Team 1
SOEN 390

Post Mortem Report

Downhill ERP System
Version 1

Submitted on
April 20th, 2021

The End of the Hill: Post mortem

Introduction

Our task was to design an ERP that would allow a hypothetical bike company to track its inventory, materials, finances and allow modifications, additions and deletions of this data. At first, our team wasn't quite sure how to proceed as building an ERP was not our first choice. However, after looking into it a bit more, we realized that It could be done. Using the concepts we have learned throughout our degree such as database creation, back-end development, API calls and front-end design, we created an application that would respond to the project's requirements.

We are quite satisfied with the final result. Our application incorporates all of the original requirements that were outlined to us at the beginning of the semester. We are able to take raw material, make a bike from it, track how many bikes we are selling, as well as the cost that is incurred from the purchase of the materials and finally we can see our total net profit/loss on a sleek graph.

The framework we used for this project is ASP.NET alongside C# as our main programming language. The database management system we used was PostgreSQL. For views, we used HTML with Razor syntax alongside Javascript and Bootstrap for the styling.

What went wrong

1 - Inexperience with the Tech

A major problem at the beginning of the implementations phases of the project was that half of the team had no prior experience or knowledge of C#, ASP.net or web development. This made the completion of features very difficult and slow during the first 2-3 sprints. Some team members really had a hard time completing their tasks on their own since they had to spend countless hours researching basic technicalities. The overall productivity was strongly affected by this issue. Also, the more experienced coders ended up coding more than the inexperienced team members. This happened because it was impossible to pick a coding language that satisfied everyone in a group of 9 people. Our team chose C# because half of the group was very experienced with it.

Our team addressed this problem in two ways. First of all, the members who were familiar with the technologies were always available to help during coding. They were very good at giving instructions and setting the inexperienced coder in the right direction to complete the task more easily. A second solution was to distribute the features to two members, a person with experience and another without so that they can work together to complete the features as smoothly as possible.

One thing that could have been improved on is that sometimes, two inexperienced coders were grouped together for a feature. In this occasional scenario, the productivity was low and it very negatively affected the rest of the team since other team members had to spend a lot of time helping them instead of working on their own tasks. This happened especially during the first 2-3 sprints.

2 - Misunderstanding of the Requirements

As in any team project, the work had to be divided between the team members to ensure that everyone did their part. This division of work led to a couple misunderstandings of what was actually required for each person's part. Members of the team were often not sure about what to do, and if they thought they knew what to do, it often wasn't exactly what was required and needed to be corrected/redone by someone else. In the future, a more concise and explicit version of what is required for someone's feature would help lower the ambiguity so that the feature ends up as intended without having to be reworked or started over from scratch.

This was more of an issue early on in the project as we weren't really sure where we wanted to go with it. As the sprints came and went, the project as a whole became a bit more clear and each part was properly divided with the necessary information in order to complete it.

In the future, in order to avoid these misunderstandings, it is crucial that the requirements for everyone's part/feature be extremely clear. This will prevent any issues down the line. If the requirements for the entire project are mapped out perfectly then every part will be completed with no problem.

3 - Schedule Conflicts

Throughout the first three sprints we were often behind our planned schedule which resulted in pushing some of the planned features/user stories to following sprints. This issue was encountered by the big differences in personal schedules across all team members. In fact, our team was composed of students taking a full 5 course load and team members having a full time employment. This caused some coding issues to be blocking someone else's task for long periods of time, even caused some pull request to take a lot of time before the requested changes were addressed, because someone would make the request during their available time, but this time wasn't necessarily a good time for the owner of the pull request/task which cause us to be behind schedule. We tried accommodating this issue by having meetings early in the sprint. However, this didn't seem to solve the issue since having meetings wasn't the issue, but the issue was mainly caused by the time availabilities to work on the project.

During the last sprint we managed to control this issue by placing strict deadlines to every task to ensure that blocking tasks would be avoided as much as possible. We believe what

could have been done better is to assign tasks according to people's similarities in schedule. We believe this would have required a lot more time on schedule preparation, but would have been very helpful in the long term.

