new patch won't cause serious problems. Currently, Western Digital's validation process encounters issues, further exacerbated by transitioning to a remote workflow. Western Digital's validation process weighs heavily on meetings to coordinate and update each other, sometimes on minor issues, which wastes vital time. Improving this workflow is crucial, and is stressed in our solution. The goal of our project is to fix the above issues with VICE, a web application that will provide a one stop shop for techs and managers to keep track of what they need to do and schedule other issues that need to be fixed.

To make the VICE project possible, intensive planning and work are required. VICE must be able to follow the Validation as a Service (VaaS) model for SSDs onto the cloud. The general expectation of the VICE MVP is the ability to store and schedule workstations that are to be used for testing purposes, and virtualize the task of loading a firmware image onto a Solid State Drive remotely for testing purposes. We additionally need to implement certain analytical capabilities that process and output the results of the different testing suites in a few different ways.

In order to 'store' these workstations, we need to create a database to populate with the information about these machines. This involves having functional capabilities that allow us to read from, delete, and modify observations in the database. The database and capabilities will need to be implemented on the cloud.

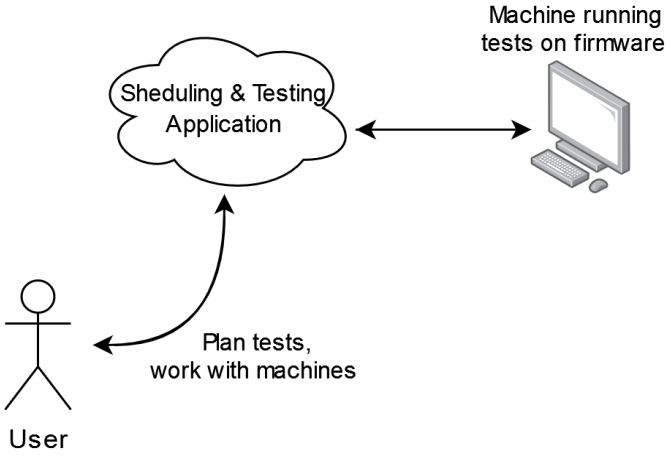
To create a scheduling tool to ensure we don't run into conflicts, we need to make use of this database, and utilize these basic functionalities. The scheduling tool also needs interactive user interface capabilities, and will essentially be the frontend of the project. In order to better understand the basic idea of our task, **Figure 1.0** below shows a simple representation of the problem.

Analytical capabilities will need to be developed once we have access to the testing environment and can better understand what outputs the suite delivers. Some nice-to-haves include using machine learning strategies to interpret what failures happen most often with what test plan, and using the results to move forward with future testing cycles more weighted towards those more probable failures. More realistically, our application will have the ability to export statistics (.csv format) that will allow for further analysis on possible optimizations of testing plans.
The outputs of testing on the virtual machines will need to be transported to multiple different libraries based on Western Digital's needs. Specifically, the issue

tracking platform JIRA will most likely be utilized to aid in discovering the most problematic combinations of machine, software, and firmware. Additionally, we will need to be able to actively monitor when issues occur, and notify those responsible for addressing said issues. The higher-level structure of our web application will need to be able to satisfy these requirements:

- Register all existing and new hardware testing platforms on a database (Store database of different machines available)
- Check the status of a platform before or after testing phases
- Allow 'virtualization' of particular platforms, adding the ability to check out a platform, while loading a software operating system image onto the SSD, and running testing suites
- Workflow "cycles" for new firmware - a validation manager tool that allows registration of SSD subsystems and >= 1 test plan (device + OS + test suite) developed for the SSD. We should be able to "launch" cycles for specific test plans, and have them run and produce output.

Figure 1.0



Users scheduling test cycles, planning tests which are input onto firmware machines.

In order to maintain a structure for keeping track of how many machines are in use and for what testing plan, we are going to need to implement a database system.

Database Structure

We are going to need a way to store all the data coming from our client and any data generated from the validation cycle. It remains important that we design our system with the back-end in mind, otherwise we may end up with implementation issues later on.

In our database we need to implement some form of scheduling system, wherein those in the network can check for availability of machines and different test suites.

2.1 Desired Characteristics

- Storage Used: Unnecessary wastes of storage must be avoided to decrease any potential costs caused by a multitude of firmware patches and different types of VM's.
- Variety of Integration: The database must be able to communicate with our chosen back end solution in sufficient manner (further discussed below in 'Secure Login Profiles' section).
- Ease of Use: A combination of a more intuitive system and an easily accessible language will allow for future development as well as faster on-boarding process.

2.2 Alternatives

2.2.1 MongoDB

A noSQL database system was created in 2009 by MongoDM Inc, it is used in a variety of areas. It is one of the more popular noSQL databases, commonly used in the MEAN stack for web applications. It used binary JSON files to store its data. They have some of their own systems for new users to set up small databases for free through their service and server versions for larger companies. It is very scalable and has a variety of backend languages it supports.

2.2.1 PostgreSQL

An SQL database system that was initially released in 1996 by Michael Stonebraker. It was written in the C coding language and is an open-source object-relational system. PostgreSQL is mainly used for its speed and strong security while also accomplishing what the user and application needs it to do.

