HBO-ICT Graduation 2022-2023

GRADUATION REPORT

Date: 08/01/2024

Prepared by

THAT NHAT HUY TON

479984

Acknowledgements

I would like to express my deepest gratitude to several individuals whose invaluable support and guidance were instrumental in the successful completion of this graduation project.

First and foremost, I extend my heartfelt thanks to our CEO, Joey Teunissen, for not only recruiting me into the company but also for providing me with the opportunity to undertake this graduation project. Joey's unwavering support and mentorship, including his involvement in one-on-one interviews to align the project's requirements, were truly instrumental in shaping the direction of this endeavour.

I am also immensely grateful to our dedicated QA Manager, Naief Bitar, for his invaluable contributions. Naief's willingness to engage in detailed project discussions, adjust project requirements, and offer constructive feedback on mock-ups and wireframes significantly enriched the project's development.

I extend my appreciation to our Product Manager, William te Wierik, for his unwavering support throughout this journey. William's insightful feedback on the graduation wireframe and his contributions to brainstorming sessions were instrumental in refining project ideas and approaches.

Lastly, I would like to acknowledge my teacher and graduation supervisor, Eelco Jannink, whose guidance and expertise have been truly indispensable. Eelco's responses to my questions were consistently well-spoken and rational, providing me with invaluable insights and direction.

The collective support and expertise of these individuals have played a pivotal role in the successful realization of this project, and for that, I am profoundly grateful.

Abstract

This graduation report unveils the outcomes of a project aimed at enhancing OpusFlow, an ERP system by ProfitFlow B.V. The primary focus was to improve the process of gathering and managing customer feedback. The challenge stemmed from OpusFlow users lacking a direct feedback medium within the system, resorting to indirect methods such as phone or email. The report illuminates the research methods, development processes, and key findings involved in addressing this challenge.

The project, initially planned for completion in 20 weeks across four phases, faced delays leading to a granted extension of 10 weeks. Project planning followed the Kanban method, coupled with note-taking in the tool Notion. Code management occurred on GitLab, utilizing the Git branch method for feature and bug management. Rigorous quality management practices were applied to ensure the codebase and documents met high-quality standards.

The solution to the identified problem was the development of SnapFlow Feedback—an interactive popup form within the OpusFlow system, allowing users to provide feedback. Additionally, OpusFlow InsightHub was introduced, a dashboard separate from the ERP system, displaying feedback and integrating with task management apps for streamlined issue resolution. The chosen technology stack aligned with OpusFlow's current technology, featuring the serverless framework Next.js with AWS S3 integration for media file storage and integration with Linear for task management.

The project successfully implemented a solution that enabled feedback collection within OpusFlow and provided access to feedback through the dedicated InsightHub dashboard.

In conclusion, the project not only met its objectives but also introduced a valuable solution for enhancing the user feedback process within OpusFlow. Lessons learned throughout the project underscored the importance of effective communication, adaptable project management, and stakeholder collaboration.

Table of Contents

List of Figures & Table

Glossary

Log Rocket: Log Rocket is a web application monitoring and debugging tool that allows developers to record user sessions on websites. It captures user interactions, errors, and console logs to help diagnose and troubleshoot issues effectively.

Linear App: Linear is a web-based task and project management application designed to streamline and improve the efficiency of task management and collaboration within development teams. It offers features for creating, assigning, and tracking tasks.

ERP (Enterprise Resource Planning): ERP is a comprehensive software system used by organizations to manage and integrate core business processes, such as finance, human resources, inventory, procurement, and project management, into a unified platform.

UI/UX: UI (User Interface) and UX (User Experience) collectively refer to the design and interaction aspects of a software application or website. UI focuses on the visual design and layout, while UX encompasses the overall user experience, including usability and user satisfaction.

SaaS (Software as a Service): SaaS is a software distribution model where software applications are hosted in the cloud and made available to users over the internet. Users access SaaS applications via web browsers, eliminating the need for on-premises installation.

B2B (Business-to-Business): B2B refers to a type of commerce where businesses sell products or services to other businesses rather than to individual consumers. It often involves transactions between suppliers, manufacturers, wholesalers, and distributors within the business ecosystem.

1. Introduction

1.1 Background & Context

The graduation project, conducted from May 8, 2023, to January 28, 2024, is a partnership with ProfitFlow B.V.—a software engineering firm that delivers digital solutions for businesses in the sustainable sector, including those involved in solar panel installation and wind turbine maintenance.

The project centered on improving the process of handling customer feedback for OpusFlow, an Enterprise ERP SaaS product by ProfitFlow. OpusFlow caters to B2B interactions, providing a broad range of features for sustainable energy installation companies. These features encompass project management, resource planning, procurement, and inventory management, among others.

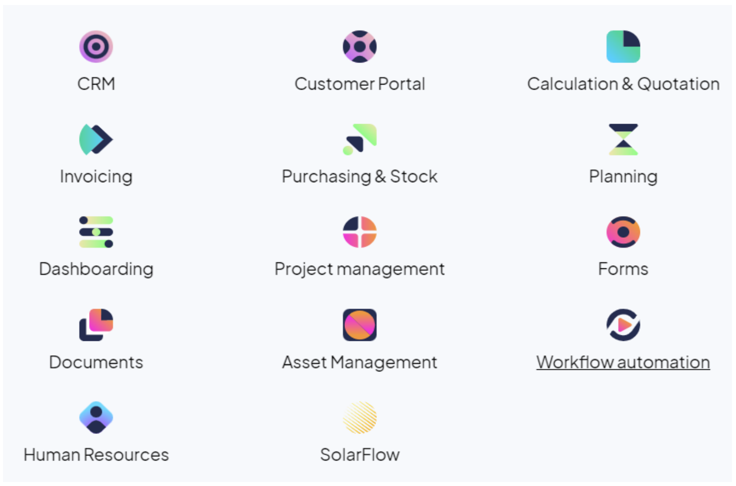


Figure 1: An Overall View of Features of OpusFlow System

Before commencing the project, a research phase delved into OpusFlow's current feedback process, resulting in the following flowchart based on the findings:

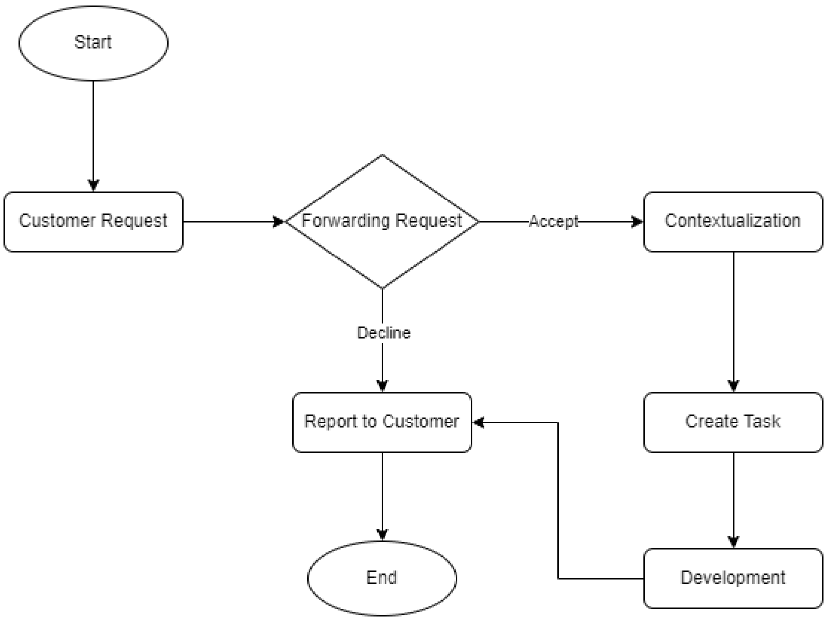


Figure 2: Flow Chart of Current Process Handling Customer's Feedback Process

The current process of handling customers’feedback involved several steps:

1. **Customer Request:** The customer initiates a request or issue a bug while using OpusFlow system**.**
2. **Forwarding Request:** The customer forward the request to the consultancy team through email or phone call.
    - If the request is accepted, the flow proceeds to the **Contextualization** step.
    - If the request is declined, it moves directly to the **Report to Customer** step.
3. **Contextualization:** The consultancy team reviews and contextualizes requests.
4. **Create Task:** The consultancy team creates a task in the task management tool and assign it to the development team
5. **Development:** The development team works on the solution.
6. **Report to Customer:** The consultancy team report the result the request outcome to customers.

Initial research revealed a gap between the first and second steps, where customers primarily relied on indirect channels like emails, phone calls, and a file-sharing platform called Hellonext. Serving as a dedicated feedback and feature request platform, Hellonext acts as a centralized hub, offering users a space to submit and discuss ideas, report issues, and efficiently prioritize features.

The pre-project research underscored that OpusFlow's existing practices fell short due to the lack of direct feedback acquisition methods. Studies indicate that incorporating website feedback through on-page surveys or widgets enhances user engagement and yields high-quality insights (Kanika, n.d.). Recognizing this, the introduction of a direct feedback mechanism within OpusFlow holds the potential to significantly enhance its feedback process.

1.2 Research questions & Goals

In order to, reach the purpose of improving the current system, the main questions the project try to solve is:

"How can a solution be developed to efficiently collect user feedback and translate it into meaningful improvements for the current system?"

To help answer the main question, the following sub-questions have been formulated:

1. What is the current method of handling user feedback after accepting customer requests in OpusFlow?
2. What enhancements can optimize OpusFlow's current feedback handling process?
3. Which features should be prioritized for inclusion in the minimum viable product (MVP) of the feedback collection solution?

In line with these questions, the project aimed to achieve the following objectives:

1. Develop a user-friendly solution suitable for OpusFlow customers.
2. Ensure the solution is fully compatible with the current system.
3. Create a visualization dashboard to display an overview of the collected feedback statistics.
4. Thoroughly document the entire project, encompassing development, implementation, and outcomes.

2. Project Management

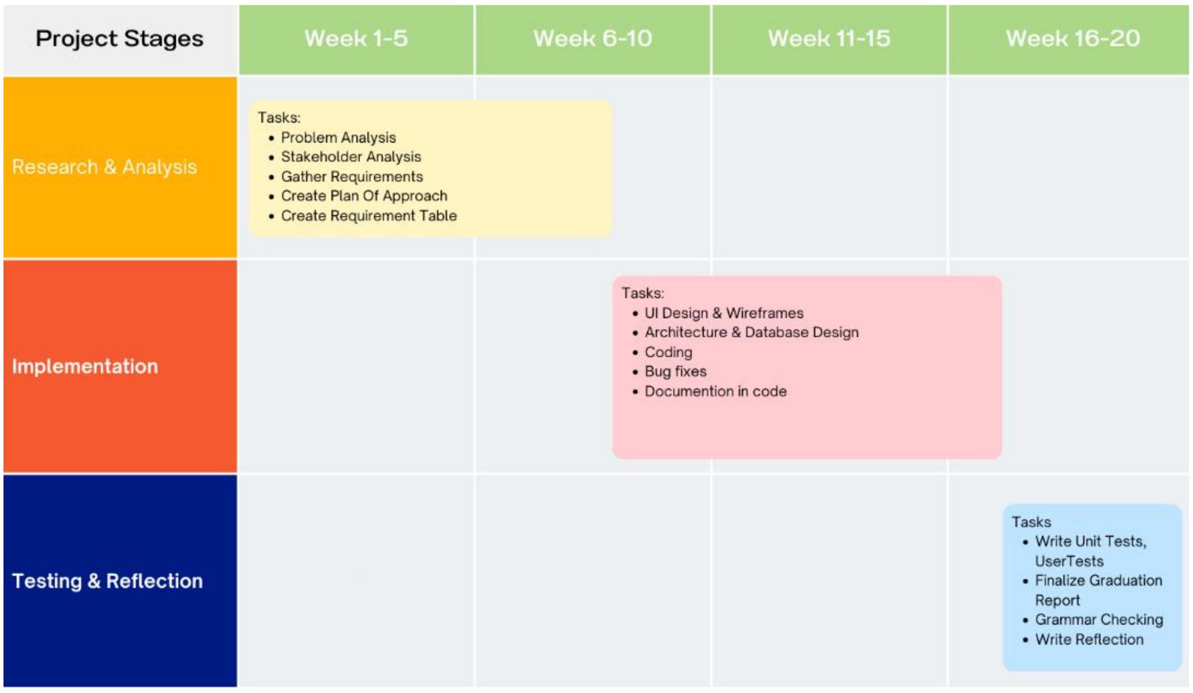
In this segment, the focal point is on the essential aspects of project management within the context of the graduation project. Examination of pivotal components, including planning, methods, code management, quality assurance, and deliverables at the end of the project. The ensuing analysis endeavours to provide a comprehensive and systematic understanding of the approaches and execution strategies employed in handling these elements. The objective is to offer insights into the strategies adopted for a successful project outcome.

2.1 Planning

The project was structured into three primary stages:

1. **Research:** This stage was dedicated to gathering pertinent information about the topic and evaluating it in order to formulate a plan and generate project requirements. The ultimate goal of this phase was to compile a checklist of requirements that would guide the work in the subsequent stage of the project.
2. **Implementation:** Upon completion of the research phase, the gathered information was utilized in the design of the flows and architecture. This resulted in the creation of wireframes and database designs, which served as the foundational structures for the front-end and back-end aspects of the project. The coding process, inclusive of bug fixes and code documentation, also took place during this stage.
3. **Testing & Reflection:** The final stage aimed to write tests to ascertain whether the requirements were met and the main research question was answered. This stage also involved the process of reflection on the project's outcomes and processes.

The phase planning chart provides a visual representation of the tasks within each stage, illustrating the sequence of task execution from the project's inception to its completion.

