Postmortem

The Hills and Valleys of Bikerr

Team 14:

Sacha Elkaim 29779698

James El Tayar 40097755

Ashraf Khalil 40066289

Shashank Patel 40094236

Adam Richard 27053929

Derek Ruiz-Cigana 40096268

Michael Takenaka 40095917

CONTENTS

1. INTRODUCTION

Over the course of the past couple months our team has laboured to develop and create an Enterprise Resource Planning system or (ERP) for a theoretical bicycle manufacturer we have dubbed *Bikerr*. For many of us this was our very first time in our software careers working on a project of such a scale and magnitude. We would be tasked with not only creating a fully functional website with a front-end, back-end and connected database, but learning the related coding frameworks, languages, and systems as well. While over the course of our classes we had garnered some experience in each of the facets, the idea of combining all these to create a full stack environment certainly had a lot of use worried.

Not only did the scope of the project in terms of coding knowledge pose a challenge, but what we were coding, a complete ERP system, also added a whole realm of depth and scale to the project. ERP systems are a complete collection of modular functions that encapsulate every facet of a modern business. Production, accounting, transportation, sales, procurement and many more modules are all integrated on one platform, constantly communicating, and sharing information with one another concurrently.

Immediately, from the very first meeting, our team developed a plan for what would become the skeleton of our project in a quick efficient manner that would become a common occurrence in our teams work ethic. The stack that was chosen was MERN (MongoDB, Express JS, NodeJS and React) with Docker being used to run the containers. Jest, Selenium and Travis CI were used for testing purposes and a plethora of node packages were installed for the various functions and utilities that were needed. With documentation being updated often and scrums occurring twice a week, tasks were distributed early and fairly with everyone being on the same page. The use of Discord to communicate meant that changes, updates, announcements, and coding in teams were a normal occurrence helping our productivity as well as our team chemistry.

As our product required the various modules and functions of our system to all communicate with each other and often relied on each other the aforementioned team chemistry and communication that we had built became indispensable. Often coding teams working on separate modules would be seen working together allowing for the various sections of our website, on the front and back end, to meld together into the integrated system we had been tasked to create. The team ethic we had fostered also helped create an environment where members felt like they all had collective ownership of the project. Meaning more often than not even if a member had finished their allocated work, they would help others finish their sections as they felt a responsibility and connection to the website as a whole rather than finishing their work and leaving the rest to their fate.

All the above details melded together to give us the final deliverable that we got to present at the end of sprint 4. A fully completed ERP system that every team member could be proud to have participated in. Everything that we had set out to accomplish at the beginning of the course had been done and what we were able to produce matched and, in many regards, completely exceeded our expectations. Despite the hiccups, setbacks and plethora of bugs that popped up, our team's response was always to be better. Our constructive and positive environment and ethic allowed us to overcome all of these challenges. This confidence will be carried on well after this course and can conclusively be demonstrated with our members' decisions to maintain this team for our forthcoming Capstone project.

2. WHAT WENT WRONG

This section identifies and analyzes the main aspect of the project that did not work well during the development.

2.1. LACK OF INVOLVEMENT FROM ONE TEAM MEMBER

One of the clearest issues we encountered during the development of this application was the lack of contribution from a certain member of our team. The member had not only not completed any of the tasks they were delegated in any of the sprints but would not even do any portion of them at all. This was especially

detrimental in the earlier stages of the project's development, being that we could not accommodate for their lack of contribution being that we had not expected it. There was a single instance where the team member actually attempted to complete a task they were given, presenting to us their work on the day of the deadline for that sprint. The page they constructed was not only non-functional, but all of the page's contents were completely incorrect, and unusable, making his work not submittable. This led another member of the team to take over that task and develop the entire page from scratch hours before the deadline.

Due to the events mentioned above, we were forced to acknowledge the fact that this team member was very unlikely to deliver on what they were delegated, and thus, stopped delegating them work. At this point forward, the team member's lack of contribution was no longer a significant detriment because we no longer expected anything of them. The only remaining issue was in regard to the fact that now the remaining 7 of us were expected to complete a larger portion of the remaining work. Fortunately, the 7 of us were all eager to contribute whatever we could, and we were not seriously overwhelmed by the increased burden.