4 - Task Assignment

We always had meetings to decide the features that would be implemented and from those features we would create tasks. Often, it was hard to get every task assigned because we tried working on a volunteering basis, but we didn't always have volunteers for every task. This issue caused us to take longer in the project management phases of the project and also created some small confusion during the development phases on who was assigned to some tasks.

In order, to solve this issue we stopped working on a volunteering system and started working with two team members splitting the tasks evenly across all team members. Splitting the task was a hard job for the two team members since it is hard for someone to take on an "authoritative" role without necessarily having more experience in the field. Therefore, we think that something that could have been done better is to split the team into subgroups such as frontend, backend developers throughout the duration of the project so the task could easily be assigned according to the team members role in the team.

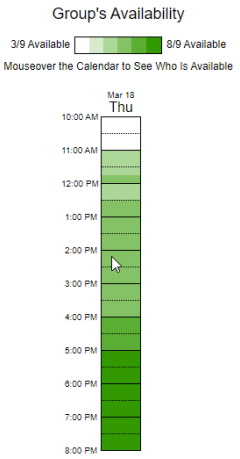
5 - The use of firebase for user authentication

Firebase allowed us to track our user's information in a safe and secure manner. However, a downfall of our use of firebase came when we wanted to incorporate the popular firebase libraries. These libraries would allow us to perform more complex tasks and improve our user management capabilities. The issue is that for C#, there are firebase libraries available however they aren't at all complete, and the main ones that we would use were not available. This meant that we were not able to do a lot of the things that we wanted to do, which we could've accomplished by using a language other than C#. It was too late to remedy this however as we had already committed to using C# as our main coding language.

What went right

1 - Communication

Over the course of this project our team maintained good communication. This occurred as a result of a team member taking on a leadership position who organized many meetings. An example of how our team scheduled meetings is displayed in the image below.



An issue that kept occurring was finding a suitable time slot that satisfied all of our busy schedules. The impact of our communication led us to always submitting the minimum requirements prior to the deadline. We communicated over one central telecommunications application known as “Discord”. There we had several text channels and voice channels to organize the type of communication that took place. Discord provided us with a handy platform to have our meetings and share important links. In addition, using GitHub provided us with a way to share our code and to visualize the tasks assigned. Using the Zenhub integration for GitHub allowed us to visualize tasks and see the progress of our peers. Our team could improve on being more consistent when it comes to updating our tasks on ZenHub.

2 - Coding practices

Throughout the project our group felt it was important to implement good coding practices when developing our ERP. During one of our first team meetings we established the coding conventions along with the softwares that we intended to use. When a PR was made we ensured that the conventions were followed and even requested changes if needed. The MVC pattern kept our code organized and was easy to follow. In order to implement a more traditional MVC pattern we had refactored the code which moved our business logic to the Model rather than having this logic in our services.

Throughout the different sprints we made improvements in our code and always strived for well written code. This included reducing duplicate lines, respecting our conventions and taking advantage of polymorphism in some cases. In the future, we could have made the conventions more explicit through examples and presenting examples to the group. This would have saved on time instead of asking for changes. Applying a linter for our custom coding convention in our pipeline could have also helped automate this process.

3 - Collaboration

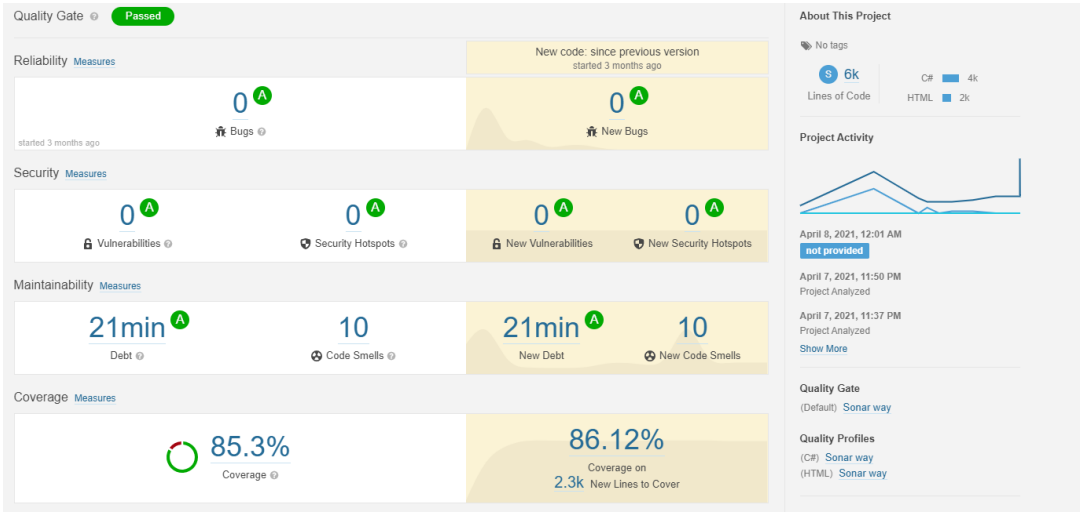
Over the course of the project during each development phase we had very good collaboration when it came to helping people. When a team member required help people were always willing to help others. In fact, one of our main strengths throughout the development phases was our pair programming. Very often, members of the team went on group calls for pair programming, which was really helpful to everyone in the team as it allowed us to learn from other people's strengths and knowledge. This also allowed our code to be more consistent and readable, while also familiarizing people with other parts of the code even though they were not assigned to it.