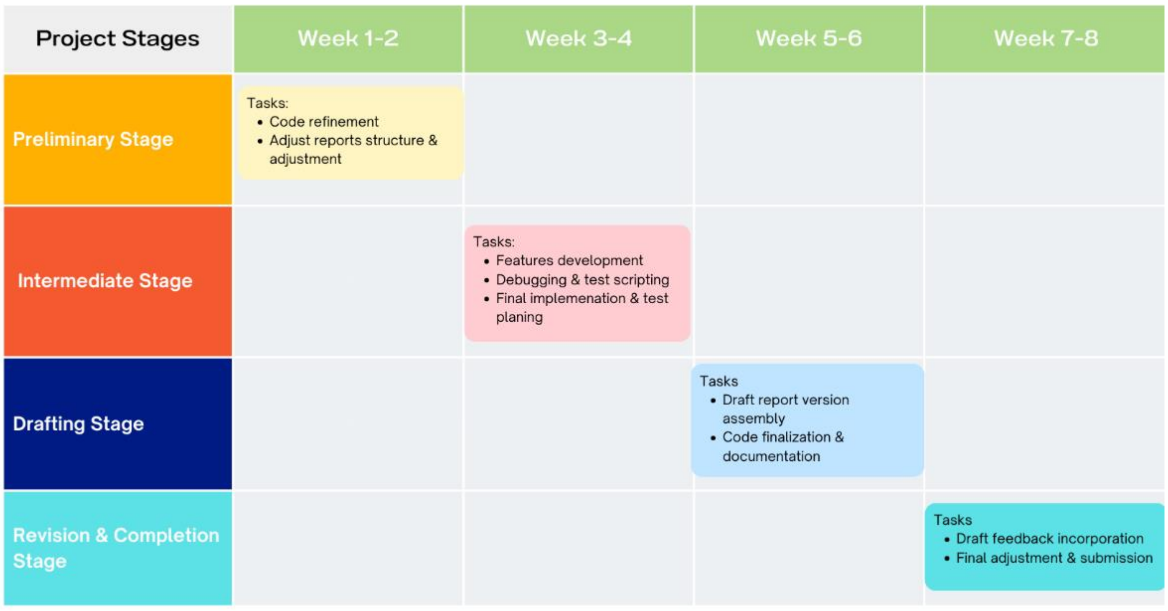


Project Stages	Week 1-5	Week 6-10	Week 11-15	Week 16-20
Research & Analysis	Tasks: • Problem Analysis • Stakeholder Analysis • Gather Requirements • Create Plan Of Approach • Create Requirement Table			
Implementation		Tasks: • UI Design & Wireframes • Architecture & Database Design • Coding • Bug fixes • Documention in code		
Testing & Reflection				Tasks • Write Unit Tests, UserTests • Finalize Graduation Report • Grammar Checking • Write Reflection

Figure 3. Phase Planning Chart

Due to unforeseen challenges and the nature of the project, several tasks outlined in the initial plan faced delays, resulting in their non-completion within the projected timeline. These tasks primarily revolved around finalizing specific project features, structuring and completing the project report, and the formulation of comprehensive tests for the project. A detailed examination of the challenges that contributed to the deviation from the original plan will be explained in the reflection section of this report.

Recognizing the critical nature of these tasks, they have been incorporated into the extended plan. This plan, therefore, reflects not just new tasks necessitated by the project's evolving requirements, but also includes those tasks from the original plan that were not accomplished. This duplication is intentional and serves to underscore the continuing relevance of these tasks to the overall project objectives.



Project Stages	Week 1-2	Week 3-4	Week 5-6	Week 7-8
Preliminary Stage	Tasks: • Code refinement • Adjust reports structure & adjustment			
Intermediate Stage		Tasks: • Features development • Debugging & test scripting • Final implemenation & test planing		
Drafting Stage			Tasks • Draft report version assembly • Code finalization & documentation	
Revision & Completion Stage				Tasks • Draft feedback incorporation • Final adjustment & submission

Figure 4. Project Extended Planning Chart

In the extended plan, these tasks have been repositioned within the four phases: Preliminary Stage, Intermediate Stage, Drafting Stage, and Revision & Completion Stage. The extended timeline allows for a more comprehensive approach to each task, ensuring that all critical aspects of the project—both old and new—are adequately addressed. This includes the completion of the project features, the improvement of the report structure and content, and the writing of necessary tests for the project.

The revised structure of the extended plan is designed to ensure that the final project goals are achieved more effectively, despite the challenges faced in the initial stages.

2.2 Working methods

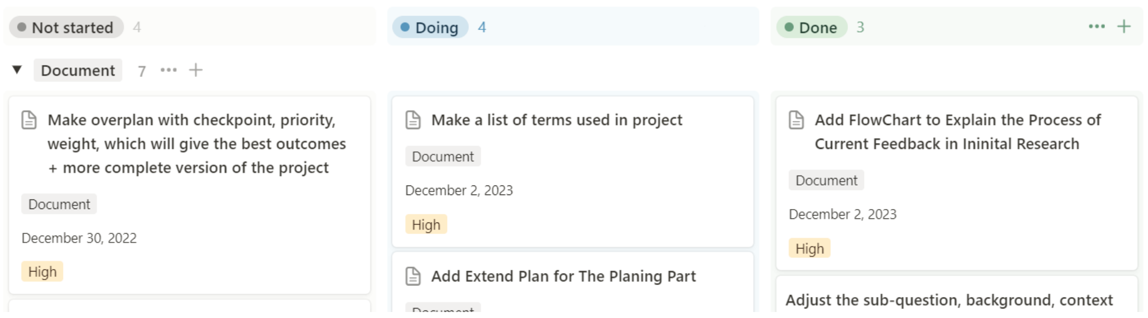
The project management methodology selected for this project was Kanban, an agile workflow management model that accentuates the visualization of processes to optimize workflow, enhance efficiency, and reduce waste (Kanban Tool, n.d.). The choice of Kanban was primarily due to its inherent flexibility and adaptive capacity, which were essential for this project's collaborative nature. The project engaged various participants at different stages, such as supervisor teacher, corporate supervisors & app users, necessitating a method that can rapidly accommodate new information and shifting requirements. Furthermore, the project required iterative reflection and modifications based on feedback as it advanced towards completion. Therefore, Kanban, with its emphasis on continuous improvement and responsiveness to fluctuations, was deemed the most suitable methodology.

Scrum was another project management methodology considered. However, in an article by Team (n.d.), it suggests that Scrum and Kanban differ fundamentally in their orientation. While Scrum is process-centric, Kanban predominantly focuses on project visualization. For instance, Scrum operates in sprints, which are time-boxed iterations usually lasting 1-4 weeks. This means that once a sprint has started, its scope is fixed and cannot be changed. However, in this project, the requirements and tasks were not fully predictable and fixed. The involvement of different stakeholders like teachers and corporate supervisors meant that new information and requirements could come in at any time, requiring immediate attention and changes to the project scope.

In contrast, Kanban's flow-based approach allows work items to continuously enter the workflow. Its emphasis on limiting work-in-progress (WIP) ensures that the team is only focused on a limited number of items at any given time, allowing for greater flexibility when new tasks or changes come in. This made it a more suitable choice for this project.

Additionally, Kanban excels in handling continuous task delivery until the project's completion, whereas Scrum is designed for delivering batches of tasks & relies heavily on predefined roles such as the Product Owner, Scrum Master, and Development Team. This project, however, did not have the luxury of such clearly defined roles, making the flexibility of Kanban a more fitting choice.

By using these specific scenarios, the choice of Kanban over Scrum for this project is further justified, highlighting the need for adaptability and responsiveness to changing requirements in the project.



Not started 4	Doing 4	Done 3
Document 7		
Make overplan with checkpoint, priority, weight, which will give the best outcomes + more complete version of the project Document December 30, 2022 High	Make a list of terms used in project Document December 2, 2023 High	Add FlowChart to Explain the Process of Current Feedback in Ininital Research Document December 2, 2023 High
	Add Extend Plan for The Planing Part	Adjust the sub-question, background, context

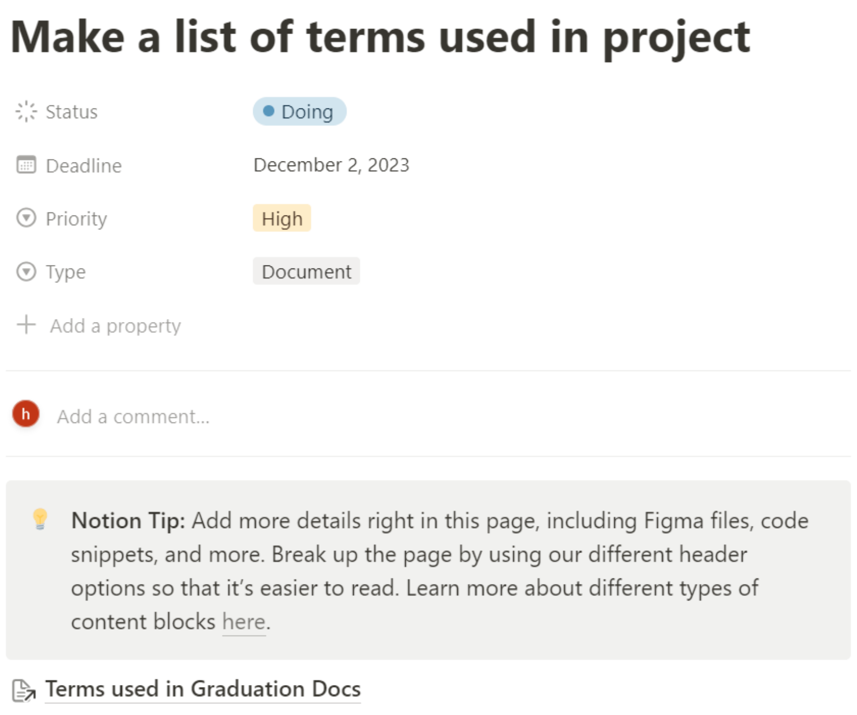
Figure 5. Notion Kanban Board for Task Management

In this project, the note-taking application, Notion, was used in conjunction with Kanban. Notion provides a template called the Kanban Board, which is optimal for the current working method. The project tasks are distributed across three main columns: "Not Started", "Doing", and "Done". These columns represent the lifecycle of a task from inception to completion, helping to visualize the progress of work.

Two distinct categories, "Document" & "Code", represent the two main types of tasks in the project. This categorization aids in distinguishing between different kinds of tasks and managing them more effectively.

Adding a new task is as simple as creating a new card in the "Not Started" column. As work begins on a task, it moves to the "Doing" column, and finally to the "Done" column upon completion. This movement from left to right gives a clear, visual representation of the project's progress.

Each task card can be expanded to a page providing detailed information about the task. This includes the title, status, priority, deadlines, and type of task. Design files, links, and task details can also be recorded here. This feature of Notion's Kanban Board allows for a comprehensive overview of each task and promotes better task management.



Make a list of terms used in project

Status	Doing
Deadline	December 2, 2023
Priority	High
Type	Document

Add a property

Add a comment...

Notion Tip: Add more details right in this page, including Figma files, code snippets, and more. Break up the page by using our different header options so that it's easier to read. Learn more about different types of content blocks here.

Terms used in Graduation Docs

Figure 6. Task Detail Page in Notion

Notion's Kanban Board has brought a systematic and organized workflow. It offers a clear visualization of tasks and their progress, and its flexible nature allows for quick adjustments in response to changing requirements or priorities.

2.3 Code Management

To ensure efficient management of the code base during the project, Git and GitHub were utilized as the primary tools for tracking the history of changes throughout each stage. GitHub, a web-based DevOps lifecycle platform, offers a distributed version control system, effectively facilitating code management even in collaborative projects. It allows tracking of source code changes during the development process, promoting smoother and more organized revisions and modifications.

The key components of version control throughout the project were:

1. Branching: This function enables the development of specific application features by duplicating the source code. Any modifications made do not affect the original source code until the process of merging occurs.
2. Merging: This process involves the integration of branches with the original source code after thorough testing.

The following diagram illustrates a workflow incorporating Git and GitHub.

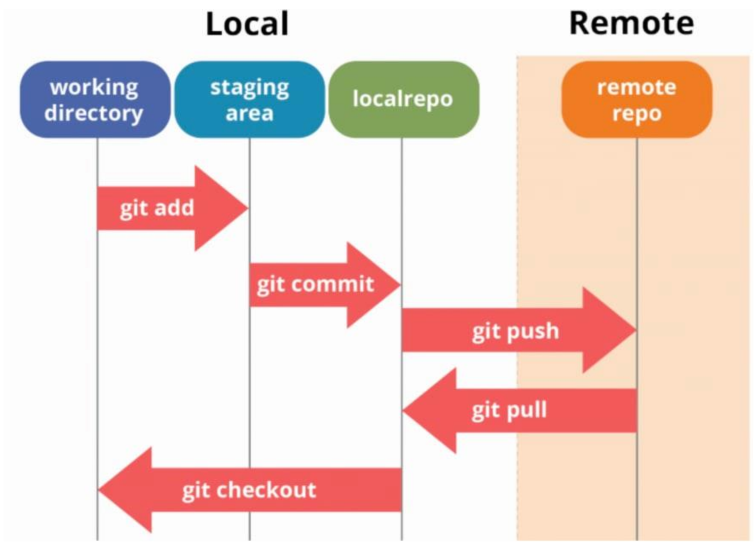


Figure 7. Flow Diagram of Working with Git & Github

The diagram list out important steps that using this the workflow:

- Git pull: This command fetches changes from a remote repository and merges them into the current branch. It's used to keep the local version of the code up-to-date with the changes made by other contributors.
- Git add: This command is used to add changes in the working directory to the staging area, marking the modified files in the current directory ready for the next commit.
- Git commit: This command captures a snapshot of the project's changes, creating a new commit object in the local Git repository. It's often accompanied by a message describing the changes.
- Git push: This command transfers commits from the local repository to a remote repository. It's used to share your changes with other contributors.
- Git checkout: This command is used to switch between different versions of the codebase. It can switch between branches or revert a file back to a previous state.

To manage the development of new features and bug fixes, a specific branching strategy was implemented.

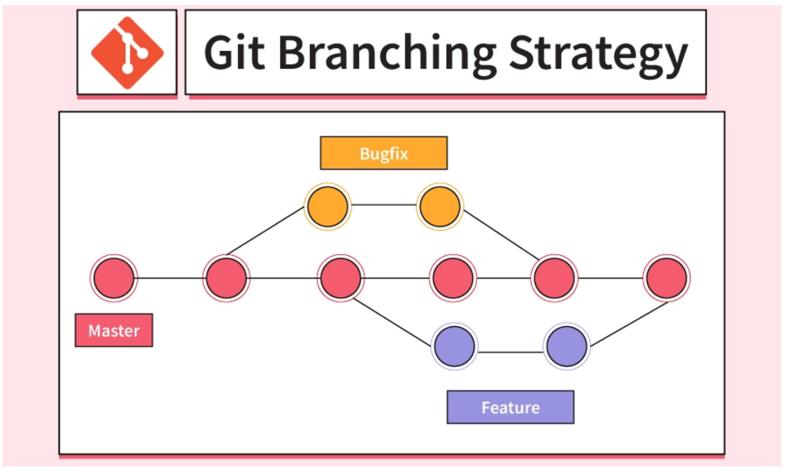


Figure 8. Git Branch Strategy Diagram

The project begins by creating a 'master' branch in the Git repository. This branch serves as the stable version of the codebase, providing a reliable baseline unaffected by ongoing developmental changes. This choice is made to ensure a consistent and dependable starting point for the development process.

When the necessity arises to develop a new feature or address a bug, a streamlined process is followed. New branches are created using the commands **git checkout -b feature/feature-name** or **git checkout -b bugfix/bug-name**. This simple process duplicates the 'master' branch, establishing an isolated space for individual development. The structured naming conventions are employed for clarity – a branch for feature development is prefixed with "feature" followed by the feature name, while a branch for bug fixes is prefixed with "bugfix" followed by the specific bug name. The rationale behind this approach is to maintain an organized and easily understandable branch structure.