2.3 Analysis

This section is focused on the advantages and disadvantages of different types of database programs and models. Furthermore, the software integration ability will be quantified, compared, and discussed.

MongoDB is a source-available cross-platform document-oriented database program. Document oriented databases are used when the data that needs to be stored shouldn't be stored in table form, but rather in full document form. MongoDB is classified as a NoSQL database management system (or non-relational database), and uses JSON-like documents with optional schemas.

PostgreSQL is a free and open-source relational database management system emphasizing extensibility and SQL compliance. Relational databases are useful for storing predictable, structured data with a finite number of individuals or applications accessing it. SQLite is another relational database management system contained in a C library. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program. SQLite generally follows PostgreSQL syntax.

In order to maintain a database and said scheduling capabilities we are going to need a way to remotely update our system. Allowing the participation of multiple sources of information to maintain and update the current availability of machines, testing being done, recent errors etc. will require a hosting platform.

2.4 Chosen Approach

One of the critical environment requirements for this project is use of Microsoft's Azure Table. Azure Table storage is a cloud-based NoSQL datastore that can be used to store large amounts of structured, non-relational data. Azure Table offers a schemaless design, which enables us to store a collection of entities in one table. This will make it easy to add and pull data from the database.

2.5 Proving feasibility

In order to prove the feasibility of the database system we will further discuss Microsoft's Azure Cloud hosting environment. Using Azure Cloud along with Azure Table storage will be relatively easy since they are cointegrated.

Hosting Options

Our system is going to need to have interactive capabilities that allow for task scheduling, running specified test suites on allocated Virtual Machines, and the ability to access our database from different devices. To account for these requirements, the best option is to utilize a cloud environment for hosting our web application because we do not already have infrastructure in place to host on.

Because we are working in a cloud environment, we have a few important quality of life constraints that we need to enumerate. We want to keep these criteria in mind:

1) The amount of support available / size of the user base - if we run into bugs we want to be able to figure it out, and have answers quickly and effectively.

2) A variety of hosting options (regarding the specifications of the machine running) depending on our computing needs - we may want some machines optimized for different tasks than others (based on processor speed, memory amount, etc.).

3) The cost of hosting - A few extra cents an hour on a system that could be running hundreds of VM's can add up to much more per year.

Maintaining a web application will be much easier in the long run when there is a plethora of online resources available to aid in troubleshooting. Typically the more widespread a specific module is, the more likely there is going to be a large amount of support for it. This is invaluable when running into any issue.

Based on what our web application will be computing, we want to consider utilizing different types of machines in the cloud to implement our program. We will weigh the different options when comparing the different cloud environments.

The cost is arguably one of the most important factors in deciding which cloud platform to host on. If the company utilizing our web application needs to host it on an expensive platform, those costs will add up very quickly which is unfavorable..

3.1 Comparison of different Cloud services

Nowadays, there are multiple cloud hosting options available that each provide similar service models. Due to the similarity of these platforms, we can break down the different cloud technologies by comparing their main characteristics based off of Computation, Storage options, and Billing
Let's compare AWS and Azure keeping our goal - using VM's to virtualize the firmware testing process of SSD's - in mind.

Since both cloud hosting options are fairly widespread, we don't necessarily need to worry about the user base / support system due to their prevalence.

Both companies provide the ability to utilize machines optimized for specific uses. **Figure 1.1** compares the different services, wherein the number represents the specific quantity of machine types offered by each company. In the first row for example, Azure provides 13 different options to choose from, while AWS provides 10. "Optimization" in the table refers to the purpose that the machines are specifically tuned for.

Figure 1.1

Optimization	Azure	AWS
General Purpose	13	10
Memory	19	9
Compute	3	5
Storage	2	4
GPU	10	7
High Performance	4	N/A
Sum	51	35

Azure seems to provide more options depending on your specific need in this regard.

One of our goals is to maintain a database containing all machines used for validation testing, as well as the operating system and firmware that we want to run on said systems. In this case it's important for the cloud service to be capable of hosting and implementing a database system reliably and smoothly.

Azure provides various Relational Database Services that will easily allow us to populate our desired database and maintain a scheduling system. Additionally, many data analytics libraries are available to assist in our goal of keeping track of

which combinations of machine / OS / firmware tend to have the most errors, and for which tests specifically.

Another important aspect of our cloud options that we need to consider is the price.

Figure 1.2

A chart comparing on-demand computing costs of Azure vs. AWS in prices per hour as of 2020:

TYPE	VCPU	MEMORY	AZURE COST	AWS COST
General Purpose	2	8GB	0.6260	0.6680
	4	16GB	0.8520	0.8560
	8	32GB	1.7040	1.7120
Compute Optimized	2	4GB	0.5780	0.6490
	4	8GB	0.7980	0.8180
	8	16GB	1.5960	1.6360
Memory Optimized	2	16GB	0.6880	0.6850
	4	32GB	0.9760	0.8900
	8	64GB	1.9520	1.7800
SUM			$8.3336	$9.694

Here we can see Azure appears to be the option with the most variety and the least cost, so of the cloud options it seems the most appealing.

