An **E**pic **R**e**P**ort

Team 4

Introduction

The team developed an Enterprise Resource Planner (ERP). The software that we created is called Epic Resource Planner. Our ERP is a web application that allows its users to define and track both manufacturing and inventory of raw materials, parts (sub-components), and final products (bicycles), as well as track orders and finances.

An Enterprise Resource Planner (ERP) is a system that allows an organization to track production resources at different stages of the product life cycle. ERP systems are common in many large companies, they increase the resource planning capabilities of the organization, give an overview of the state of the affairs, as well as perform analytics on the vast amounts of data that the enterprise produces. Like we did with our product, ERPs can also be integrated with finance, human resources, sales, etc… Overall ERPs help companies streamline and digitize their business.

Our expectations were that this project would be a learning opportunity for all of us. We expected that the project would emulate the real project development experience. This entails communicating with a stakeholder, producing documentation, planning releases, etc… All in all, we would definitively say that the project met our expectations.

The team did great. Meaningful and constructive feedback was received from the TA at every sprint review, and the professor had only positive comments when he was in the sprint review. It is fair to say that the software turned out well. We would say that we are proud of the final product.

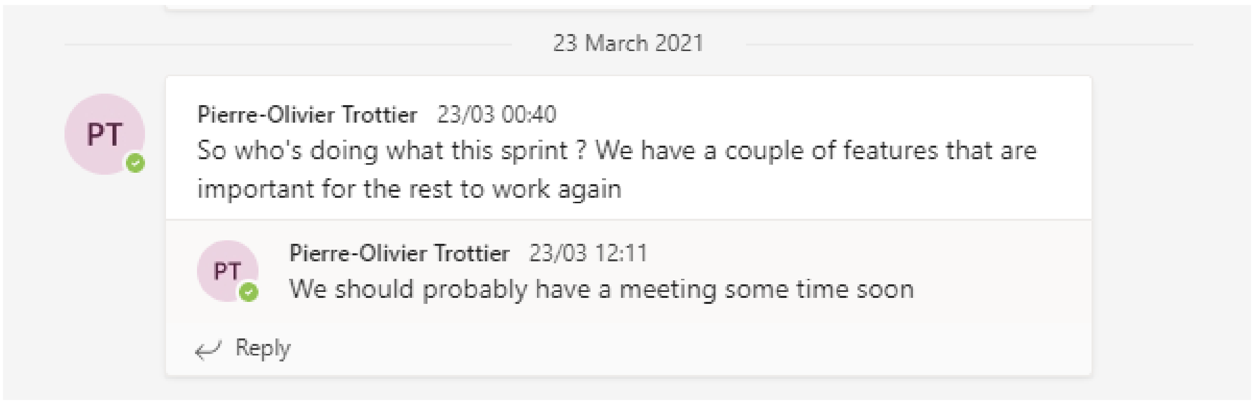
Our ERP meets all the functional requirements. The result is extremely fast and responsive because we used the React.js library. It is also very visually appealing because we used the AntD component library. There were a few bugs that remained by the final sprint, but they are minor. The system is very function-rich and we could easily see our product being used in the industry.

In terms of the tools and technologies used, the project uses React.js for the frontend and NestJS for the backend, which is a framework built on top of Express.js. The data is stored in a MongoDB database stored on the same physical server as the HTTP server. While that organization is not optimal, it helped the team reduce the hosting costs of the application to a minimum. Finally, the project was hosted on a DigitalOcean droplet and deployed using a GitHub Actions pipeline.

What went wrong

1 - Team communication (Moving from Teams)

We initially began Sprint 1 using Microsoft Teams as our main means of communication. This proved to be ineffective as the response times from teammates were often delayed or disregarded. Many team members also complained that the application was slow and that there were collaboration issues while working on shared documents. The impact of using Microsoft teams was frequent small frustrations amongst some team members. To address this problem, we kept our documentation on Teams but started to communicate on Discord. Once the transition was completed, the communication between teammates was very efficient and responses were received in a timely manner.



23 March 2021

Pierre-Olivier Trottier 23/03 00:40
So who's doing what this sprint ? We have a couple of features that are important for the rest to work again

Pierre-Olivier Trottier 23/03 12:11
We should probably have a meeting some time soon

As shown above, nobody saw this post and resulted in us meeting later than needed during the sprint. Moving forward, choosing a means of team communication should be discussed and agreed upon by everyone at the start of a project. Selecting Discord from the start would have avoided this situation.

2 - Time management (coding-wise)

Part of the difficulty of any university project, course, or assignment is to balance it with other courses and life outside of university. As a result of our attempts, some features would be merged briefly before the sprint deadline (by a day or by an hour).

This method of time management led to some risks of the code base breaking (although the pull requests were carefully reviewed), to lower team cohesion, and to a decreased overall understanding of the ERP’s features. We addressed this issue by planning sprints earlier, which allowed us to have a better timeline for every task. If this project were to be done over, we would want to either scope down the project to achieve better time management or we would want to dedicate more time to the project as a full-time endeavor.

3 - Choice of the technology stack

While most of the team thought this project was a great opportunity to learn ReactJS, we quickly found out that the tooling and overall structure and architecture of a vanilla ReactJS project are not fit for an industrial application project.

Looking back on our choice of framework, using a framework that’s more centered towards large industry applications like NextJS would probably have helped us with simple concepts such as Authentication, Authorization as well as Routing, and State-Sharing between components.