Subsequently, following the established workflow, code is added to the branch locally. The **git add** and **git commit** commands are then utilized to stage and commit changes, accompanied by a descriptive commit message. The **git push** command is employed to upload the branch and its changes to the remote repository. This approach ensures that changes made during solo development are appropriately documented and tracked.

Before merging the branch into the 'master' branch, a **git pull** from the 'master' branch is executed. This step is crucial to update the branch with any changes made to the main codebase, aligning the solo development with the latest codebase state. In cases of conflicts, necessary conflict resolutions are performed, followed by another **git push** to update the remote repository. This choice is made to ensure that the solo development is in sync with the latest changes in the main codebase.

A pull request is then initiated, listing all the changes made in the branch. After addressing a bug and completing the code review, only the essential changes related to that bug are carefully selected and added to the deployed release branch. The selection process ensures that only crucial modifications for fixing a specific issue are included in the stable release.

2.4 Quality Management

Quality management was a top priority in this project, and it applied to both the code and the project's documentation.

1. Coding Standards: Quality management stands as a paramount concern in this project, extending its focus to both the code and the accompanying project documentation. The adherence to coding standards becomes a pivotal aspect of this commitment, emphasizing the creation of a high-quality code repository. The approach involves:

    - SOLID Principles Integration: The code strictly follows SOLID principles, a set of principles that enhance the code's maintainability and flexibility. By emphasizing Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion, the codebase becomes more robust and adaptable. These principles guide the development process, promoting clarity and facilitating future modifications, resulting in a well-structured and efficient codebase.

    - TypeScript's Type Safety: Leveraging TypeScript's type safety becomes instrumental in elevating code quality. By implementing strict typing, potential errors are effectively identified during the development phase, fostering the creation of a more reliable and resilient codebase.

    - Concise Functions with Single Responsibilities: The design philosophy revolves around creating concise functions with singular responsibilities. This not only enhances maintainability but aligns with the widely-accepted software design principle of "Don't Repeat Yourself" (DRY), mitigating unnecessary code duplication.

    - Structured Code Formatting: Prioritizing structured formatting, the project minimizes deep nesting to enhance readability. Additionally, code lines are kept succinct for ease of comprehension.

2. Testing: Unit testing was employed to verify the functionality of individual components, while manual testing was conducted based on specific scenarios to ensure the code met its intended objectives.

3. Static Code Analysis: ESLint and Prettier were utilized as static code analysis tools, helping to detect potential bugs, enforce a consistent coding style, and prevent potential issues before they could occur.

4. Documentation: Comprehensive documentation was meticulously maintained, detailing all aspects of the project - from project objectives to design, development process, and test results. Comments were included in complex functions and at the beginning of files to explain their purpose and functionality. This not only contributed to the maintainability of the code but also ensured that anyone reviewing the project would have a clear understanding of the processes and decisions involved.

5. Design Files: The project's design files were also carefully managed and organized. They were reviewed regularly to ensure they accurately reflected the current state and direction of the project.

Through these strategies and tools, high standards of quality were maintained for both the code and the project documentation. This ensured that the project was not only technically sound but also well-documented, making it easy to understand, modify, and improve in the future.

2.5 Deliverables

The culmination of this project is expected to yield several key deliverables, which are outlined as follows:

- Project Documentation: Comprehensive documentation will be provided, detailing all aspects of the project. This will include project objectives, design, development process, and test results. The project documentation will serve as a crucial reference for understanding and potentially replicating or building upon this project in the future.
- Codebase: The complete source code of the project will be delivered. The codebase will embody high-quality, fully commented, and thoroughly tested code, ensuring easy comprehension and potential reusability.
- Figma Design Files: All design files created using the Figma design tool will be delivered. They will encompass all the User Interface (UI) and User Experience (UX) designs, wireframes, and prototypes constructed during the project's design phase. These files will offer a visual and interactive representation of the project's design components.

3. Research

The Research phase is a key element of this project, along with Implementation and Testing aiming to establish the project's foundation. This involves defining project requirements through in-depth analysis and comprehensive research methods.

To recap, the primary question this project seeks to answer is:

"How can a solution be developed to efficiently collect user feedback and translate it into meaningful improvements for the current system?"

Supporting this main question are three sub-questions::

1. What is the current method of handling user feedback after accepting customer requests in OpusFlow?
2. What enhancements can optimize OpusFlow's current feedback handling process?
3. Which features should be prioritized for inclusion in the minimum viable product (MVP) of the feedback collection solution?

Each sub-question will be addressed methodically, with findings presented and summarized in a conclusion. The application of the Development Oriented Triangulations (DOT) framework reinforces the robustness of the methods employed. The collective findings from these sub-questions will contribute to a comprehensive solution to the primary research question.

3.1. Understanding the Current Feedback Handling Process

3.1.1. Introduction

To improve the user feedback process in OpusFlow, it is crucial to gain insights into the current methods used after accepting customer requests. This section focuses on addressing the sub-question:

"What is the current method of handling user feedback after accepting customer requests in OpusFlow?"

3.1.2. Approach

To enhance comprehension of the current feedback process and identify potential areas for improvement, interviews were conducted with the QA Manager, Naief Bitar of the OpusFlow system. A questionnaire was employed during the interviews to gather detailed insights into the specifics of the channels used, the roles of stakeholders, the integration of external tools, and the overall current feedback handling process within OpusFlow. The questionnaire included the following:

1. Could you provide an overview of the current process for managing user feedback in OpusFlow?
2. Who are the key stakeholders involved in the user feedback process, and what roles do they play?
3. Is there any external tool integrated into the feedback process? If so, how does the consultancy team utilize it to manage tasks based on customer feedback?
4. Can you elaborate on how a task progresses from the initial feedback stage to completion?

For comprehensive interview transcripts, refer to **Appendix A**.

3.1.3. Result

The results of this comprehensive investigation, guided by the questions raised in the questionnaire, are illustrated in the sequence diagram below. To provide context, it's crucial to highlight the role of Linear, a task management platform, which plays a pivotal role in the feedback process. This platform was seamlessly integrated into the OpusFlow system to enhance the workflow of the consultancy team in creating and managing tasks based on customer feedback.

This diagram effectively depicts the current flow of user feedback in OpusFlow from its initiation to completion, answering question 1, and the key stakeholders involved in the process along with their specific roles, answering question 2.

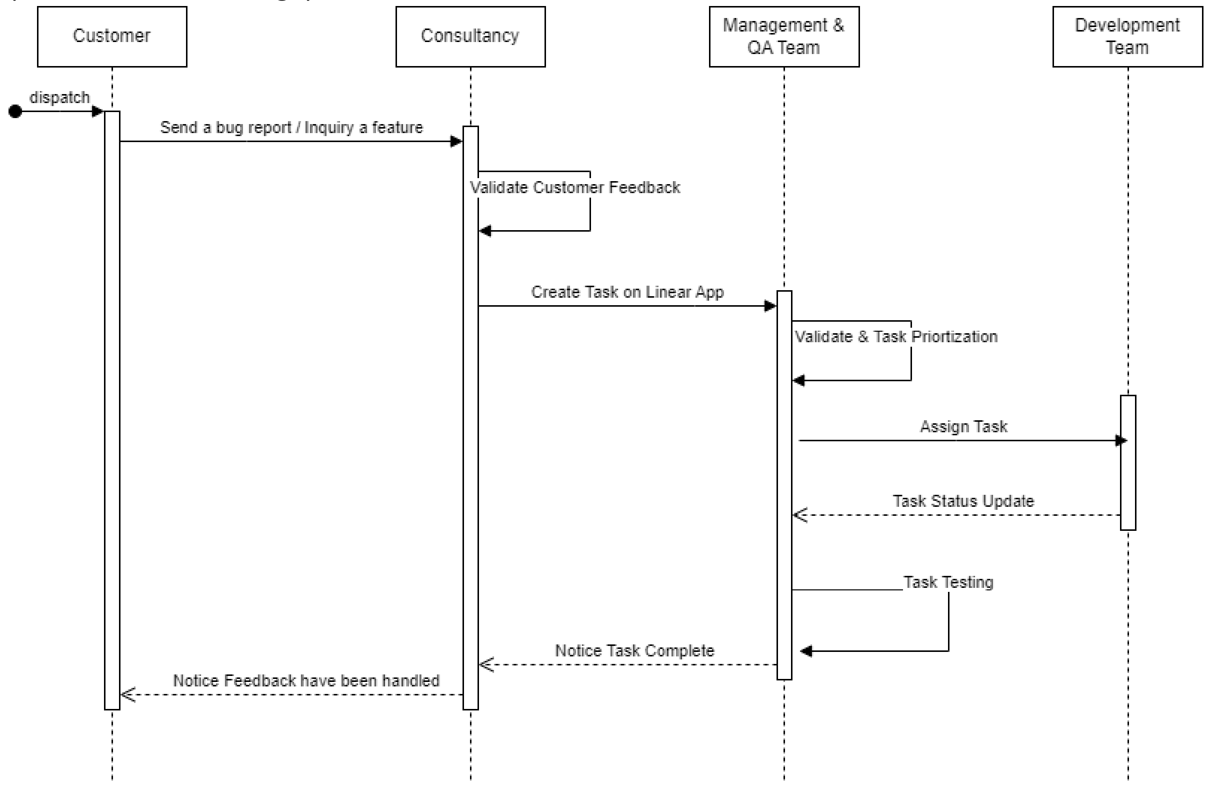

Customer
Consultancy
Management & QA Team
Development Team
dispatch
Send a bug report / Inquiry a feature
Validate Customer Feedback
Create Task on Linear App
Validate & Task Priortization
Assign Task
Task Status Update
Task Testing
Notice Task Complete
Notice Feedback have been handled

Figure 9. Sequence Diagram of Customer Feedback Process in OpusFlow

Based on this sequence diagram, two main user roles that are of particular relevance to this project were identified:

1. OpusFlow User (Solar Panel Installer, Wind Turbine Maintenance Team, Employees from Sustainable Companies): These users, the primary source of feedback, initiate the process by submitting their requests or issues. Their firsthand experiences with the system are invaluable for identifying potential areas of improvement and ensuring that the system continues to meet user needs and expectations.
2. Consultancy Team: This team serves as the frontline, receiving and analyzing the feedback, and creating tasks for the development team based on necessary changes or bug reports. Their pivotal role in translating user feedback into actionable tasks allows for a more effective and efficient feedback handling process, which addresses question 3.

The development team follows the Linear Method for handling tasks created from user feedback. This process involves writing issues for identified tasks, breaking them down as much as possible, assigning a single owner, and prioritizing them. The team works in 1 or 2-week cycles without specific deliverables, allowing flexibility and individual time management. Regular updates are made to the main branch and the implementation is made available in production. QA then ensures the issue is ready for a production release, then sets the Status to 'done'. This process answers questions 4 .

These two roles, the OpusFlow User and the consultancy team, are central to the feedback process in OpusFlow. Their direct involvement in the feedback loop, from initiation to task creation, makes them key players in this process. This project focuses on these roles as any improvements made to the feedback mechanism directly impacts them the most.

While other roles, such as the management & QA team and the development team, significantly contribute to the overall feedback handling process, they are not as directly involved in the feedback loop. Their roles focus on overseeing the process and implementing tasks, respectively. Although their work is integral to the process, changes to the feedback mechanism do not impact their work as directly.

3.1.4. Conclusion

This investigation provided a comprehensive understanding of the current situation, including a clear identification of the process and the roles involved. The sequence diagram and subsequent analysis of user roles not only confirmed the findings of the initial research but also provided a more comprehensive understanding of the existing feedback handling process.

3.2 Optimizing OpusFlow's Feedback Handling

3.2.1 Introduction

Building on the comprehensive examination of the OpusFlow system's current feedback handling process conducted in section 3.1, this section serves to dissect the implied values that the proposed feedback collection solution could potentially deliver.

The sub-question under focus here is:

"What enhancements can optimize OpusFlow's current feedback handling process?"

The aim is to pinpoint specific improvements and advancements that could offer over the existing system.

3.2.2 Approach

To make clear the intended capabilities of the project, an interview with the company's CTO was conducted. This approach yielded an understanding of the project's desired capabilities. The detailed notes from this interview played a pivotal role in developing user stories, defining user roles, and finalizing project requirements. For comprehensive interview transcripts, refer to **Appendix B**.

Key points from the interview that significantly influenced the project include:

1. **Enhanced User Experience:** The primary objective of the feedback mechanism is to enhance UI/UX. A solution that can significantly improve the user experience by making the feedback process simpler and more effective would add substantial value to the existing system.
2. **Granular Feedback:** The ability for users to provide feedback on a per-page or per-component level is highly valued. A solution that enables such granularity would streamline the feedback process and make it easier for users to identify and report issues.
3. **Abstract and Reusable Feedback System:** The expectation is to create a feedback system that can be reused across different pages and components without needing custom code. A solution that meets this requirement would significantly enhance system efficiency and maintainability.
4. **Actionable Insights:** The collected feedback should be presented in a way that provides actionable insights. A solution that can deliver this would greatly enhance decision-making and prioritization of improvements.
5. **Integration with Error Tracking Tools:** The integration of error tracking tools like Sentry and LogRocket is seen as beneficial. A solution that can align automated error reports with manual customer feedback would offer a more comprehensive view of system issues and help prioritize improvements.

3.2.3 Result

The interview provided in-depth user stories and user flows that outline the anticipated enhancements. The user stories and flows, along with their corresponding benefits, are outlined in the table below. While some of these user stories were directly derived from the interview, others were developed based on the understanding of the project's specific goals in providing a seamless, user-friendly feedback system. It should be noted that while the table provides a succinct summary of the requirements, the explanations provide a more comprehensive understanding of how these requirements contribute to the overall value of the proposed solution.

User stories

ID	As a	I want	So that
U1	OpusFlow User	the feedback process to be simple and effective	I can easily communicate my experiences and issues with the OpusFlow system.
U2	OpusFlow User	to be able to provide feedback on specific pages or components	I can help the OpusFlow identify and resolve issues more effectively.
U3	Consultancy Team	to receive actionable insights from user feedback	I can gain a more comprehensive understanding of system issues and prioritize improvements more effectively
U4	Consultancy Team	integrate management task Linear application with user feedback	I can create task based on user feedback
U5	Consultancy Team	view all collected feedback in a centralized dashboard	I can easily analyze and prioritize them.
U6	Consultancy Team	filter and sort feedback based on various criteria (e.g., date, page, component)	I can streamline my analysis.
U7	Consultancy Team	see a summary & charts of feedback trends	I can quickly identify areas of focus.
U8	Consultancy Team	see the link or connection between user feedbacks	I can get a comprehensive view of system performance and user experience

3.2.4 Conclusion

In conclusion, the analysis of OpusFlow and the subsequent user stories reveal the potential value of the proposed solution. These user stories, derived from the CTO's insights, highlight key improvements such as enhanced user experience, granular feedback, system efficiency, actionable insights, and effective error tracking. The solution should addresses the significant needs and expectations of the OpusFlow Users and Consultancy Team. It promises to streamline feedback, facilitate task creation, and provide a comprehensive feedback analysis dashboard, thereby significantly improving upon the existing system.