Self Hosting Option

An alternative to cloud hosting options is self hosting. If we were to choose a self-hosted web application, we would have more control over the encompassing implementation details, and the costs associated with hosting would most likely be less compared to utilizing the cloud. This could provide security related benefits, as

well as the bonus of having the entire application handled within the company's walls.

There are some drawbacks to self hosting however, that we must keep in mind:

1) It is not as flexible as the cloud - we would be missing out on several tools and quality of life implementations provided by the service - adding the task of building the modules that we need from scratch.

2) We would be missing out on a safety net of disaster recovery options provided by the service, potentially putting the system and clients in danger.

3) The processes implemented in our self-hosted application may not be as efficient or robust compared to the cloud implementation. It would require more effort to get working. Keeping the limited timeframe of capstone in mind, this on top of the already existing problem does not seem like a feasible goal.

Chosen approach

Due to the disadvantages of self hosting, we will most likely be choosing the cloud option. Furthermore, any company using the web application would be linking a large amount of proprietary data, so security is essential.
In the cloud services, AWS and Azure were considered, as seen in **Figure 1.1 & 1.2** the flexibility and cost of Azure is a better choice. Azure and some of its modules have also been requested by our client.

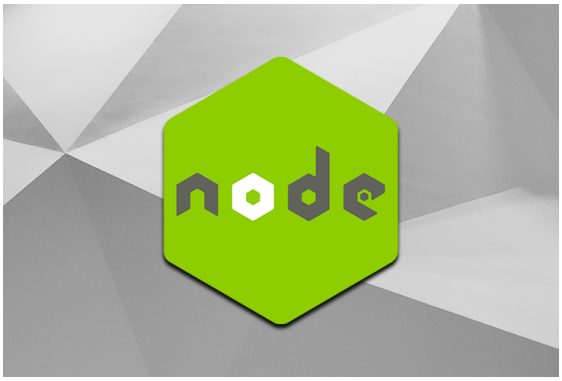
Proving feasibility

In order to prove the feasibility of hosting our webapp in the cloud, we are going to further prepare by working with Azure to implement remote interactivity with our machine database. We may possibly utilize webhooks as well to allow for operations to be performed with the database.

Secure Login Profiles

While there are plenty of options to explore when expanding security across a web-application, first we must discover the essential needs required to meet the specifications of VICE. To provide this security, we need to understand the importance of authorization and encryption. Encrypting user login information is crucial in maintaining user privacy, therefore opting for an up-to-date back-end architecture is necessary.

4.1 NodeJS



To start, NodeJS is flexible and easy to incorporate; it's strength resides in back-end architecture. For example, creating a log of user registration and encrypting passwords is fairly straightforward and is practical for our usage. To do this, we add the Express framework on top of NodeJS, accomplishing the secure profile authorization goal set earlier. Downfalls of NodeJS come when performing CPU heavy operations, but luckily, processing for SSD validation will be housed at WD. Our implementation will come through various API calls to our application database. This, in part, makes the back end not responsible for performing large scale processing.

Practical use cases of NodeJS also fit with our front-end goals, adding the strengths of API calls. Queued Inputs, Data Streaming, and using an API on top of a database object is where NodeJS shines. This adds onto the request of our client, as it was also mentioned that utilizing a REST API would benefit the VICE web-application.

4.2 Alternatives

The plethora of options to research on application security would be an endless list, but below suggest popular options for accomplishing the same task. When analyzing each architecture/framework, considering the risks associated with each is important in diluting this list.

4.2.1 PHP

PHP was developed in 1994 and was the most popular side scripting language used on websites. It allows connection to almost any type of database while also being easy to learn. PHP is an object oriented language as well, but with this, there still are some drawbacks. PHP has a lack of structure and is often inconsistent by today's standards. Maintaining PHP is far more difficult than other options, often leading to deciphering spaghetti code.

Most importantly, PHP has a bad reputation for the security vulnerabilities that are all too common. It is prone to SQL injection attacks or even XSS (cross site scripting). The list unfortunately doesn't end as session hijacking, remote file inclusion, and cross site request forgery are all problems PHP has encountered in the past.

Thus, concluding that PHP seems outdated and runs too many risks of exploitation. Although it remained a popular route to take, the downfalls of PHP overtake the positives.

4.2.2 Ruby / Ruby on Rails

Ruby is a popular choice as well in terms of a general purpose programming language for web development. With the addition of Rails, this provides the ability to construct an application with less time constraints. Ruby on Rails is a server-side framework which is built on top of the language Ruby. Additionally, it has integrated API modules to apply services faster.

Generally, Ruby would be a valid option to choose for development purposes, but the learning curve of Ruby gets in the way of project progression. Sticking with JavaScript is more ideal for our projects as it would get the same job done, without needing to learn a separate language.