The impact of this choice was that the tooling also led to many problems such as UI components not hot-reloading when expected and error messages not being cleared after the bug was fixed until the development server was manually reloaded.

We did not address this problem because it was not worth changing the tech stack halfway into the project.

The team could have done better by creating a table of pros and cons for each technology and decide less spontaneously which one would be chosen.

4 - Infrequent meetings

Our team didn’t adhere to very strict Agile methodology during our sprints, which resulted in infrequent meetings. Regularly, at least a week passed after the start of a new sprint before we started discussing which tasks and features to tackle, and work was only delegated and started around the middle of the sprint. This happened because most members were preoccupied with other deadlines, from other courses. This impacted our time management issues, as individual members of the team who might have been able to start on their tasks were unable since work had not yet been assigned.

A contributing factor was the lack of a well-defined meeting schedule, so the responsibility of setting a meeting was on whichever member of the team took the first step. If we had decided on scheduled meetings at set times early on in the project, such as twice-weekly scrums, it would have taken the responsibility of asking for a meeting off of individual team members. This would result in tasks being assigned and work started and completed earlier in the sprint.

To address this issue, later sprints were assigned from the start and many members worked in pairs. This worked well because the pairs would meet frequently throughout the sprint. If we would have redone the project, we would have worked more in pairs, this definitely improved the lack of team meetings as more people were active in Discord voice chats every evening.

5 - Code ownership

A few *knowledge silos* emerged over time as some team members became very familiar with some sections of the codebase while others remained mostly ignorant. This happened because team members that worked on a certain page would tend to take other issues on the same page. This is negative because extension, debugging, or refactoring is dependent on the participation of this individual. Thankfully, every team member who had ownership over sections of code was always willing to meet and work with others to share their knowledge. To address this problem, as mentioned in the previous section, many issues were given in pairs, this promoted knowledge sharing. To further share the knowledge of the system if we had to redo this project, issues should be assigned in a way so that every developer touches upon all aspects of the project: front-end, back-end, testing, CI, etc.

What went right

1 - Team Cohesion

The team had high Team Cohesion, a metric of success related to the scale factor in the CoCoMo II model. The high cohesion was possible because most of the team members already had previous experience with each other and because memes were often posted on Discord, increasing the relationship between members. The impact was that all team members felt free to talk to the rest of the team if they required assistance or knew that their issues would take longer than they expected. Since team members cared about each other, they would be willing to help even in hard situations. The only thing that could have been done better was to have real-life meetups, to improve team cohesion further. Unfortunately, due to the pandemic, this was not feasible.

2 - Responsibility

During each sprint, each team member displayed a great sense of responsibility. Members consistently delivered their tasks on time. This was due to the great work ethic and cohesion within the team. In cases where tasks depended on others, members assigned to those tasks made sure to promptly complete their work so other members could start their work.

Furthermore, when some members seemed to be overwhelmed by their tasks, other members would more than willingly assist that person. Therefore, not only did members do their part, but they were also open to taking on more than what they were assigned to for the benefit of the team. However, as a collective, the bulk of the team's work was done fairly late into the sprint which could have been improved with better management.

3 - Delegations of Tasks

Before starting work on each sprint, tasks were appropriately assigned to each team member. These decisions were taken as a team to promote fairness. This allowed members to plan ahead to complete their tasks. The delegation of tasks encouraged team members with similar issues to work together to accelerate the coding process.

Members were also assigned to different parts of the codebase to familiarize everyone with different parts of the project. Although this was not always the case, as members were more often than not assigned to parts of the project that they have already worked on, this made working on the tasks more efficient. However, what could have been done better would be to delegate the tasks sooner during the sprint. This would have led to less confusion and more preparedness throughout the sprint.

4 - Highly competent students

Through a 4-year university curriculum with many team projects, it is natural that team members that contribute more to some projects tend to team up with students that also contribute sufficiently while avoiding those who contributed less. Given that this is a large project, we made sure to find and recruit the best-performing students. The impact of this was that every member was well seasoned with programming in general and had a good work ethic. Even the team members that had never worked on any section of the aforementioned stack were able to contribute meaningfully to the project. If we had to redo the project, we would have not changed any member of the team.

5 - Agile Product Design

Not only was the backend solidly designed and implemented for most of the sprints, but the frontend was also sleek and worked on many different platforms. None of the features are faked, and the only mocked aspects are the payments of payables and receivables (done automatically every night). Ultimately, because we revisited the design of the backend multiple times without fear of modifying it, our product became better for it.

In a future project, or, if we had to do things differently, we would want to allow for the product to be even more of an alternative by allowing live, simultaneous use of the ERP. Regardless of the inclusion of this feature, our ability to go back to our designs and edit them led to a better final product design.

Conclusion

During this semester, the team was able to learn so much about the software development life cycle of an ERP software application and potentially about a new web framework. Not only did this project allow us to put into practice many of the Agile methodology concepts that we have seen in class, but it also allowed us to learn about new technologies. Choosing to use technologies that no one was an expert with allowed us to learn and grow together.

This additional challenge helped each team member rely on the others when they needed help. Because of that, the team cohesion was immense. The entire team learned to collaborate on a bigger scale than other school projects.

Lastly, the team learned that making good technological choices before the project is in full development is very important as making a bad choice at the start will be annoying for the remaining duration of the project.