3.3 Prioritizing Features for MVP

3.3.1 Introduction

This section aims to address the sub-question:

"Which features should be prioritized for inclusion in the minimum viable product (MVP) of the feedback collection solution?"

Drawing on insights from previous sections, this part seeks to establish a comprehensive list of requirements for the MVP.

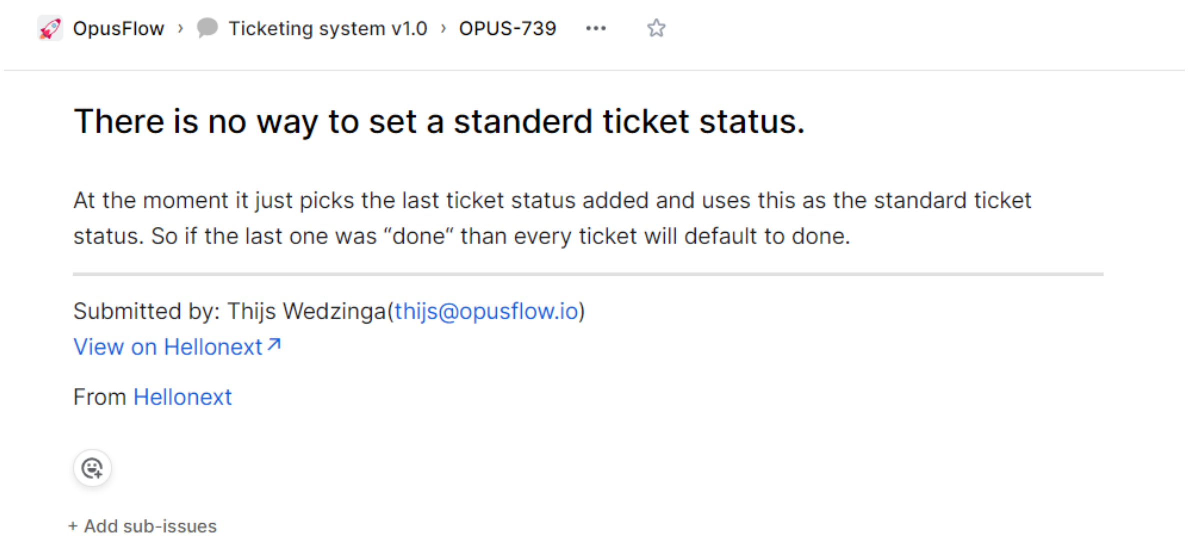
3.3.2 Approach

The sub-question is divided into two core interrogatives:

1. What key areas of the process need improvements as suggested by user feedback and operational challenges?
2. What potential strategies or tools can be employed to refine feedback management?

The first query relies on continuous interaction with Naief Bitar, the QA Manager of OpusFlow. It is apparent that the absence of a direct feedback feature in OpusFlow is problematic, as it requires users to exit the application to provide feedback, resulting in two primary issues:

- Ambiguous tasks often prompt the development team to seek further clarification, causing delays in feedback processing.
- The need to use additional tools for feedback disrupts user workflow and might discourage feedback provision.


OpusFlow › Ticketing system v1.0 › OPUS-739 •••

There is no way to set a standerd ticket status.

At the moment it just picks the last ticket status added and uses this as the standard ticket status. So if the last one was "done" than every ticket will default to done.

Submitted by: Thijs Wedzinga(thijs@opusflow.io)
View on Hellonext ↗

From Hellonext

+ Add sub-issues

Figure 10. Example of Current Task in Linear

In response to the second research query, an examination of industry best practices was undertaken to glean insights. The research conducted by Haije (2023) served as a foundational reference, emphasizing three primary objectives: real-time updates on performance and quality, enhancements in user experience (UX), and product adoption.

Achieving real-time updates involves incorporating feedback forms within the software. Additionally, integrating the feedback tool with issue tracking tools like Linear can streamline bug reports and task management.

Furthermore, feedback plays a pivotal role in contributing to UX improvements and fostering product adoption. For example, deploying active feedback forms to a subset of users allows for the collection of valuable insights into user preferences, guiding iterative product enhancements and aiding decision-making during product launches.

3.3.3 Result

Derived from meticulous interviews and the user stories previously developed have informed the creation of a comprehensive requirements table, use-case diagrams, use-case descriptions, and user flow diagrams have been developed. These crucial elements form the backbone of the project:

1. Functional Requirements: These detail the actions the system must execute.
2. Non-Functional Requirements: These specify the criteria the system must adhere to in its operations.
3. Use Case Diagrams: These are graphical depictions of user-system interactions.
4. Use Case Descriptions: These provide detailed sequences for each use case.
5. User Flow Diagrams: These illustrate the path a user follows through the system.

These components collectively provide a robust blueprint for the feedback module development, ensuring alignment with user needs and project objectives. The findings also underscore the importance of integrating direct user feedback within the software, incorporating issue tracking tools, and utilizing feedback for user experience enhancements and product adoption.

Functional requirement

ID	Requirement Name	Description	User Stories	Acceptance Criteria	MoSCoW
F1	Submission of Feedback	Facilitate users in communicating their experiences and issues related to any particular page or component of the OpusFlow system.	U1, U2	- Users can transmit their feedback from any page or component. - Users are notified upon successful transmission of feedback.	Must
F2	Capturing Context of Feedback	Enable users to incorporate the context, via screen recording or capture, while submitting feedback.	U2, U3	The context (e.g., page, component) is recorded along with each feedback entry.	Must
F3	Task Initiation in Linear App	Generate tasks using the information obtained	U4	- A pre-filled form is available based on the data in the feedback.	Must

		from feedback in the Linear application.		- Two types of forms, namely Request for Change (RFC) and Bug Report, must be accessible for immediate use. - The tasks created through this process shall be visibly displayed within the Linear application interface.	
F4	Centralized Feedback Dashboard	Establish a consolidated platform for viewing, filtering, and sorting feedback.	U5, U6, U7	- All feedback is accessible in a centralized location. - Feedback can be filtered and sorted based on various parameters.	Should
F5	Feedback Correlation View	Provide a mechanism to visualize the association or link between different pieces of user feedback.	U8	- Different pieces of user feedback can be viewed in conjunction with each other. - Patterns and areas of concern can be identified through this correlation.	Could

Note:

- "Must" indicates that the requirement is fundamental and the project would be considered a failure without it.
- "Should" means the requirement is important but not vital. The project's success doesn't rely on it, but it should be included if possible.
- "Could" suggests the requirement is a nice-to-have feature. It's not necessary for the project's success and can be implemented if there's time.
- "Would" (not used in the table) typically represents a requirement that stakeholders agree will not be implemented in the current release but will be considered for the future.

Non-functional requirement

ID	Req. Name	Description	Rationale	Measurement
NF1	System Performance	The feedback module ought to respond to user interactions within a time frame of 0.5 seconds.	This is crucial to ensure seamless user experience.	Response time for user interactions is measured.
NF2	System Usability	Users ought to be facilitated to submit	This is pivotal to encourage feedback	The count of interactions required to initiate

		feedback within no more than 3-5 interactions.	submission by simplifying the process.	feedback is measured.
NF3	Data Security	All data pertaining to feedback should be encrypted both during transit and at rest.	This is integral to safeguard sensitive user feedback.	The employment of encryption protocols and methods is verified.

User flow

User Story 1: Submission of Feedback on a Specific Page or Component

In the capacity of an OpusFlow User, the objective is to contribute feedback on a distinct page or component, facilitating the communication of any issues or potential suggestions.

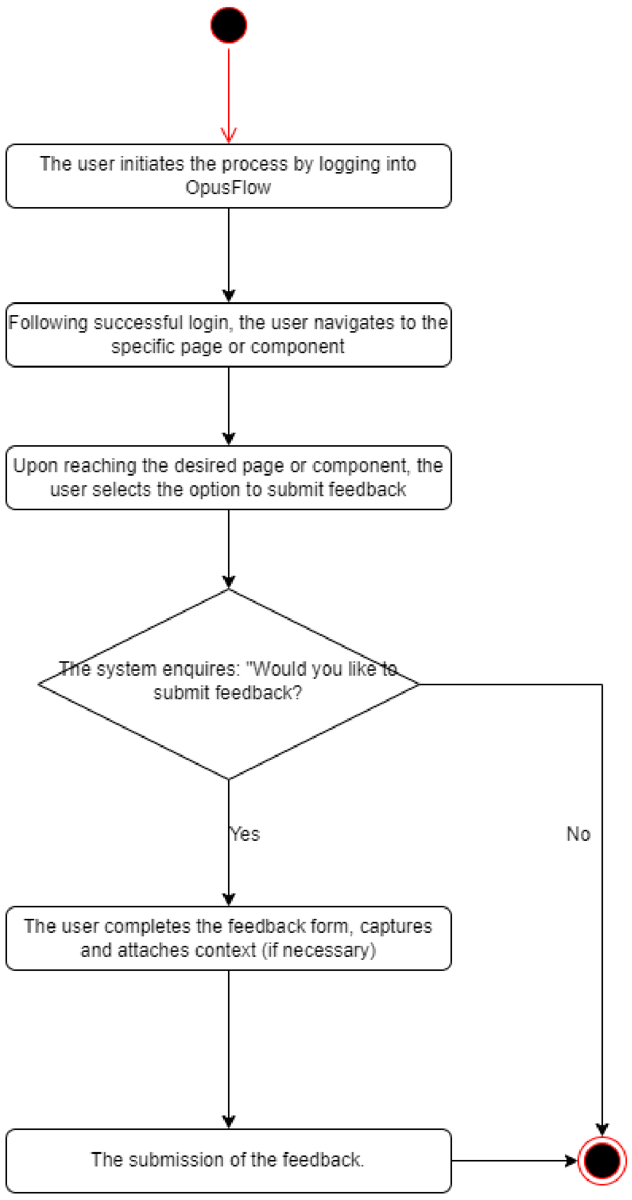
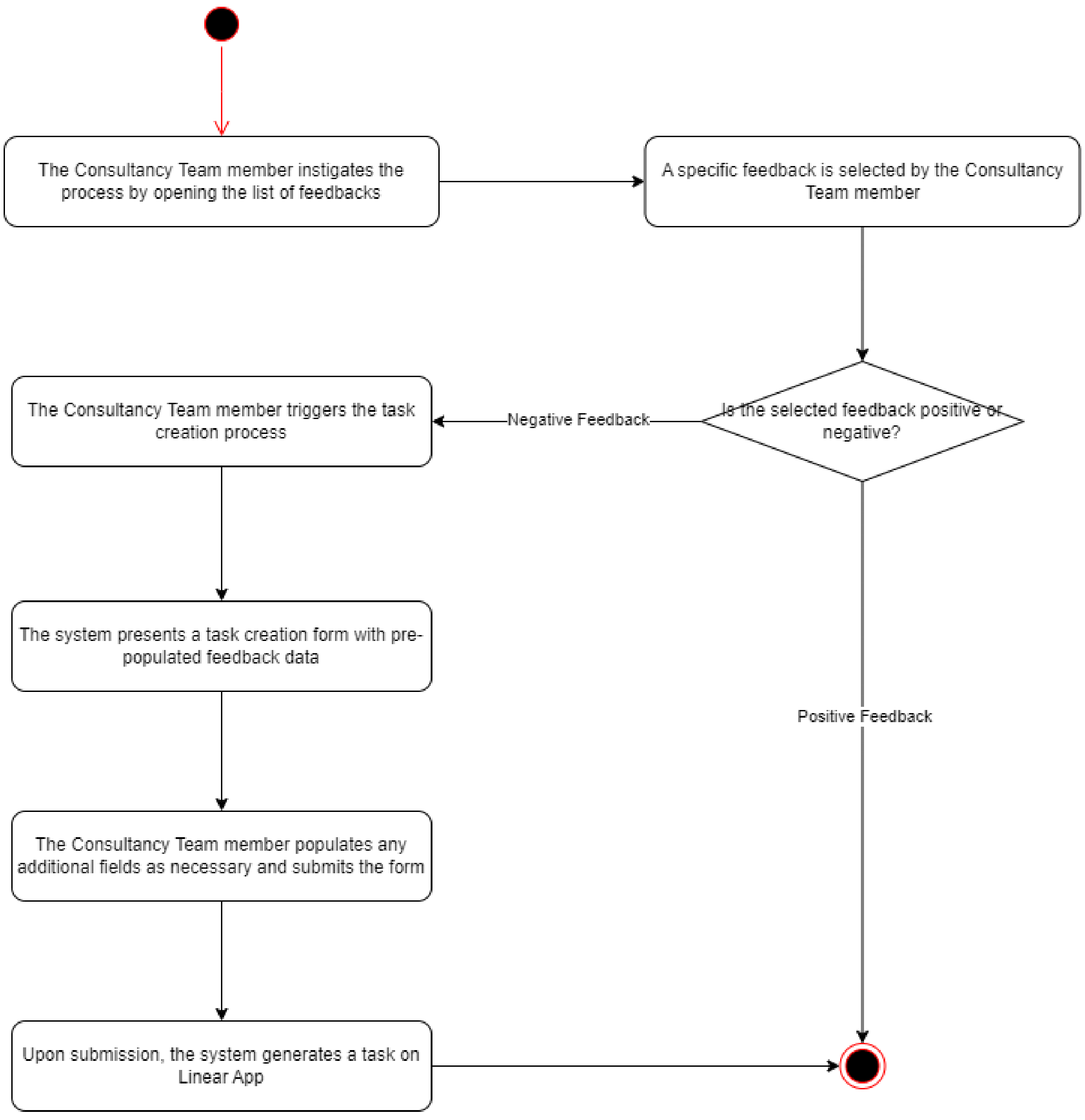

The user initiates the process by logging into OpusFlow
Following successful login, the user navigates to the specific page or component
Upon reaching the desired page or component, the user selects the option to submit feedback
The system enquires: "Would you like to submit feedback?
Yes
No
The user completes the feedback form, captures and attaches context (if necessary)
The submission of the feedback.

Figure 11. Activity diagram for submit a feedback

User Story 2: Task Creation on Linear from Feedback Dashboard

As a representative of the Consultancy Team, the goal is to log into the Feedback Dashboard, enabling access to the feedback page, reviewing the list of tasks, and instigating the creation of a Linear Task with pre-populated data derived from the feedback.