4.3 Analysis

To further validate our decision in using NodeJS, a simple password encryption program was created to demonstrate the ease of use Node provides. With the use of Express and Bcrypt, user password information is easily manipulated, hiding the contents of their account password.
Figure 1.3 - Password hashing with NodeJS

```
// Post will make a new users, it also hashes passwords with 'salt' for
// security puporses
app.post('/users', async (req, res) => {
    try{
        const salt =  await bcrypt.genSalt()
        const hashedPassword =  await bcrypt.hash(req.body.password , salt)
        //console.log(salt)
        //console.log(hashedPassword)

        const user = {name: req.body.name, password: hashedPassword}
        users.push(user)
        res.status(201).send()
    }
    catch{
        res.status(500).send()
    }
})
```

As illustrated in **Figure 1.3**, this simple program is used to apply password encryption to a user object. This is similar but not recommended for use, as this example uses a local structure to store username and password information. Ideally, this would be replaced with an external database object for added security. To perform these actions, “bcrypt” is used to generate “salts” or a randomized string of characters, hiding the inputted password.

4.4 Chosen Approach

In summary, there are various options for providing back-end security, although those listed are typically presented with a learning curve, security concerns. Node provides the easiest adaptability, sticking with traditional JavaScript and illustrates a security focus.

Figure 1.4

Technology	Security	Learning Curve	Total

NodeJS	5	5	5
PHP	1	3	3
Ruby / Rails	3	1	4

Scale from 0-5, worst to best.

4.5 Proving Feasibility

NodeJS against other competitors, illustrated a clear and logical option to other alternatives. It provides an adaptable and easy to learn structure that is well suited for the VICE application. API integration and event driven IO will directly correlate to our task for secure profile authentication.

Our integration will not rely heavily on CPU processing, adding more reasoning behind this decision. Integration with the front end will not be a challenge with Node as it is flexible with other libraries as well. Another important aspect that Node will have a part in, is its connectivity with the core of our web application, thus being integration of Docker.

Virtualization

Because we have to run firmware remotely on various different machines, we need to think about using VM's or Containers. They both can have significant impacts on the usability of our product in different ways. We need a system that will have reasonably low overhead and allow validation to be thorough. The speed needs to at least match that of WD's current system. For validation, we need a system that will allow any system calls to be carried out in their original state. If a system call from a Ubuntu guest OS gets replaced by a Windows system call because of the hypervisor, we are no longer validating the hardware for Ubuntu.

5.1 Alternatives

Desired Characteristics

- Ease of use: good documentation and developer support is key for keeping development time under control.
- Speed: the system has to operate quickly so time spent processing doesn't add up.
- Large range of OS support: firmware validation has to be checked across all possible OS platforms, it will not be viable to test on only a few.
- Scaliable: as this product could be used by many companies, being able to handle many users is a must.

5.1.1 VM's

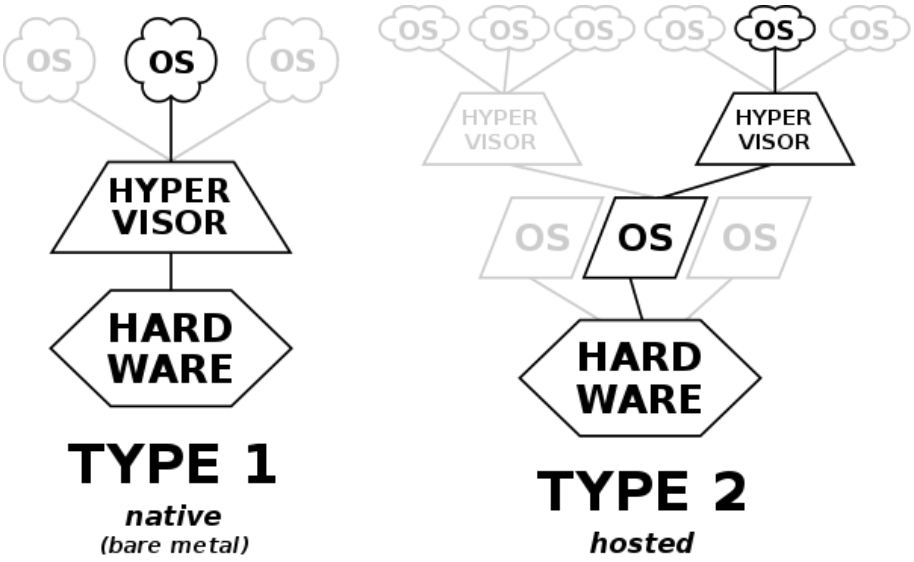


Figure 1.6

Starting with VM's, there are a large number of possibilities for hypervisors, we will focus on Hyper V, Linux KVM and VirtualBox. While Hyper V and Linux KVM are native/bare metal hypervisors, VirtualBox is a hosted hypervisor. As seen in **Figure 1.6**, hosted hypervisors incur a larger overhead, but they do have some benefits.

5.1.2 Hosted Hypervisors

Although hosted hypervisors are not quite in the running, they should be mentioned for completeness. Hosted hypervisors provide a good service for many use cases but not for ours. They mainly work well for a new user wanting to test out a new OS or one that needs to do testing on a different OS. When you only have a few running at a time at most, the overhead that comes with a hosted supervisor is minimal. When you scale up to having hundreds of VM's running intensive firmware testing, the few extra seconds used to start up a VM and milliseconds for a few extra operations, adds up fast.

