Rently Kickstart

Introduction

This postmortem is written in a context where the first sprint and the layout for the whole development of Rently, a condo management system, is completed. The plan for the 5 sprints was devised and some groundwork has been started. Particularly, technologies have been chosen, architecture diagrams have been developed and roles have been separated in the team.

What went wrong

1 - Auth Setup

Initial implementation was not up to the security standards, so we had to reimplement the whole authentication and authorization logic. We realized it had to be changed after we saw the refresh tokens endpoints not behaving in a way we expected it to. Instead of storing the refresh token in the HTTP-only cookies, we were instead exposing it to the user, making auth vulnerable to XSS and CSRF attacks. It was fixed by implementing an HTTP-only mechanism and storing it in the cookies. Although it was fixed in the end, it is a negative point because of the time it took us to fix it, which could have been allocated to other tasks.

2 - Redux setup problem

The Redux setup issue, characterized by an overly complex and inefficient state management system, was primarily due to a lack of optimal use of Redux Toolkit features. This led to performance issues and increased development time. The problem arose from an initial misunderstanding of Redux Toolkit's capabilities, especially around state optimization and asynchronous logic handling. To address this, the team refactored the setup by applying best practices, such as using selector functions and the createSlice and createAsyncThunk utilities. However, a more streamlined solution was found in leveraging React Context, which provided a simpler and more direct way to manage state without the overhead of Redux, significantly improving maintainability and performance. This experience underscored the importance of selecting the right tools for project needs and the benefits of a simpler approach for state management.

3 - Tight timeline

We found the timeline to be very tight to have code implemented, as well as having all the documents ready. We struggled with having all the types of diagrams, some of them we had never worked with before, produced at the same time as we had to handle the JIRA board and the coding. Our good separation of tasks helped us a lot, since everyone was working on something different.

4 - Jira & Confluence Setup

What went wrong?

The project's Jira and Confluence setup was initially ineffective, leading to operational inefficiencies. Key issues included default configurations that didn't meet project needs, insufficient training on tool capabilities, and inconsistent adoption across the team.

Why did it happen?

The setup fell short due to inadequate configuration for the project's specific requirements, lack of comprehensive training for the team, and the absence of a clear strategy for tool adoption.

Impact

This resulted in decreased productivity, communication breakdowns, and the perception of resource wastage, as the tools did not deliver the expected improvements in efficiency and collaboration.

Resolution

Actions taken to address these issues included customizing the configuration of both tools, conducting detailed training sessions, and enforcing usage policies to ensure consistent application across the project.

Improvement Opportunities

Better initial planning, including a detailed needs assessment, consultation with tool experts, and regular reviews for adjustments, would have significantly enhanced the effectiveness of Jira and Confluence in the project workflow.

5 - Inconsistent Code Review Processes

What went wrong?

The project suffered from inconsistent code review processes, leading to uneven code quality and delays in feature deployment. Without a standardized approach, code reviews varied significantly in thoroughness and effectiveness, allowing bugs and coding standard violations to slip through.

Why did it happen?

The inconsistency arose from a lack of formalized code review guidelines and training. Reviewers had varying levels of experience and different perspectives on what constituted quality code, and there was no system in place to ensure uniformity in the review process.

Impact

This led to several negative outcomes, including increased bug rates in production, longer development cycles due to the need for additional rounds of revision and testing, and frustration among team members over the perceived arbitrariness of the review process.

Resolution

To address the problem, the team implemented a set of standardized code review guidelines and provided training sessions on best practices. Additionally, a code review checklist was introduced to ensure that all reviews covered essential quality criteria consistently.

Improvement Opportunities

A more structured approach from the start, with clear code review standards and regular reviewer training, could have prevented these issues. Integrating automated code quality tools into the development workflow might also have helped maintain consistency and reduce the workload on human reviewers.

What went right

1 - Team role separation

The team was separated appropriately for the project, according to each member’s strength. This meant that we were able to all start working on what we need to do without having people be forced to learn new technologies in a short time span. What was also good is that we designated leaders to manage the backend and frontend subteams. This eased the separation of tasks, where instead of one person managing 9 people, we had 1 person managing 4-5 people in their subteam.

2 - Technology selection

It was easy for us to choose the technologies, thanks to our familiarity with some frameworks over others. We chose Spring Boot for the backend due to its great ability to separate concerns. Various layers allow us to keep a structured code and produce maintainable code. For the frontend, we chose React, as it is the framework most people know.

3 - Design of the system

We are happy with how we designed our system, most specifically our class and domain diagrams. We were able to apply our knowledge from previous courses to this project and have something developed fast. Our domain model covered all the core aspects of the project and we kept in mind modularity and future changes. It was straightforward and we didn't struggle on it. The only thing that took a bit more reflexion was on assigning responsibilities to the classes. We did not know for certainty whether to assign some responsibilities to the Company or the Property. We made decisions based on how we thought it would be difficult to implement.

4 - JIRA board handling

After the difficulties faced in the setup, we found it is easy to use JIRA once we look past the initial setup. The JIRA board helped us familiarize more with how this environment works. We assigned 2 people to work on the board to keep it clean, and prepared documents to help the other members of the team navigate through JIRA. We found it really helpful to see that we can include the id of a task in a commit or branch name to have it automatically tracked by JIRA. It was impressive to see. We also enjoyed the automations that change the status of tasks based on whether the id is in a commit, a branch or a pull request. For example, the status goes from "In Progress" -> "In Review" when there is a pull request that has the id of that task. Similarly, the status changes from "In review" -> "Done" when a pull request is accepted.

Conclusion

Conclusion: Rently Kickstart Overview
The initial sprint of the Rently project, aimed at creating a comprehensive condo management system, has been a period of significant learning and achievement. Despite facing challenges such as security flaws in authentication setup, inefficiencies in Redux configuration, tight timelines, difficulties with Jira & Confluence setup, and inconsistencies in code review processes, the team has made considerable progress.

Key Learnings:

- The importance of security from the start.
- The value of selecting the right development tools.

- The need for effective task management and team training.
- The benefits of consistent code quality checks.

Achievements:

- Efficient team role separation leveraging individual strengths.
- Successful technology selection enhancing backend and frontend development.
- Effective system design ensuring a solid project foundation.
- Improved project management through streamlined JIRA workflows.

Moving Forward:

This sprint has set a solid foundation and direction for the Rently project. The challenges encountered have provided valuable insights, shaping our strategies for improved efficiency and success in future sprints. With a focus on leveraging our strengths and applying the lessons learned, we are well-positioned to advance the development of Rently as a leading condo management solution.