The Consultancy Team member instigates the process by opening the list of feedbacks
A specific feedback is selected by the Consultancy Team member
Is the selected feedback positive or negative?
Negative Feedback
The Consultancy Team member triggers the task creation process
Positive Feedback
The system presents a task creation form with pre-populated feedback data
The Consultancy Team member populates any additional fields as necessary and submits the form
Upon submission, the system generates a task on Linear App

Figure 12. Activity diagram for create a task from feedback

Use Case Diagram

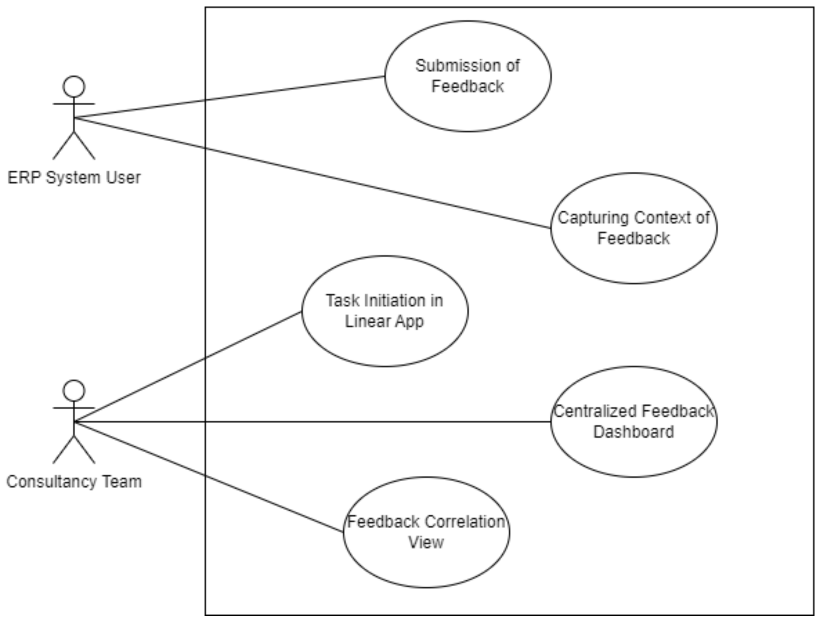

ERP System User
Consultancy Team
Submission of Feedback
Capturing Context of Feedback
Task Initiation in Linear App
Centralized Feedback Dashboard
Feedback Correlation View

Figure 13. Use Case Diagram of Actors in OpusFlow Monitoring System

The Use Case diagram offers a comprehensive overview of the system's functionalities, alongside the interactions of various roles (actors) with these functionalities (use cases). This diagram is designed to visually encapsulate the primary actions the system facilitates and the manner in which different users engage with these actions.

Actors:

1. **OpusFlow User**: Represents users like solar panel installers, wind turbine maintenance teams, and other employees from sustainable companies who use the OpusFlow ERP system for their daily operations.
2. **Consultancy Team**: This team acts as the frontline support for OpusFlow Users. They analyze user feedback, provide assistance, and suggest tasks to the development team. Their role also involves reviewing user feedback and creating tasks in Linear app to ensure timely resolution.

Use case descriptions

These Use Case Descriptions provide an in-depth delineation of each use case, detailing the interactions between users and the system. They function as a guide for comprehending and implementing the functionalities represented in the Use Case Diagram.
The use cases were structured as follows:

- **Use Case No. :** This is the unique identifier.
- **Use Case Name:** This is the title of a specific scenario that describes how a user interacts with the system.
- **Primary Actor:** The main user or role that initiates the use case and interacts with the system.

- **Description:** A brief overview of what the use case is about and what it aims to achieve.
- **Pre-conditions:** The conditions that must be met or true before the use case can start.
- **Main Flow:** The step-by-step sequence of interactions between the primary actor and the system to accomplish the use case.
- **Post-conditions:** The outcomes or changes that occur in the system after the successful completion of the use case.
- **Related User Stories:** These are specific examples that dive deeper into the interactions and actions of users within the use case, offering more detailed insights into their needs and goals.
- **Related Functional Requirements:** These are the specific functionalities that the system should possess in order to fulfill the use case, providing a link between the use case and the technical requirements of the system.

Use Case No.	**Use Case 1**
Use Case Name	User Feedback Submission
Primary Actor	OpusFlow User
Description	This use case enables the OpusFlow User to contribute feedback on specific pages or components within the OpusFlow system via the feedback module.
Pre-conditions	The user has successfully logged into the OpusFlow system and is engaging with a page or component.
Main Flow	1. The user navigates to a specific page or component on the OpusFlow system. 2. The user accesses the feedback submission feature. 3. The user inputs their feedback and submits it.
Post-conditions	The user receives a confirmation of successful feedback submission
Related User Stories	U1, U2
Related Functional Requirements	F1 (Submission of Feedback), F2 (Capturing Context of Feedback)

Use Case No.	**Use Case 2**
Use Case Name	Feedback Analysis
Primary Actor	Consultancy Team
Description	This use case outlines the process by which the Consultancy Team accesses, analyzes, and prioritizes improvements based on received feedback
Pre-conditions	Feedback has been submitted by OpusFlow Users and is available for review
Main Flow	1. The team accesses the centralized feedback dashboard. 2. The team filters and sorts the feedback based on various criteria. 3. The team views a summary and charts of feedback trends. 4. The team identifies areas of focus and creates tasks in the Linear app based on the feedback.
Post-conditions	Tasks based on user feedback are created and designated to the RFC team.

Related User Stories	U3, U4, U5, U6, U7, U8
Related Functional Requirements	F4 (Task Initiation in Linear App), F5 (Centralized Feedback Dashboard), F6 (Feedback Correlation View)

3.3.4 Conclusion

In conclusion, the process of addressing the sub-question, "Which features should be prioritized for inclusion in the MVP of the feedback collection solution?" has led to significant insights. The identified features, including direct feedback integration, task initiation in Linear app, and a centralized feedback dashboard, are critical for enhancing the quality, user experience, and adoption of the product. The development of a comprehensive requirements table, use-case diagrams, use-case descriptions, and user flow diagrams form a robust blueprint for the development of the feedback module. This ensures alignment with user needs and project objectives, promising a user-oriented, efficient feedback collection solution.

3.4 Final Conclusion

The Research phase, constituting a critical component of this project, aimed to establish the groundwork for the latter phases - Implementation and Testing. This was achieved by defining the project's requirements through systematic analysis and rigorous research methodologies, with a focus on addressing the sub-questions delineated in the Research Question and Objectives section.

The primary research question - "How can a solution be developed to efficiently collect user feedback and translate it into meaningful improvements for the current system?" - guided this investigation. In response, three supporting sub-questions were formulated and methodically explored.

The first sub-question shed light on the current feedback handling method in the existing system, revealing the necessity for a more streamlined feedback mechanism. Subsequently, the second sub-question explored potential enhancements that could optimize OpusFlow's current feedback handling process, emphasizing the anticipated benefits of incorporating a user-friendly feedback module. The third sub-question emphasized the need for direct user feedback mechanisms, integration with issue tracking tools, and targeted feedback strategies for the minimum viable product.

In conclusion, the collective findings from these sub-questions provide a comprehensive roadmap for the project's development. These insights not only lay a robust foundation for the Implementation and Testing stages but also set a precedent for future projects aimed at optimizing user feedback processes in software development. By focusing on user needs, operational challenges, and industry best practices, the project is poised to deliver a solution that significantly enhances the user feedback handling process, thereby improving software quality and user satisfaction. This research phase has thus successfully achieved its objective, providing valuable insights that directly inform the solution to the primary research question.

4. Implementation

4.1 Introduction

This section delves into the implementation phase of the project, transforming the conceptual designs, informed by rigorous research, into a functioning system. This stage serves to actualize the research objectives outlined previously, focusing on the practical application of the identified requirements for a more streamlined feedback mechanism and system architecture.

The implementation stage marks a critical juncture in the research project. Here, theoretical insights gleaned from the analysis and design phases are put into practice. It provides an opportunity to validate the research findings, ensuring the devised solution effectively addresses the identified issues and meets the user needs for a user-friendly feedback module and direct user feedback mechanisms.

Within the context of this research, the objectives of the implementation stage include developing a user-friendly feedback module that integrates seamlessly with the existing OpusFlow system. It aims to improve the feedback handling process, enhance system functionality and usability, and facilitate integration with issue tracking tools. This stage will also yield valuable insights that could guide future projects and initiatives, particularly those aimed at optimizing user feedback processes in software development.

4.2 Approach

The chosen methodology for this stage of the project incorporates a combination of literature review, best practices study, prototyping, and the use of specific programming languages, frameworks, and cloud services. Each method was selected based on its alignment with the project objectives, its potential to address the identified needs, and its capacity to contribute to the successful development of the current project.

1. **Literature Review and Best Practices Study:** These methods enabled the project to draw upon existing knowledge and industry standards to guide the design and functionality of the feedback module. By studying successful and unsuccessful practices in existing products, the project can avoid common pitfalls and emulate successful strategies, thereby improving the anticipated user experience and functionality of the feedback module.
2. **Prototyping:** Prototyping was used to visually represent the system's aesthetics and functionality, aiding in requirement elicitation. This method is beneficial as it allows stakeholders to visualize the proposed solution, provide feedback, and ensure it meets their needs before full-scale development begins. This iterative process aims to enhance user satisfaction and minimize the need for extensive changes during the development stage.
3. **Technical Implementation:** The development of the project was carried out using specific programming languages, frameworks, and cloud services. The chosen technologies align with the existing technical stack of the OpusFlow software, ensuring seamless integration and reducing potential compatibility issues. By conducting continuous testing during development, the project can ensure the product's functionality aligns with the identified needs, thereby increasing its potential to deliver the anticipated benefits.

The selected methods and strategies aim to ensure that the developed solution aligns with the existing OpusFlow ERP system, addresses the identified needs, and delivers the anticipated benefits.

By adhering to the same technical stack, the integration process is anticipated to be efficient, thereby contributing to the project's overall success.

4.3 Design

In the design phase, the conceptualized OpusFlow feedback module takes shape through meticulous planning of the system and architecture. This section navigates through the key elements of system design and architectural considerations, providing a roadmap for transforming theoretical insights into a tangible and functional solution. The design phase plays a pivotal role in aligning the envisioned solution with the research objectives, focusing on creating a user-friendly feedback module and refining the system architecture for enhanced functionality and integration capabilities.

4.3.1 Architecture Design

This section serves as a comprehensive guide, detailing how the envisioned system takes shape, ensuring a cohesive and efficient structure that aligns with the identified research objectives and user needs.

Overview

The research phase of this project identified a pressing need for an efficacious mechanism that would allow customers to provide feedback on the features of the OpusFlow ERP system. Also, there is a clear need for a dashboard that can visually show the collected feedback and simplify task creation in the Linear Application.

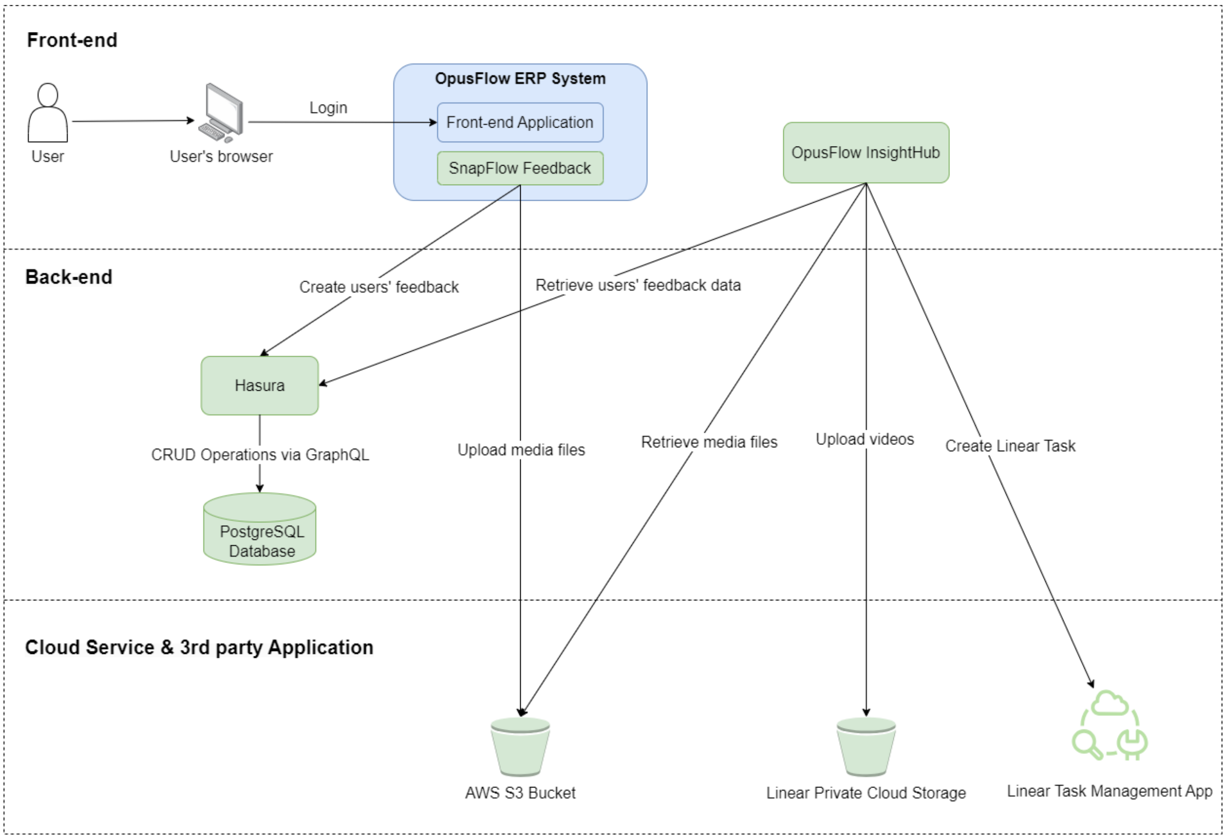


Figure 14. System Architecture of the Project

The system architecture exhibits the interaction of several components. The **blue** components belong to **the existing OpusFlow system**, while **green** components **while the green components will be developed or integrated** within the project**.** Here is the breakdowns:

1. **Front-end:**
   - **SnapFlow Feedback:** Integrated with OpusFlow, it streamlines user feedback, enabling users to express experiences and issues, including context through screen recordings or captures.
   - **OpusFlow InsightHub:** An autonomous web application providing a consolidated dashboard for viewing, filtering, and sorting user feedback data within OpusFlow. It integrates with Hasura for data management and Linear for task creation.
2. **Back-end:**
   - **PostgreSQL Database:** Central to the project, it stores various data types from user feedback. Accessible and manageable via the Hasura console.
   - **Hasura:** This open-source engine plays a key role in our architecture by serving as the primary interface with the PostgreSQL database. Hasura auto-generates a GraphQL backend, providing a flexible, efficient API for our front-end components. It offers an interface for managing the data and allows querying the data using GraphQL, making it an essential tool for managing and accessing feedback data.
3. **Cloud Service:**
   - **AWS S3 Bucket:** The Amazon Simple Storage Service (S3) bucket is utilized to store media files, including screenshots and screen recordings associated with user feedback, uploaded during the feedback process.
   - **Linear Private Storage:** A secure storage solution that ensures the privacy and safety of sensitive task-related data. It seamlessly integrates with the Linear application, allowing for smooth data exchange and task management based on user feedback.
4. **Third-Party Application:**
   - **Linear:** Linear is a task management application integrated with the OpusFlow InsightHub. It streamlines the process of addressing user feedback and supports task creation through its GraphQL API.