5.1.3 Native Hypervisors

5.1.3.1 Hyper V

Recommended to us by our client, Hyper V is a native hypervisor created by Microsoft. Originally an optional part of Windows server 2008, Hyper V has a freeware server edition that has less overall functionality than Windows server and uses a simple CLI. This in the end might actually benefit us because we need a reasonably lightweight hypervisor, having many bulky Windows server utilities taken out could increase performance. Hyper V's architecture includes a hypervisor layer, root partition and many guest partitions that can be linked via a VMBus. These partitions, normally unenlightened, can be enlightened to provide them more access to the underlying hypervisor architecture. Hyper V also has a large list of supported OS's that will allow validation to be thorough. The maximums for each VM resource is also more than sufficient.

5.1.3.2 Linux KVM

Also recommended to us by our client is Linux KVM. Linux KVM was developed by the Linux Kernel community in 2007, and is a built-in module that allows for virtualization. It is used in a large variety of ways, one being cloud computing, where a provider can easily partition any number of OS's and load programs onto them. The lightweight nature of Linux with some features like CPU/GPU passthrough make it a very attractive option for many projects, including our own.

When it comes to virtual machines, Linux KVM wins out. Linux KVM actually supports more types of Windows operating systems than Microsoft's Hyper V. Along with more support for more operating systems, Linux KVM gives a more bare metal speed to the VM's it creates, compared to the overhead that comes with Windows server operating systems. Hardware passthrough allows VM's to more directly access the hardware they need without having to be rerouted by the host.

5.1.4 Containers

Containers provide a great service in the area they are needed. In many cases, when most people just went with VM's, containers cut the operational cost by a large margin allowing more work to be done, faster. For our use case, containers, although useful, could cause problems at the low level for firmware validation or for the architecture of our web app.

5.1.4.1 Docker

Created in 2013 by Solomon Hykes, Docker is the main application for containerization. It has a large range of support across many operating systems, which makes it one of the most versatile container solutions. It has become a staple for software development companies and application sharing. Many scientific programs have moved to docker so they can share the programs without having to develop versions for each OS.

Most other ways of containerizing a problem have limited operating system support and are limitedly maintained. Some do have a couple extra features that Docker does not but most of them only support one OS.

5.2 Analysis

Initially, Hyper V is already at a bit of a disadvantage because it is a part of Windows. Because Windows is known to have more overhead than other operating systems, Hyper V has more to make up for in speed. Linux KVM is one of the fastest when it comes to hypervisors but regardless of that, just because of the architecture of hypervisors, it will never beat out containers. Container systems may have the edge in speed but hypervisors still have many features that would complicate containers if implemented on them.
The number of operating systems that Hyper V supports, as mentioned before, is less than Linux KVM, even for the Windows family. Since cloud computing has taken off, the open source nature of Linux has allowed its virtualization to be well supported and rich in features.
Each option discussed here also can easily scale. The hypervisors allow large amounts of resources to be allocated to each VM(as long as the server has them) and Docker has Orchestration systems that allow you to create a swarm of containers to tackle a task.
Both hypervisors have good ease of use, well as much ease as you can get from virtualization. Docker has less specifically for our project. Because firmware validation has to be tested across operating systems, there are two situations: we either have to install each operating system on the container or we have separate systems running on different operating systems so the containers can test the right systems. This can cause some unneeded complexity compared to just using VM’s.

5.3 Chosen Approach

Figure 1.7

	Scalability	Supported OS's	Overall speed	Ease of use	Total
Hyper V	5	4	3	5	17
Linux KVM	5	5	4	5	19
Docker	5	5	5	4	19

Scale from 0-5, worst to best.

As discussed in analysis, Linux KVM is the better of the hypervisor solutions but because of the nature of hypervisors, Docker puts up a good fight. I have found that Linux KVM, along with many great features, provides better adaptability that allows us to work around problems that may arise. It provides this while still having solid overall performance in every characteristic we look for.

5.4 Proving Feasibility

Linux KVM is a native supervisor, so it allows VM's to run very close to the hardware. A Linux hypervisor also brings the lightweight side of Linux so we spend more processing time doing the work we need and not doing superfluous background work. Linux KVM also allows CPU/GPU passthrough to be set up, allowing the guests to pass the host, giving much faster overall speeds. As we move, to prove this we can run speed test demos along with making a compiled list of all the benefits Linux KVM gives us. We may also pinpoint the specific problems that Docker could give us with SSD validation.

Front-End Technologies

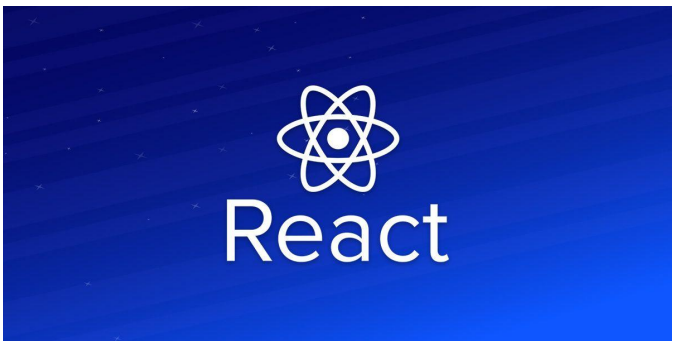
In regard to front end design and formatting, there are plenty of obstacles to consider before dedicating some library or framework to the heart of your web application. The wide variety of options available make it difficult to decide which route to take when creating a responsive and adaptable UI. Primarily, the issue we are challenged with is cross platform capabilities. Ensuring that browser support across the board is capable of producing accurate and functioning components. Eliminating dependencies for front-end production is a difficult task to accomplish, but delegating certain technologies for specific jobs can help eliminate some of this confusion.