2.2. SETTING UP JEST

Jest is a JavaScript testing framework that was used in our application to perform unit tests on our back end NodeJS framework. We ran into a substantial issue during development when running tests where one of our tests would complete but all the other tests would automatically fail. This issue was hard to debug as all our tests individually were working correctly but did not seem to be able to all pass when running all the tests using npm run test. The reason the issue happened was that each file was calling server.js and subsequently all trying to connect to the app.listen port but only one instance could connect at a time, causing the rest to fail. This issue was fixed by making a condition that when running tests, no entry to app.listen was allowed and a connection to a virtual database was made instead, which allowed all of our tests to pass successfully.

In general, our integration of Jest into our application was a success. A few hiccups in the first sprint happened but was expected as none of us had used this testing framework before this project. One improvement that could have been done was to not underestimate the amount of time that would be needed to research and get these tests running correctly.

2.3. SETTING UP DOCKERS

Docker uses OS-level virtualization to deliver software in packages called containers. We were required to publish our containers on Docker-Hub to share our containers with the TA and in doing so ran into two issues.

The first issue was with the communication between containers. When creating multiple containers using docker-compose for the front-end and back-end, both containers worked independently but did not communicate with each other. The application essentially broke as no API calls were being fulfilled. This issue was resolved by making our front-end proxy's address name match the back-end containers name when creating these containers.

The second issue was that we could not run the docker images in one command line as required by our TA. Since we had two separate docker images, we had one command to run the front-end and another to run the back end. We could not solve this issue for the first sprint, but eventually found a solution before sprint 2 by binding both commands with an && statement.

In general, the docker integration into our application was a success, but had we researched more on how to combine the front-end and back-end into a single container instead, we would have avoided the communication error and command line problem entirely.

2.4. TRAVIS-CI

Travis CI is a continuous integration service used to build and test software applications that are hosted on GitHub or Bitbucket. We used Travis CI as a way to ensure that every commit, pull requests and merge to master did not break the application. We had a couple of issues with Travis CI throughout development. The first issue was with the Jest version and the second was related to Selenium.

Firstly, when we installed Jest as a dependency to perform tests, we did not take into account the jest version embedded when we first created the app. The other issue was we did not specify which folder it must run the tests on. This led to Travis-CI failing the build, thus leading to failed tests. Thankfully, it did not have a severe impact on the application as the tests were still passing locally on our machines. We solved this issue by reverting the Jest version to 26.6.0 and adding a regex so that Travis-CI knows which files to test.

Secondly, while performing UI testing with Selenium, we came up with another error. Travis-CI was unable to open chrome and perform the tests. This occurred because we did not have a headless browser. This impacted our progress significantly as it stopped a couple of the teammates from working on their feature as they helped solve the issue. We solved this issue by making Travis-CI use a headless browser and install the chromedriver before it performed any tests.

In general, setting up and using Travis-CI was a success. In the future, we could anticipate the upcoming features and make sure that Travis-CI has all the required commands to handle the implementation ahead of us.

2.5. TIME MANAGEMENT

The final issue that we faced during the development of Bikerr was time management. The development was split up into 4 sprints and for each sprint we would have meetings at the start and end to coordinate the different sections. Even though we had recurring meetings often, we did not coordinate between each section as efficiently as possible. For example, if two sections had high cohesion, the tasks would be done separately then joined at the end and a lot of refactoring would be needed. This led to having last minute meetings very close to the deadline to finish and submit everything. Waiting until the last minute was very bad for us, because for the last sprint we rushed to commit and merge all our work together however we did not realize that we were overriding some of the work that others committed minutes before.

The catastrophe had the project unable to compile and some code not working as intended. It took another 1 hour just to find all the issues that arose due to the conflicts and refactor them again. To combat this issue in the future, we should set artificial deadlines at least one day earlier than the official final date so we can work out any issues and test the application for every feature that is required.