Justification

The architecture design of the project was carefully crafted, taking into account various crucial considerations, particularly the user stories identified during the research phase (marked with IDs starting with 'U'):

1. **SnapFlow Feedback Integration:** SnapFlow Feedback, an integral component in the front-end, has been seamlessly integrated into the existing OpusFlow system. This integration allows users to provide feedback efficiently, expressing experiences and issues, including context through screen recordings or captures. The decision to tightly integrate it ensures real-time data exchange, addressing the user's need for a straightforward and effective feedback process (U1).

2. **OpusFlow InsightHub Separation:** OpusFlow InsightHub, designed as a standalone web application, offers flexibility in development and maintenance. By keeping it separate from the OpusFlow system, updates or modifications to the dashboard won't impact the ERP

system, ensuring a smooth user experience. The decision to develop it as a separate entity aligns with the specific needs of the consultancy team, providing actionable insights (U3) and a centralized dashboard (U5) for reviewing all collected feedback.

3. **PostgreSQL and Hasura Utilization:** The selection of PostgreSQL as the central database and Hasura as the open-source engine is grounded in their proven effectiveness in the OpusFlow system. Consistently using the same technology stack not only makes maintenance easier but also supports ongoing development efforts. In this project, a separate instance of the database and Hasura technology mirrors the ones used in OpusFlow for user data management, operating independently to guarantee data integration integrity. This choice is strengthened by its built-in scalability, efficiently handling an increased volume of user feedback. PostgreSQL's ability to handle heightened workloads is noteworthy, owing to its robust architecture, optimized query processing, and efficient support for indexing. Paired with Hasura, which generates a flexible GraphQL backend, they facilitate smooth communication between the front-end and the database. Together, PostgreSQL and Hasura form a scalable foundation capable of adapting gracefully to the evolving demands of the system.

4. **AWS S3 Bucket Integration:** The incorporation of AWS S3 Bucket as a secure and scalable storage solution for media files in the feedback process is a strategic choice. Aligning with OpusFlow's technology stack ensures consistency, simplifying the task for subsequent developers familiar with the technology. This integration not only ensures reliability but also contributes significantly to scalability, empowering the storage solution to adeptly manage the burgeoning volume of media files. AWS S3's design, tailored for scalability, manifests in industry-leading storage capabilities that seamlessly scale with escalating data volumes. Its object storage architecture facilitates limitless storage expansion, ensuring the system's efficient handling of the growing influx of media files linked to user feedback. Leveraging AWS S3, the project gains a resilient and scalable storage solution, adeptly aligning with the envisaged surge in media file uploads without compromising on performance or security.

5. **Linear Application Integration:** Linear, a third-party task management application, is seamlessly integrated with OpusFlow InsightHub, streamlining the process of addressing user feedback. This integration facilitates task creation via the GraphQL API, allowing the consultancy team to initiate tasks based on user feedback (U4). Linear Private Storage complements this by securely handling sensitive task-related data, ensuring smooth data exchange and efficient task management.

4.3.2 Database Design

Introduction

Within the architecture of the SnapFlow Feedback and OpusFlow InsightHub, the database holds a crucial role as the central storage for user feedback and associated data. This section delves into the pivotal role and significance of the database in the project.

Entities & Attributes

The database comprises five tables: Feedback, Task, and three Enum tables (Feedback Issue, Feedback Type, Feedback Element). To provide a clearer picture of the database structure and the relationships between tables, a diagram is presented below:

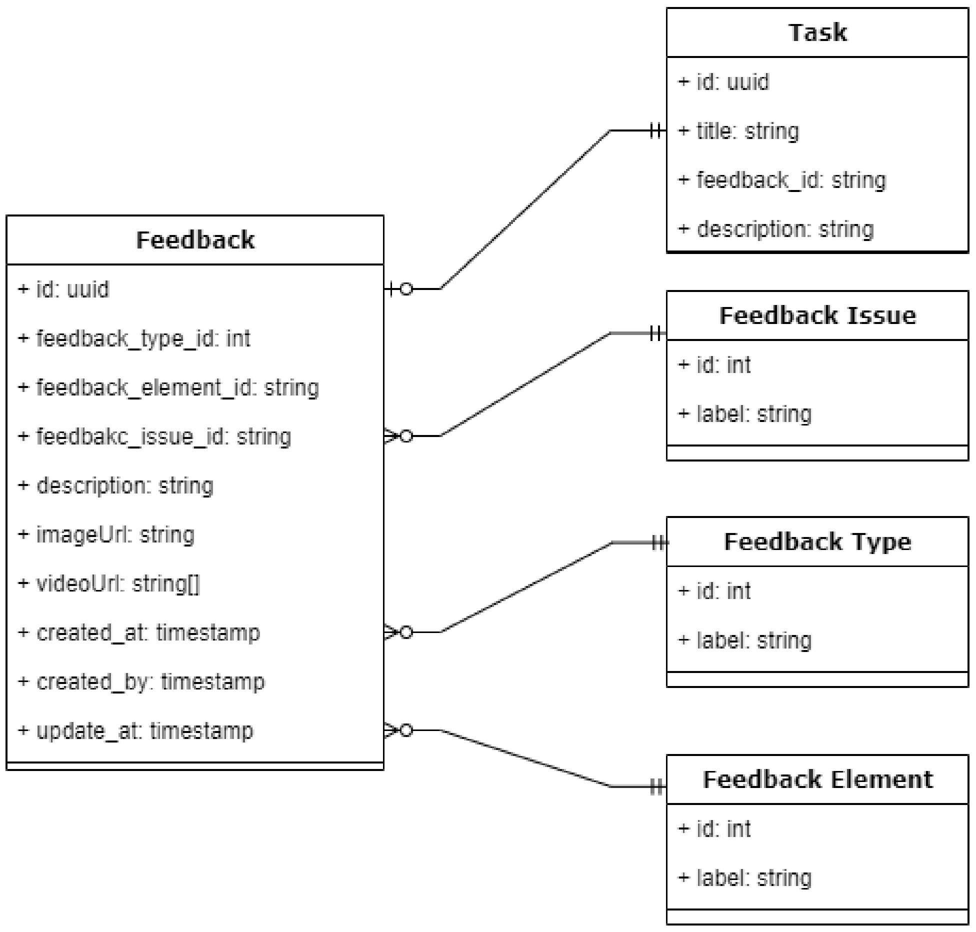


Figure 15. Project Database Diagram

The Feedback table is the central entity in the design, capturing comprehensive user feedback data. The Feedback table includes the following fields:

- id (UUID): A unique identifier for each feedback entry.
- feedback_type_id (Int): The feedback type, referencing the Feedback Type Enum table through a foreign key relationship.
- feedback_element_id (Int): The feedback element, referencing the Feedback Element Enum table via a foreign key association.

- feedback_issue_id (Int): The feedback issue, referencing the Feedback Issue Enum table through a foreign key connection.
- description (String): A detailed description of the feedback.
- imageURL (String): The URL of the related image, stored on AWS.
- videoURL (String Array): An array of URLs of related videos, stored on AWS.
- created_at, created_by, update_at (Timestamp): Time-related fields tracking the creation and updating of the feedback.

The Task table has a one-to-zero-or-one relationship with the Feedback table, linking each task to a specific feedback. It includes these fields:

- id (UUID): A unique identifier for each task.
- feedback_id (String): The ID of the related feedback.
- description (String): A detailed description of the task.

The three Enum tables, namely Feedback Issue, Feedback Type, and Feedback Element, establish a one-to-many relationship with the Feedback table, enabling the association of multiple feedback instances with each entry. These tables play a crucial role in categorizing feedback and contribute to the effective filtering and analysis of feedback data within the OpusFlow InsightHub web application.

These Enum tables share a uniform structure, characterized by two fields:

- id (Int): An unique identifier for each entry.
- label (String): A string indicating the specific category or classification associated with each Enum table.

Rationale & Functionality Support

The database design for the SnapFlow Feedback and OpusFlow InsightHub projects was methodically shaped to align with specific project requirements. A pivotal need of the SnapFlow Feedback component was the precise capture and categorization of diverse user feedback forms. The Feedback table, characterized by its comprehensive fields and associations with the three Enum tables, serves as a direct support mechanism for fulfilling this requirement. Through the systematic categorization of feedback by type, element, and issue, the system can efficiently filter and analyze feedback, contributing to an insightful comprehension of user experiences.

The design of the Task table aimed at facilitating effective task management based on the received user feedback. Through a one-to-one relationship, each task is directly linked to a specific feedback entry, ensuring a clear traceability of issues from identification to resolution. This design decision simplifies task tracking, enhancing the efficiency of the consultancy team.

In summary, the database design establishes a robust foundation for the functionalities of SnapFlow Feedback and OpusFlow InsightHub. Its structured relationships are tailored to support efficient data capture, categorization, task management, and in-depth data analysis, all of which are critical for the successful operation and continuous evolution of the system.

4.3.3 Wireframe Design

The success of a software system depends on how well it meets what users want and expect. Wireframe design is essential as it provides a visual representation of the functional needs and serves as a guide for both design and functionality. This part of the discussion focuses on wireframe design, specifically looking at two important areas: SnapFlow Feedback and OpusFlow InsightHub..

4.3.3.1 SnapFlow Feedback

As a crucial part of the project's development, SnapFlow Feedback is essential for users to give their opinions within the OpusFlow system. The wireframes show how SnapFlow Feedback will work, illustrating the functional needs and providing a visual guide for design and functionality. Following this, each wireframe will be analyzed, establishing their connections to the specified requirements.

1. Wireframes for requirement F1 – Submission of Feedback

These wireframes depict a popover form, accessible via a bottom-right icon resembling a message. The form includes a stepper component at the top, indicating the current step and the total steps to completion. A breakdown of each step is outlined below:

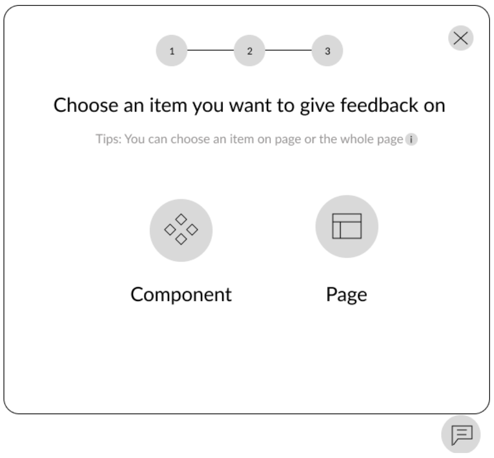

1
2
3
Choose an item you want to give feedback on
Tips: You can choose an item on page or the whole page
Component
Page

Figure 16. Initiating Feeback - Element Selection

Step 1: Users select the specific element on the page for feedback, choosing between a component or an entire page.

The wireframe incorporates a "tips" section for user guidance and two buttons representing each type of selectable element.

Upon selecting either a component or a page, the capture phase initiates, leading the user to Step 2.

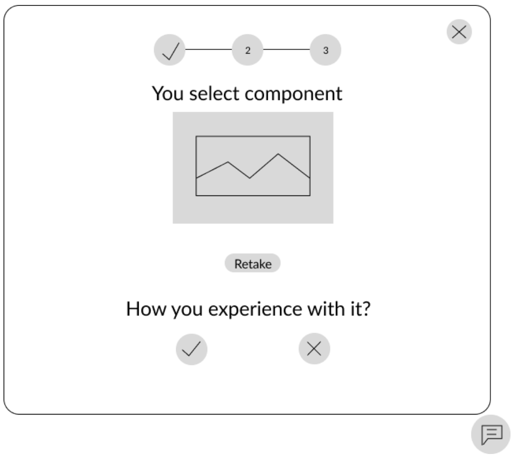


Figure 17. Capturing and Providing Initial Feedback

Step 2: After selection, users proceed to this step, where the form reminds them of their choice, displaying the captured selection.

Options for retaking the capture, a text field prompting users for feedback, and buttons to advance to the next step are included.

Following this, users proceed to Step 3.

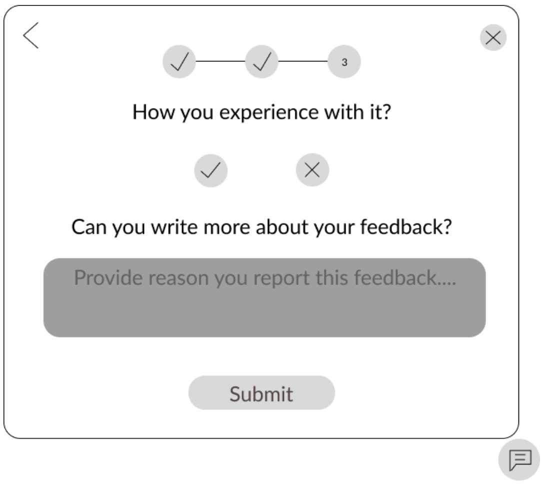

3
How you experience with it?
Can you write more about your feedback?
Provide reason you report this feedback....
Submit

Figure 18. Positive Feedback Submission

Step 3 (Positive Feedback Scenario): For users giving positive feedback, this wireframe show the chosen selection and allocates a section for users to explain what their affirmative experiences.

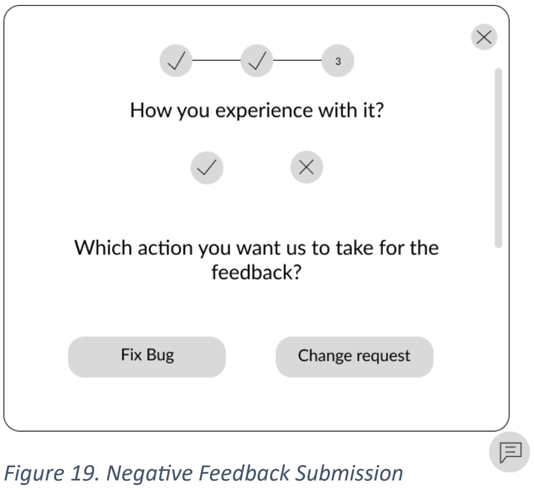

3
How you experience with it?
Which action you want us to take for the feedback?
Fix Bug
Change request

Figure 19. Negative Feedback Submission

Step 3 (Negative Feedback Scenario): In instances where users are dissatisfied and choose to provide negative feedback, two categories are anticipated: reporting a bug or requesting a change in a feature causing dissatisfaction.