When delegating these tasks, our goal is to pinpoint what is important for the front end. For VICE, we want a clean and organized UI with cross platform support while upholding design and fluidity. Questions that we encounter while thinking about our long-term goals include, “What technologies meet our clients needs?”, “How should we structure our UI and how will it be built?”, “What does a sustainable framework for front-end look like?”. Ultimately, how will we combine design and backend components to provide a functional UI for Western Digital.

Numerous frameworks come to mind when adhering to our goal of UI design and profile security. As prefaced in our initial project documentation, our client Western Digital suggested using TypeScript for the UI development. An ideal solution incorporates ReactJS, Bootstrap, and CSS Styling as they provide cleaner UI production.

6.1 ReactJS

Opposed to TypeScript, ReactJS is the optimal route for designing the VICE UI. Consistency takes precedence when creating a practical interface for a wide variety of users, and the ReactJS JavaScript library provides reusable UI components, aiding in composing a simpler model with enhanced performance. React can also render on a server using Node, eliminating a common obstacle. Additionally, React uses VDOM, a JavaScript object. This will help improve performance of our web-application. In contrast, TypeScript only uses DOM. Virtual DOM is essentially a lightweight copy of DOM, as it has all the same properties.



6.2 Bootstrap 5 & CSS Styling

Bootstrap is ideal for our front end design process, primarily due to its formatting. The use of a grid system to structure elements on the page will make for an organized UI. Additionally, this will aid in the cross platform compatibility addressed earlier, as Bootstrap is diverse and focuses on responsiveness. Whether a user opts for a tablet viewport or a desktop, Bootstrap will adapt to these device changes, making the web-application viewable regardless of the device. Using this framework in conjunction with ReactJS will provide concrete structure for presentation, keeping our reusable elements fluid.

Without a doubt standard CSS styling will also be applied in this area, as it is the most common and widely used language for styling. Although, we want to limit the amount of CSS used because large scale CSS files tend to become cluttered. WIth countless class and ID names, it gets difficult to track which element controls what. As project sizes increase, readability and maintenance decrease substantially.

6.3 Angular

Regarding frameworks vs libraries, Angular would be the framework of choice instead of React. Angular is a JavaScript framework built using TypeScript, while React is a library using JSX (a JavaScript extension). Here lies the most difficult decision on which is better.

Angular is great for creating active and interactive web apps, while React has strength when the app has frequently variable data.

6.4 VueJS

Lastly, VueJS is another common tool when creating web interfaces and one-page applications. It extends to both desktop and mobile environments, while

maintaining lightweight. It uses two-way data binding similarly to Angular, also adopting a DOM representation. While there is no apparent problem with using Vue, it has not been around or is frequently used in comparison to its competitors. The community is smaller, resulting in lack of support for instant issue fixes. React or Angular support outweighs Vue's drastically. The learning curve is also a larger jump. Given that it was developed in China, most discussion forums, plugin descriptions and documentation is Chinese. This would lead our team to get lost in translation when trying to develop our product.

6.5 Analysis

To conduct a further analysis on whether NodeJS, React, Bootstrap and CSS were in fact the correct technology choices, simple programs and broke each item down subjectively.

React, Bootstrap, and CSS are quite common and very easy to use. To demonstrate the functionality of each, we incorporated them into our team website. This allows us to easily visualize each element and how they react when the viewport is changed. Additionally, through extensive research, React is highly ranked in regard to UI design. Reusability of HTML elements allow for structure for creating a clean cut interface.

6.6 Chosen Approach

In summary, each front end technology has its own pros and cons, each suited for different goals the developer is reaching to achieve. Overall, ReactJS, Bootstrap 5, and standard CSS styling outweighed the characteristics of its competitors. **Figure 1.8** below demonstrates our decision making process based on these factors.

Figure 1.8

Technology	UI Design	Cross - Platform	Organization & Reusability	Total
ReactJS	5	5	5	15

Angular	3	5	5	13
VueJS	2	2	2	6
CSS Styling	5	5	4	14
Bootstrap 5	5	5	5	15

Scale from 0-5, worst to best.

6.7 Proving Feasibility

With the numerous options easily capable of accomplishing our design goals, ReactJS, Bootstrap, and CSS provide us with the most straightforward approach. ReactJS provides an easy to use UI design platform, thus being effective in achieving a user friendly experience. It branches off basic JavaScript implementing reusable components through JSX.

Additionally, Bootstrap will organize our product through its effective grid system. This addition will aid in overcoming cross platform functionality resolves a common roadblock for front end developers. The structure of our UI will be basic but informative for firmware engineers, allowing all necessary components to be front and center. Pairing this with traditional CSS styling techniques, we strive to make a modern and functional UI for WD.