3. WHAT WENT RIGHT

This section identifies and analyzes the main aspect of the project that did work well during the development.

3.1. STRONG TEAM COMMUNICATION

The team agreed early on that we would meet on a weekly basis. Each meeting, we had a list of topics we wished to discuss, and we went through them. Anyone present at the meeting would then be informed for all tasks that needed to be completed for the following week. Most members, if not all members, were present at each meeting, which meant the team as a whole had a good idea of where we were at in the project and what still needed to be completed. Notes were taken during each meeting and uploaded to the GitHub's wiki page for review should anyone need a reminder of what we discussed, or if anyone needed to catch up after missing. Thus, team communication had a strong base to function well. Further to these weekly meetings, we also had additional meetings to further communicate more minute details of specific tasks. These often took the form of programming meetings on Thursdays. At the end of each meeting, we would organize when the next meeting

would be. This information was added to the meeting notes on the GitHub wiki, as well as on the wiki's main page for that sprint, so team members could easily find out when we would next meet.

Further to this, no one on the team was afraid to discuss shortcomings or difficulties they encountered. This made it easy for the rest of the team to help, and anyone who could lend a hand would. We used our discord server for our general discussions and served as a place where the team could meet. This discord server provided both text and voice channels which enabled communication over a distance. We knew that working remotely would come as a challenge, and that team cohesion could be low as a result. We set up the server to combat that, and it worked, allowing us to effectively communicate all semester.

3.2. QUICK ADAPTATION OF STACK

During the first meeting of the semester, we brainstormed on different stacks we could possibly use for the development and after much research on different stacks, we decided to use a MERN stack. There are a couple of reasons as to why we used this stack. Firstly, two of the team members, Shashank and Sacha, were already familiar with it and thus they could guide the team if any member needed help. Secondly, it is a very popular stack currently and a lot of good resources are available to us online. Lastly, everyone was eager to learn a new programming language as it is a huge advantage professionally.

During the first sprint, everyone focused on learning the language while the two team members already familiar with the language set up the project. This allowed everyone to get up to speed with the language and install all the necessary software needed to properly code. JavaScript is an easy language to learn if you have good knowledge of how programming works. The fact that we used bootstrap elements, it made our application mobile friendly without putting in much effort. MERN stack was the right way to go, and we will be using it for our Capstone project.

3.3. GOOD TASK DISTRIBUTION

All members of the team were motivated to contribute whatever they could to the project. Tasks were distributed fairly, and as equally as possible. At no point did a team member complain about what they were assigned in regard to its size or its contents. This definitely aided in terms of the efficiency of the development of this project, but also in terms of the team's morale. All team members respected one another, in their individual ability to contribute a fair amount of work that met a satisfactory standard of quality in a timely manner.

Another equally important aspect of task distribution that positively impacted our development was the flexibility in which tasks could be delegated. Due to the fact that members of the team trusted one another to complete tasks satisfactorily, team members felt comfortable asking for help when they were overwhelmed or felt that they were assigned too much work. This made it so that even if we miscalculated the difficulty of certain tasks, making it so that certain members were burdened with unfair amounts of work, they were able to let the other members know and ask for help, allowing for a more equitable redistribution of work that satisfied everybody.

Another aspect in which the flexibility of the delegation of tasks was hugely conducive to our eventual success was the flexibility in which we were able to assign tasks to different sprints. As is mentioned above, if a given team member felt overwhelmed with the amount of work that they received, team members were always more than happy to redistribute the delegation of tasks for that given sprint. Unfortunately, it was sometimes impossible to redistribute the work in a way that allowed for its feasible completion within the deadline of that particular sprint, and thus the rescheduling of the deadlines for certain features was necessary. By virtue of our team's impeccable communication, rescheduling and redistributing work was always a seamless process.