Users are prompted to select one of two buttons corresponding to the mentioned cases. Subsequently, users will need to scroll down to screen recording and additional information section in the wireframe below.

2. Wireframe for requirement F2 – Capturing Context of Feedback

This part focuses on the need for users to provide additional details about their feedback, particularly when it comes to screen recordings.

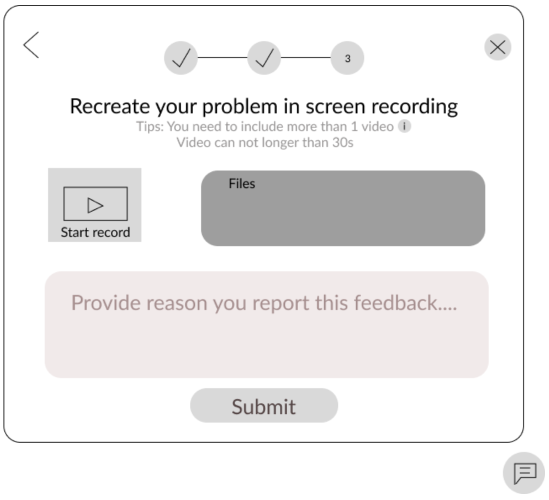

3
Recreate your problem in screen recording
Tips: You need to include more than 1 video
Video can not longer than 30s
Start record
Files
Provide reason you report this feedback....
Submit

Figure 20. Capturing Context for Feedback with Screen Recording

Step 3 (Negative Feedback Scenario): Once users choose between two clear categories—reporting a bug or requesting a modification in a feature—they move on to the next step. Here, they scroll down to a specialized wireframe designed for screen recording and additional information.

Within this section, users can effectively recreate the problem while simultaneously recording their screen. Clear instructions for recording are available, and users can provide any extra details in a text box provided for additional information.

Rationale

This section analyzes the reasoning behind specific design choices for SnapFlow Feedback, offering insights into the decision-making process for each wireframe. The objective is to demonstrate how the design aligns with the project's requirements.

1. Layout Decision: Popover Form for SnapFlow Feedback

In the design of SnapFlow Feedback, a strategic decision was made to implement it as a popover form, activated by user-initiated actions.

The rationale behind adopting a popover form lies in its capacity to offer a non-intrusive and easily accessible avenue for users to convey their feedback. Research conducted by Kanika (n.d.) on diverse methods of website feedback collection identifies two prevalent methods for website feedback collection, both of which align seamlessly with the context at hand:

- Website popup survey: This survey appears within a popup on the website, immediately capturing visitors' attention, thereby increasing survey participation rates.
- Website feedback popover form: Comprising a button that visitors can click to open a survey overlaying the screen, this method ensures a user-friendly approach.

Additionally, the consideration of micro surveys, short and efficient tools for collecting feedback on websites, further underscores the project's commitment to enhancing response rates. Given the intricacies of the database design for feedback, the first two options are deemed more suitable. Moreover, research supports the adoption of the feedback popover form, aligning seamlessly with a preference for unobtrusive approaches.

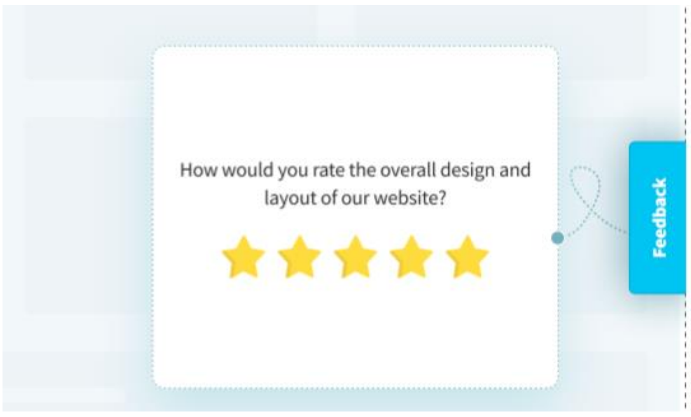

How would you rate the overall design and layout of our website?
Feedback

Figure 21. Example of Website Feedback Popover Form

The design choice is also influenced by well-established applications like AccuWeather and Notion. Inspired by AccuWeather's web application, a strategically positioned "Feedback" button is incorporated on the right border of the page. Additionally, a secondary design consideration draws from Notion's Floating Action Button (FAB), ensuring easy access to the feedback button from any page location.

In contrast to the alternative, website popup survey like the illustration example below, which may disrupt the user experience by appearing abruptly.

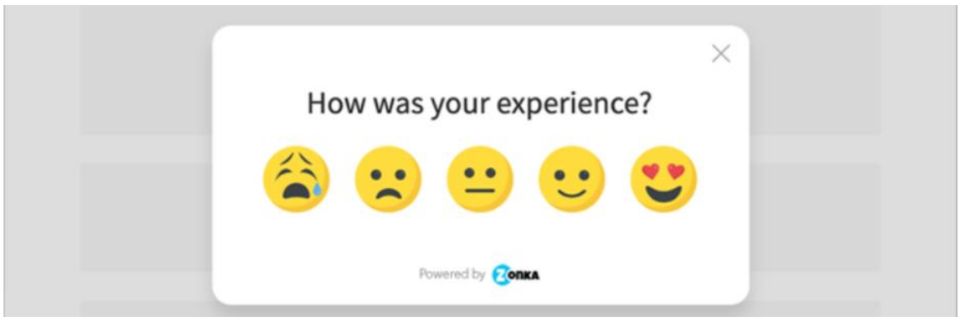


Figure 22. Example of Website Popup Survey

The popover form strikes a balance by avoiding interruption and ensuring an uncluttered interface, aligning with the project's goals of providing a feedback mechanism that is both unobtrusive and readily available. This decision respects the user's autonomy, contributing to an enhanced overall user experience by seamlessly integrating feedback submission into the user's workflow. Therefore, employing a popover form as the layout for SnapFlow Feedback is a deliberate and well-considered step toward achieving the project's user-centric goals.

To enhance user experience during negative feedback submission, a visual stepper has been incorporated on the top of the popover form. Serving as a visual roadmap, the stepper guides users through the process, indicating their current position and facilitating an intuitive and transparent submission process.

2. Form Structural Overview and Steps with Reasoning

The structural framework of SnapFlow Feedback revolves around a three-step process, carefully designed to streamline user interaction. This three-step approach is in line with the specified requirements for structuring user feedback, ensuring a systematic and user-friendly process.

The decision to divide the feedback submission into three steps serves the purpose of providing users with a clear and guided path, enhancing comprehension and engagement. Following is the detail of each step:

- Step 1: Activation and Element Selection

  The first step allows users to activate SnapFlow Feedback and select a specific element on the page for feedback. This choice was made to offer users a straightforward initiation process, aligning with their expectations.

- Step 2: Confirmation and Additional Input

  Building on the user's selection, the second step involves confirming the choice and providing additional input. This ensures a structured approach, allowing users to express their thoughts clearly.

- Step 3: Positive and Negative Feedback Scenarios

The third step caters to both positive and negative feedback scenarios, accommodating a diverse range of user interactions. This flexibility aligns with the varied nature of user experiences.

By breaking down the process into these three steps, SnapFlow Feedback wireframe design aims to create a smooth and intuitive user experience, facilitating effective communication and feedback submission. This integrated approach allows for a seamless transition from the overall structural overview to the detailed reasoning behind each step, maintaining clarity and cohesion in the explanation.

Furthermore, the granularity of detail within each step is purposefully aligned with the incorporation of specific fields—type, element, issue, description, imageURL, and videoURL. This alignment directly correlates with the underlying database design for feedback, enhancing clarity and mitigating ambiguity in the overall task creation process.

4.3.3.2 OpusFlow InsightHub

As a pivotal front-end component in the project's development phase, OpusFlow InsightHub assumes a crucial role. It serves as a key interface for displaying user feedback and plays an integral part in generating tasks in the Linear application for the consultancy team.

Wireframes

The presented wireframes encapsulate the functional requirements stipulated OpusFlow InsightHub , acting as both a visual representation of design aesthetics and a functional guide aligned with specified requirements. An exploration of each wireframe, along with connections to associated functional requirements, follows:

1. Wireframe for requirement F5 – Feedback Correlation View

The Feedback Correlation View is designed to meet the requirement of visualizing the association or link between different pieces of user feedback (F5). The wireframe includes:

- An intuitive display showcasing various pieces of user feedback, presenting essential information such as the number of feedback received from customers, the count of change requests, bug reports, and issues created on Linear based on the received feedback
- Visual cues, including chart types such as bar charts or doughnut charts, to identify patterns and areas of concern. This representation summarizes the current state of feedback, distinguishing between positive and negative feedback, and categorizing the feedback received based on page or component type.

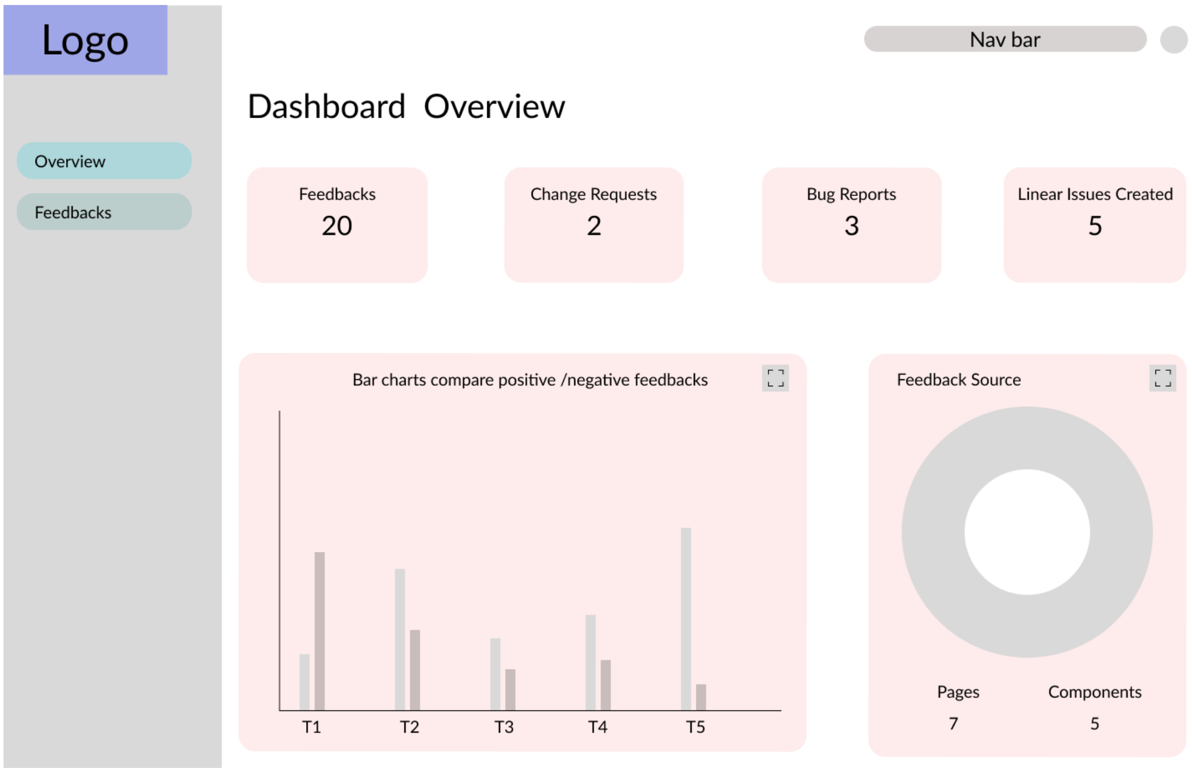
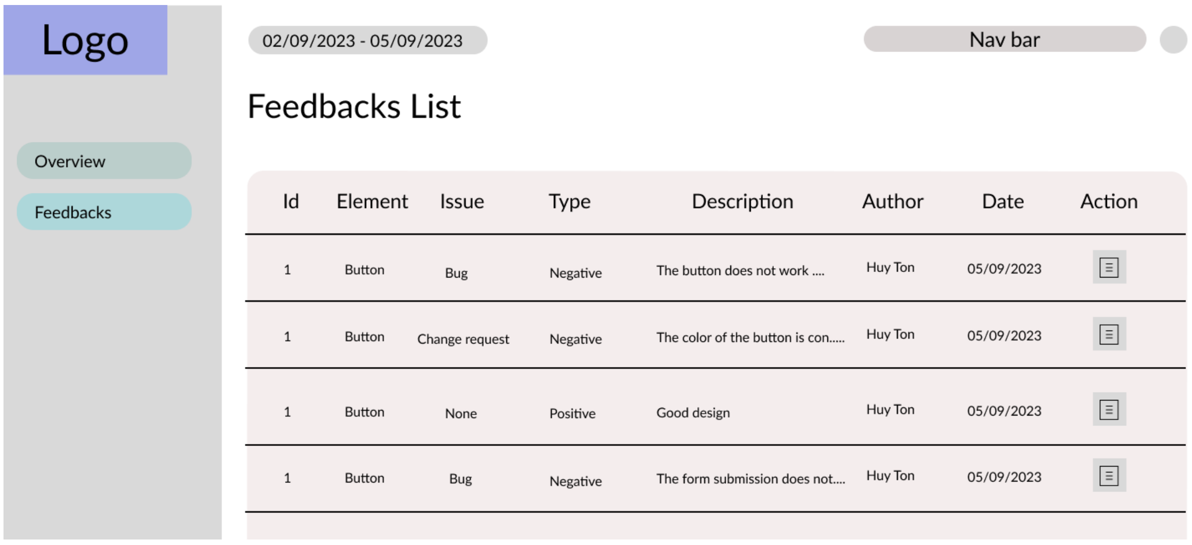


Figure 23. Comprehensive Overview Dashboard

2. Wireframe for requirement F4 – Centralized Feedback Dashboard

Addressing the need for a Centralized Feedback Dashboard (F5), this wireframe introduces a consolidated platform in list view for viewing, filtering, and sorting feedback. Key components include:

- Accessibility to all feedback items in a centralized location, presented in a listed view with columns such as element, issue, description, creator, and creation date. Action columns provide buttons to interact with feedback, such as creating tasks or viewing details.
- User-friendly features enabling filtering and sorting based on various parameters such as element, issue, and type.
- The "create task" option applies only to negative feedback, as it necessitates actions within the OpusFlow system to rectify the identified problems.