Reporting Systems (Jira, BitBucket, Confluence)

There are many steps to completing a project, whether it comes down to arranging meeting times, having a to-do list, or even sharing project or data reports. All of these steps require specific mediums to assist in speeding up the process. Some of these mediums would be Jira/Trello, Bitbucket/Github, and Confluence/SharePoint. We are going to be comparing each of these against each other, and deciding which tools would work best for our application.

7.1 Jira vs Trello

What is Jira? Jira is a project and issue tracking and management tool, Created and published by Atlassian in 2002, written in the Java programming language. Jira focuses on the aspect of task management. Jira has the ability to aid with software development, Agile project management, bug tracking, and scrum management. Jira has many different types of 'boards' to use, all categorized by different types of software development processes.

Trello on the other hand, mainly follows one board style. Columns in order from left to right, with each having a list of tasks to accomplish known as 'cards'. Of course, the user can personalize how the board and cards look as well as what the background is, but overall Jira has more customization options for technical aspects. Both management systems offer free plans and membership plans. **Figure 1.9** below demonstrates our decision making process based not only on these factors, but as requested by our client.

Figure 1.9

Reporting System	Integration / Synergy	Ease of Use	Flexibility	Total
Jira	4	5	4	13
Trello	3	3	4	10

Scale from 0-5, worst to best.

7.2 BitBucket vs GitHub

What are Bitbucket and GitHub? These two tools are Git based version control and collaboration platforms. Git is a distributed version control system that these web-based applications use to share code either within a group or with the public. Owners and members of their repositories are able to collaborate on the code, and have one central base to work off of.

BitBucket is a source code repository hosting service, mainly used for public repositories. BitBucket is owned by Atlassian, was coded in the Python Programming language, and published and released in 2008. This web-based application is very modular and supports a variety of different import codes including Git, Mercurial, CodePlex, Google Code, SourceForge, and SVN. BitBucket is also flexible and can use Microsoft Azure, scriptrunner, amazon web services. However, there are a few downsides when using BitBucket. BitBucket has fewer plugin options than GitHub, and only allows up to five free users accounts on a repository. BitBucket has free and paid memberships while GitHub is relatively cheap.

GitHub's biggest selling features are that it's cheap and it is open source. Many people use Github even though it might not have as many advantages over BitBucket because of BitBuckets price wall. GitHub allows users more options and

customizability before having to spend any money. GitHub offers users an unlimited amount of free public and private repositories. A team membership with GitHub is only 4 dollars a month per user, while a standard membership for BitBucket is around 15 dollars a month for 5 users, or 3 dollars a month per user.

Between the two Github is better for everyday people/software developers who work with small projects, because of GitHub's free membership options. While BitBucket has better pricing and flexibility with paid memberships. Both Jira and Bitbucket are owned and managed by Atlassian, which makes the decision to use both for their modularity and simple cointegration. **Figure 2.0** below demonstrates our decision making process based not only on these factors, but as requested by our client.

Figure 2.0

Reporting System	Integration / Synergy	Ease of Use	Flexibility	Total
BitBucket	5	4	3	12
GitHub	4	3	4	11

Scale from 0-5, worst to best

7.3 Confluence vs SharePoint

These next reporting systems are specifically useful for document sharing between corporations and teams. These two reporting systems, Confluence and SharePoint, are categorized as collaborative software. In more specific detail, these two pieces of software are fairly flexible and straightforward.

Confluence is a web-based corporate wiki and collaborative platform. Confluence was written in the Java programming language and first published in 2004 by Atlassian. Confluence is useful for sharing information and documentation between a team, assigning tasks to specific members, keeping track of deadlines, and so much more. Confluence works based on a wiki hierarchy.

What is SharePoint? SharePoint is a web-based collaborative platform that works alongside Microsoft Office. Sharepoint is another wiki-like service that is mainly used to share documents and collaborate. Our client had requested we mainly work with Confluence solely based on integration with the rest of our system, as shown in **Figure 2.1**.

Figure 2.1

Reporting System	Integration / Synergy	Ease of Use	Flexibility	Total
Confluence	4	3	4	11
SharePoint	2	3	3	8