3.4. ESLINT AND PRETTIER

ESLint is a tool that is used to identify problematic patterns in JavaScript based code. It checks for syntax and style errors throughout your project. On the other hand, Prettier is a code formatter to ensure that the code is consistent. Consistency encompasses indentation, spacing, single quotes, etc. We decided to use them together in order to ensure that everyone follows the rules, to save time when reviewing code, they catch the syntax errors and mainly it is a one-time cost.

We decided to opt for these two tools because we thought that it would save us a lot of time when reviewing code and that we would not have to worry about formatting since it would do it for us on save. In addition, Prettier and ESLint allows you to set your own rules, meaning if a team is more comfortable using double quotes over single quotes, then they can simply add that as a rule. They are amazing tools that make a programmer's life a lot easier and we absolutely love these tools, and we will be using it in our future projects.

3.5. WORKING PROTOTYPE

After working on the project for about 3 months, the team was able to produce a working prototype that we are proud to put on our CVs. The prototype that consists of segregated sections, ranging from accounting to manufacturing, accumulated into one ERP system is as put together as the team could make it. We did not have parts that were completely broken or could break other sections of the system. The way that we achieved this goal of ours was by setting deadlines and certifying that we did not suggest features that would not be possible to implement in the allotted time. Because we were able to complete a working prototype, we have learned a great deal from seeing a project through to the end as well as have a significant reference point to refer to when we need to complete a large project again.

We also applied the concept of a working prototype for the deliverables in every sprint. At the end of each sprint, we designed a prototype that followed the same rules as the final product; there should be no broken code and complete all functionalities required. During production, everyone guaranteed that their section had complete coverage of what they were tasked to do. In addition, we tested that all interactions with the interface were complete and did not cause substantial issues that would interrupt the application's flow. One aspect that we could have improved on however, was to remove unused code from parts of the production build. Once the build was complete, we tried our best to erase as much of the dead code that we could find, but some are scattered in hard-to-find corners. These snippets of code do not affect our final prototype of the application, however it does detract from the more professional portions that we are very proud of. With the use of ESLint and Prettier, we can easily mitigate the problem and take the proper steps to clean the build.

3.6. GOOD TEAM MANAGEMENT

Team management here refers not only to the people acting as project and software managers, but to the team as a whole and how it was organized. The discord server, as previously discussed, was a huge benefit here. It allowed us to remain organized through the entire process of developing the ERP software, and without it we would not have seen the same amount of success as we have. The discord server was split into multiple text channels, each with its own purpose so that our communication could be organized, and important information was not lost.

Further to this, we met regularly with the TA. This was beneficial in that it helped us understand our shortcomings each sprint and allowed us to meet to discuss them and brainstorm ideas to improve. This led to a massive improvement between sprint one and sprint 2 and carried us through to the end of the project.

We also used our release plan to great effect. Each task was given story points so we could understand the weight of each task, which helped us distribute them more fairly. Each sprint had a Gantt chart beside it so we could easily visualize how the sprint was supposed to play out. Most importantly, this served as the main place we could go to make sure each member knew what they were doing and knew how to get it done. This made it so that team management required minimal effort, as everyone could effectively manage themselves. All other management discussions were held during meetings, but often did not require more discussion than dividing

the tasks and checking in to see where everyone was and how everyone was doing. The overall smoothness of this process is why we define our team management as being one of our strong points.

4. CONCLUSION

In conclusion, we learned a lot during this project that will help carry us through future affairs. The knowledge we gained during this project is due to poor time management and problems getting certain software applications to work like Jest, Docker, and Travis CI. Our takeaway we obtained from this was to set deadlines a day before the actual deadline in order to account for unpredictable problems that occur. That way, we will have time to fix the software application problems instead of rushing to finish last-minute. Each sprint brought new challenges that we faced head-on and worked very hard to complete. Although a major part of our problems was caused by a member that lacked involvement during the project, there are still things we need to work on. However, at the same time, we did show strong team communication, quick adaptation, and a good distribution of tasks, among other things, that we will maintain hereafter. For our capstone project, we will continue to improve upon the things that went wrong, while maintaining the things we did good in order to maximize the value of our product.