Our overall collaboration was good. Indeed, we were never in a situation where a team member didn't know how to do something with no one there to help them. This created an environment fun to work and easy to learn in. We don't think there was anything to work on when it comes to collaboration since we feel this was a success and we did everything there was to be done.

4 - Code Quality

Throughout the sprints, we enforced certain rules to ensure that our codebase kept a high quality. We decided from the start that these rules needed to be put in place to avoid having to deal with lots of refactoring and fixes later on. These rules included: having a minimum of 80% code coverage on all pull requests, making sure our code followed all agreed upon conventions and making sure we got rid of all code smells and bugs before merging. This way, our main branch would never have bugs and always contain a working version of the application.

We monitored code quality using the SonarCloud dashboard that would be updated upon every pull request being made, evaluating each one and comparing it with our standards. The dashboard provides us with many different measurement metrics that help us in evaluating our code, but also to identify areas that need improvement. This is what the main dashboard looks like:



5 - Testing

To add onto what was said in the previous point, we also enforced good testing practices to make sure we were able to meet our code coverage goals. These rules meant that we had to write tests for each model and controller in the backend covering all the functions when possible. The SonarCloud Dashboard helped identify uncovered lines by our tests.

Additionally, we implemented manual UI testing to ensure that our Front end was running smoothly and all the features were working as intended. The manual UI tests we wrote were a big help since whoever was testing could always go over the same set of tests and even add some if necessary. This was especially important since we had different people testing different versions of the UI and this gave us a way to stay consistent with our testing. As a whole, everyone on the team did a good job of testing their own parts, this led to a great overall coverage at the end of the project.

Conclusion

Throughout the project, we had the opportunity to learn many skills that will be useful for our careers in the future. This ERP project enhanced our knowledge of being able to analyze user's needs and then design, develop and test a software that conforms to those needs. Many of the tools used in the project were new to most of the team. For example, Zenhub, Firebase, SonarCloud, etc. Knowledge of these tools can be very helpful for future software projects.

On a more technical note, although many of us were familiar with C#, not everyone had an in-depth knowledge of the language. This project allowed us to learn the language more thoroughly and discover the tools that can be used in order to achieve the desired functionality of our project. As we know C# is a very versatile language, it can be used to develop many different types of applications. This means that the knowledge gained during this semester will not go in vain.
On a less technical note, as software engineers, technical skills are not the only skills needed to be useful in a team. Abilities like teamwork, communication, problem solving and attention to details are very important to carry out a software project in a civilized fashion. Most members of the team had an adequate amount of skills in those areas which made the project workflow very smooth. We also learned different techniques to enhance these skills to benefit from the project. These techniques consisted of using tools to see who is doing what (Github and Zenhub), to provide availabilities for team meetings (Schedule tool mentioned above), being able to review a team member's code in order to make sure that the problem addressed was solved properly (Reviews on GitHub) and testing to ensure that every bit of the code is properly implemented.

Finally, this ERP project was very challenging but helpful for everyone involved in it. It allowed us to develop and enhance many technical and non-technical skills that will be very helpful for our future. A lot of this acquired knowledge will be taken with us to future projects in school and later in the field.