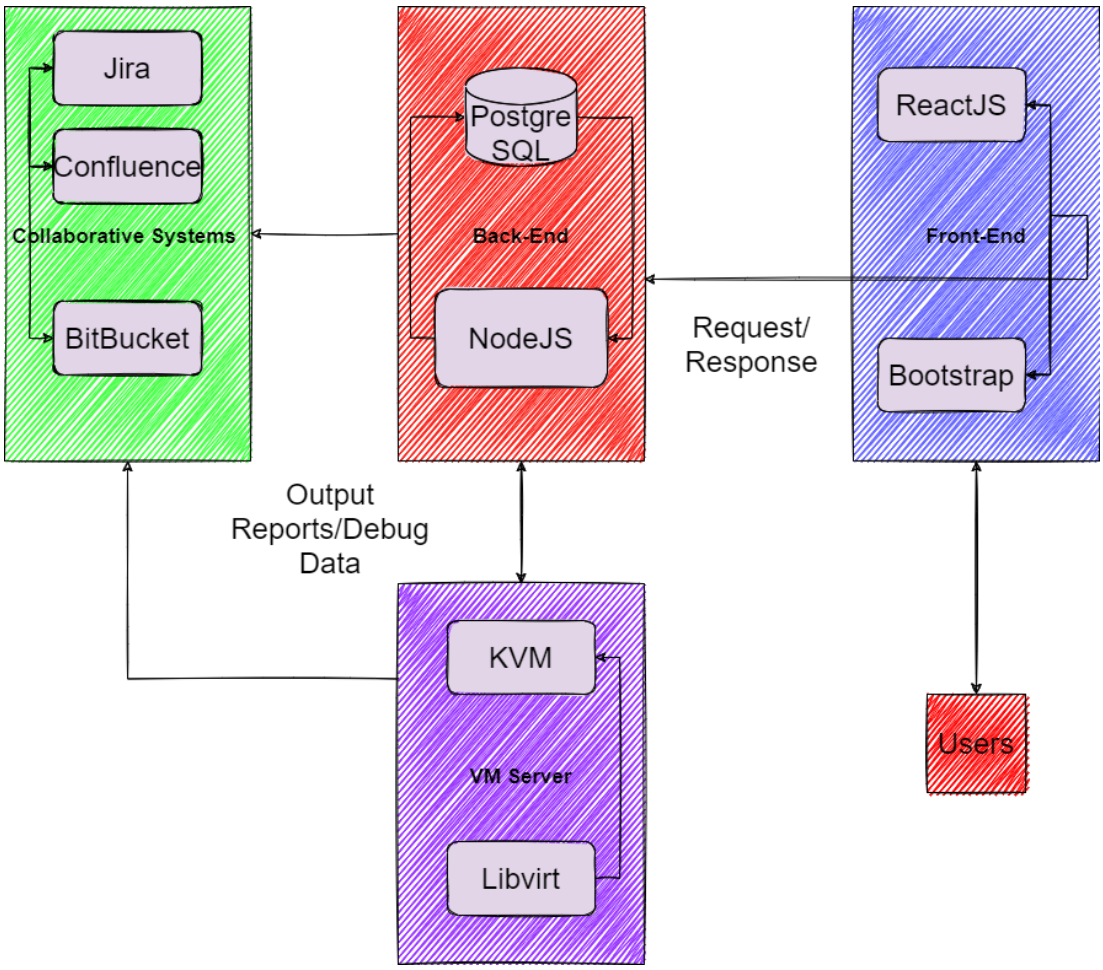
Scale from 0-5, worst to best

The reporting systems we will be using are products all developed and published by Atlassian. These products, BitBucket, Jira, and Confluence, were all specifically created to integrate easily with one another. Our client has already implemented the use of these services with their current systems and have requested that we do the same for this project. Our team does have minimal experience with these pieces of software, within the near future we will be getting the experience we need to implement our solution.

Technology Integration

Connecting this many technologies needs to be done carefully, especially with firmware validation and virtual machines.
Our envisioned architecture is in the figure below. A ReactJS and Bootstrap front end will get data about VM's status and assigned tasks from a backend server and from the previously mentioned collaborative systems WD already uses. NodeJS will take care of most of this information transfer in the back end and format in such a way that React JS will know what to do. This will allow techs to see what they need to work on, what systems are being worked on and so forth, a one stop shop for what they need to work on. The web app will also allow linking into a virtual machine environment for a FWEngineer to work on a bug in. In a similar area, virtual machines that are running tests will send data to our back end so that it can be formatted and pushed to the collaborative systems. The back end will also pull data from the collaborative systems to display what bugs need to be fixed and what tests need to be created.

Figure 2.2



Conclusion

Streamlining any process will always require creative solutions and validating software is a tedious process that may need to be repeated many times to get a passable result. When communication problems are introduced the process can grind to a halt. Creating an environment that gives workers all the information and tools they need to work efficiently will have large impacts on performance.

Implementing a system such as this is not trivial. We have to account for issues such as:

- Storing and Maintaining a database of machine information
- Schedule use of machines and future test plan operations
- Hosting said database and allowing operations to be performed remotely to allow for testing, installing firmware, scheduling machines, etc.
- Outputting testing results to multiple libraries and in multiple views, depending on the user's credentials

We can begin tackling the problem by implementing a front-end User Interface that interacts with our database. The UI interacting through Angular with our hosted database will be the linchpin of our web application. By utilizing the cloud hosting technology Azure, we can maintain an SQL database that allows our system to have a solid back-end. The end result will allow for our application to be connected to by any machine needing to work on firmware validation in the company. All of these pieces will combine together to allow us to create a robust interactive scheduling, testing, and debugging system. The system could be made to accept any firmware-test pairs, run them and send out debug information to the parties that need it.

This solution could also be extended to have a broader impact on all firmware testing, or additionally the Validation as a Service model in general. Our solution could provide a schematic for other validation services in the future. If a GPU company had the same problem, with a modular solution, all they would need to do is link a database of the necessary information to increase their performance by massive margins.