Feedbacks List

Id	Element	Issue	Type	Description	Author	Date	Action
1	Button	Bug	Negative	The button does not work	Huy Ton	05/09/2023	☰
1	Button	Change request	Negative	The color of the button is con.....	Huy Ton	05/09/2023	☰
1	Button	None	Positive	Good design	Huy Ton	05/09/2023	☰
1	Button	Bug	Negative	The form submission does not....	Huy Ton	05/09/2023	☰

Figure 24. Feedback Listing View

3. Wireframe for requirement F3 – Task Initiation in Linear App

To meet the requirement of Task Initiation in the Linear App (F3), the wireframe outlines the process of generating tasks using information obtained from feedback. Key elements include:

- Availability of a pre-filled form based on feedback data for seamless task creation.
- Two types of forms, one for Request for Change (RFC) and another for Bug reports, are automatically selected based on the issue identified in the feedback.

For a comprehensive view of RFC task templates, refer to Appendix C. Comprehensive wireframes for the entire bug report can be accessed in Appendix D.

Rationale

This section provides a comprehensive understanding of the design decisions made for OpusFlow InsightHub, considering layout, components, list view, and template fields, along with alternative considerations at each section.

1. Layout Design and Navigation:

Opting from the system architecture, OpusFlow InsightHub, a stand-alone web application, was intentionally designed with a layout featuring a left-sided vertical navigation bar. This design was driven by the goal of enhancing user experience and navigation efficiency. A standalone app allows for a focused and dedicated interface, minimizing distractions. The vertical navigation bar offers clear and intuitive access points, with distinct links to the overview and feedback pages. This layout choice ensures ease of navigation, catering to user expectations and contributing to an uncluttered, user-centric design. Despite considering a top horizontal navigation bar, the vertical option prevails for its scalability and user-friendly attributes, particularly in scenarios involving the app's expansion with additional features.

2. Overview Components:

The components within the overview section play a pivotal role in meeting the requirement of visualizing associations between different user feedback pieces. Essential information, strategically chosen, includes the number of received feedback, change requests, bug reports, and issues created on Linear. Visual cues, implemented through chart types like bar charts and doughnut charts, offer an immediate snapshot of the feedback landscape, allowing users to quickly identify patterns and areas of concern. While an alternative design considered a more minimalistic overview with only essential metrics, it was dismissed in favour of a detailed approach, providing a nuanced understanding of user feedback.

3. List View and User-Friendly Features:

The list view for the Centralized Feedback Dashboard is selected to fulfill the requirement of a consolidated platform for viewing, filtering, and sorting feedback. This format ensures easy accessibility to all feedback items, with columns providing relevant details such as element, issue, description, creator, and creation date. Action buttons enable efficient user interaction, supporting tasks like creating tasks or viewing details. User-friendly features for filtering and sorting based on various parameters enhance the platform's usability, aligning seamlessly with the goal of delivering a centralized and efficient feedback management system. An alternative card-based layout was considered but discarded due to its potential clutter and reduced efficiency in handling large volumes of feedback items.

4. Template Fields and Relevance:

The fields within the Bug Report and RFC task templates are strictly chosen to align precisely with the requirements outlined by the QA Manager. These fields, encompassing element, issue, description, and additional action buttons, are deemed indispensable for seamless task creation based on user feedback. Each field serves a specific purpose in facilitating the task initiation process, ensuring that relevant information is captured for efficient issue resolution. Detailed field references can be found in Appendix E for RFC template and Appendix F for bug fix template. All fields marked with an asterisk (*) are mandatory in the task creation form.

4.4 Development

In the development phase, the detailed plans outlined during the design stage transition into practical and function applications. This section provides a thorough exploration of the process involved in transforming architectural design, database structures, and wireframes designs into fully functional components.

4.4.1 Framework & Technology Stack Overview

Next.js serves as the framework of choice for this project, functioning as a React framework to facilitate the construction of full-stack web applications. Using React Components for UI development and Next.js for additional features and optimizations, the architecture of the project is intentionally crafted to be serverless, removing the conventional separation between front-end and back-end.

The entire project, comprising user interface components and serverless functions, is encapsulated within the Next.js framework. This unified approach seamlessly manages both the user interface and server-side operations within a single project. Functions within the '/api/' folder serve as dedicated API endpoints, forming server-side-only bundles that remove the need for a conventional server. This enables the consolidation of code in a single project without the necessity of separating it into distinct front-end and back-end codebases. This development approach is chosen for its alignment with the existing OpusFlow framework, promoting efficient and scalable development practices.

Combined with Next.js, the technology stack used in the development of the project includes a PostgreSQL database with a GraphQL API, AWS S3 for cloud storage services, and integration with a third-party application, Linear. The following diagram illustrates how these components collaborate within the system.

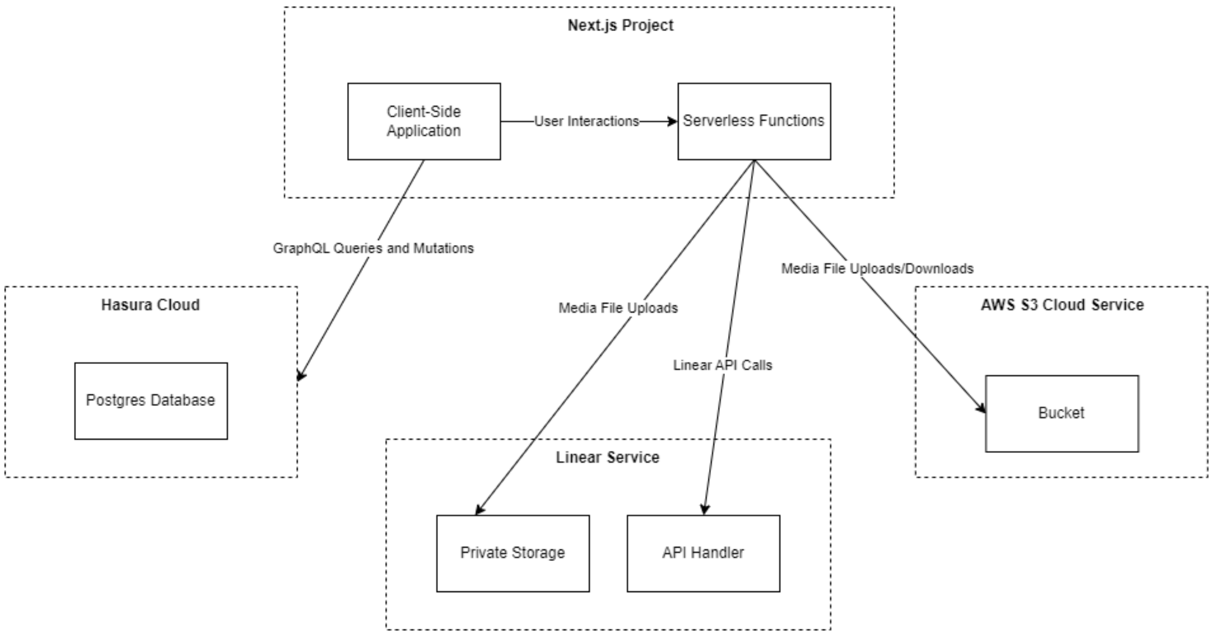


Figure 25. High-Level System Architecture Overview

In terms of functionality, Next.js is structured around two key components the client and the server side.

- **Client Side:** The client side in Next.js refers to the part of the application that runs on the user's device. Its primary role is to render and display content, ensuring a seamless and interactive user experience. Notably, user interactions can trigger serverless functions, adding dynamism to the application. Furthermore, the client side actively participates in GraphQL API calls to Hasura Cloud, facilitating queries or mutations on the PostgreSQL database.

- **Server Side:** The server side manages backend operations, employing serverless functions to streamline endpoints for handling API calls and executing specific tasks. It initiates API calls to AWS S3 for media file uploads or downloads and creates tasks through GraphQL integration with Linear.

Establishing connections with the database, AWS S3 bucket, and Linear involves incorporating specific keys into the environmental file. In essence, these keys can be likened to exclusive codes that unlock access to various services within the application. The environment file as a secure vault, safeguarding these keys and ensuring that only authorized users, specifically the application, can access and engage with particular functionalities.

This foundational setup, defined by unique environment keys, forms the basis for secure, authenticated, and efficient interactions with PostgreSQL, GraphQL, AWS S3 bucket, and Linear within the application. Each key serves a distinct purpose, enabling the application to communicate securely and seamlessly with the integrated services. Here's a breakdown of the setup and the meaning behind each key:

- PostgreSQL and GraphQL Integration: Two critical keys are essential for integration
  - HASURA ADMIN SECRET: This key ensures administrative access to Hasura, allowing the application to perform essential actions such as creating, updating, or modifying the database.
  - HASURA URL: This key specifies the URL endpoint for the Hasura GraphQL API, providing the application with the necessary address to interact with the database.
- AWS S3 Bucket Integration: For seamless integration, two keys are required:
  - Bucket Key: Essential for authenticating and accessing the designated AWS S3 bucket.
  - Bucket Secret: Serving as a confidential counterpart to the bucket key, the bucket secret ensures secure communication and access to AWS S3 resources.
- Linear Integration:
  - The Personal API key serves as a secure gateway for the application. This key acts as a secure token, establishing a trustful and authenticated connection between the application and Linear's GraphQL API.

4.4.2 Libraries Used In Development

Following is the list of the primary libraries utilized in the project for development:

1. **ReactJs:**
    - **Role:** React is a JavaScript library for building user interfaces. It enables the creation of reusable UI components that efficiently update and render based on the application's state.
    - **Significance:** React simplifies the development of interactive and dynamic user interfaces, providing a declarative approach to designing UI components.
2. **TypeScript:**
    - **Role:** TypeScript is a superset of JavaScript that adds static typing to the language. It helps catch errors during development and improves code maintainability.
    - **Significance:** TypeScript enhances code quality by providing a structured and typed approach to JavaScript, reducing potential runtime errors.
3. **@apollo/client:**
    - **Role:** Apollo Client is a state management library for managing data in a React application. It is commonly used for fetching and caching data from a GraphQL API.
    - **Significance:** Apollo Client simplifies the integration of GraphQL with React applications, providing a scalable and efficient way to manage data.
4. **MUI material and MUI icons material:**
    - **Role:** Material-UI is a React UI framework that provides pre-designed components following the Material Design principles.
    - **Significance:** MUI makes it easy to create visually appealing and consistent UI components, enhancing the overall design and user experience.
5. **React Hook Form:**
    - **Role:** React Hook Form is a library for managing forms in React applications. It uses React hooks to simplify form state management.
    - **Significance:** React Hook Form streamlines the process of handling forms in React, making form development more efficient and maintainable.
6. **Yet Another React Lightbox:**
    - **Role:** A lightbox library for displaying images or other media in a modal-like overlay.
    - **Significance:** Enhances user experience by providing an elegant way to showcase images or media content within the application.
7. **Yup:**

- **Role:** Yup is a JavaScript schema validation library.
- **Significance:** Yup is used for validating form data, ensuring that input follows the specified schema, preventing incorrect or malicious data submission.

8. **@aws-sdk/client-s3 & @aws-sdk/s3-request-presigner:**
    - **Role:** AWS SDK for JavaScript provides a client for Amazon S3 (Simple Storage Service).
    - **Significance:** Enables seamless integration with AWS S3 for cloud storage, allowing the application to upload and download media files.
9. **Next:**
    - **Role:** Next.js is a React framework for building server-side-rendered and statically generated web applications.
    - **Significance:** Next.js simplifies the development of React applications by providing features like server-side rendering and automatic code splitting.
10. **@linear/sdk:**
    - **Role:** The Linear SDK is a library for interacting with the Linear app API.
    - **Significance:** Facilitates integration with the Linear app, allowing the application to create and manage tasks seamlessly through GraphQL.
11. **node-fetch:**
    - **Role:** A lightweight module that brings window.fetch to Node.js environments.
    - **Significance:** Facilitates making HTTP requests within the Node.js environment, enabling communication with external APIs.
12. **axios:**
    - **Role:** A promise-based HTTP client for the browser and Node.js.
    - **Significance:** Simplifies making HTTP requests, handling responses, and managing request and response transformations.
13. **Html2canvas:**
    - **Role:** Html2canvas is a JavaScript library that renders HTML to canvas.
    - **Significance:** Useful for capturing and converting HTML elements, such as images or charts, into a canvas for further use or export.

For comprehensive details about each library, refer to the reference section of the report, where links to the documentation of each library are provided.

4.4.3 Database & GraphQL

PostgreSQL setup with GraphQL and Hasura Cloud

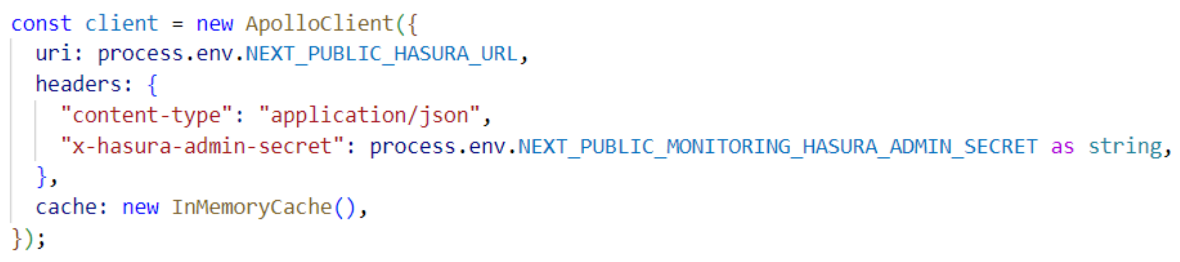
In this project, the PostgreSQL database is implemented through Hasura Cloud, a platform designed to streamline database management using GraphQL APIs. Hasura Cloud not only simplifies database connections but also provides an intuitive console interface for effective data handling. The setup process was guided by the company supervisor, and to provide more details on the specific configuration:

- Hasura Cloud:
    - Explanation: Hasura Cloud is a cloud-based service that facilitates the configuration of PostgreSQL databases through GraphQL APIs.
    - Setup Process: The setup process within Hasura Cloud involves utilizing a graphical console interface, allowing users to manage and configure their databases seamlessly.
- Neon Cloud Server:
    - Explanation: The PostgreSQL database is hosted on the Neon Cloud server, a cloud infrastructure chosen based on the guidance of our company supervisor.
    - Selection Rationale: The decision to use Neon Cloud aligns with OpusFlow's existing technology stack, ensuring compatibility and making it easy to control and maintain the database.
- Configuration Process:
    - Explanation: Configuring PostgreSQL involves obtaining a specific database URL from Neon DB and integrating it into the Hasura console.
    - Steps:
        - Acquiring Database URL: Obtain the database URL from Neon DB, serving as the unique address for the PostgreSQL database.
        - Integration into Hasura Console: The obtained URL is integrated into the Hasura console, finalizing the PostgreSQL setup seamlessly.

Application connection

To establish the connection between the database and the client-side of the application after adding the necessary keys to the project's environment file, two distinct approaches were employed based on specific conditions.

For the SnapFlow Feedback component, integrated into OpusFlow's with its own GraphQL-configured database, an Apollo Client variable was created to facilitate mutations or queries.

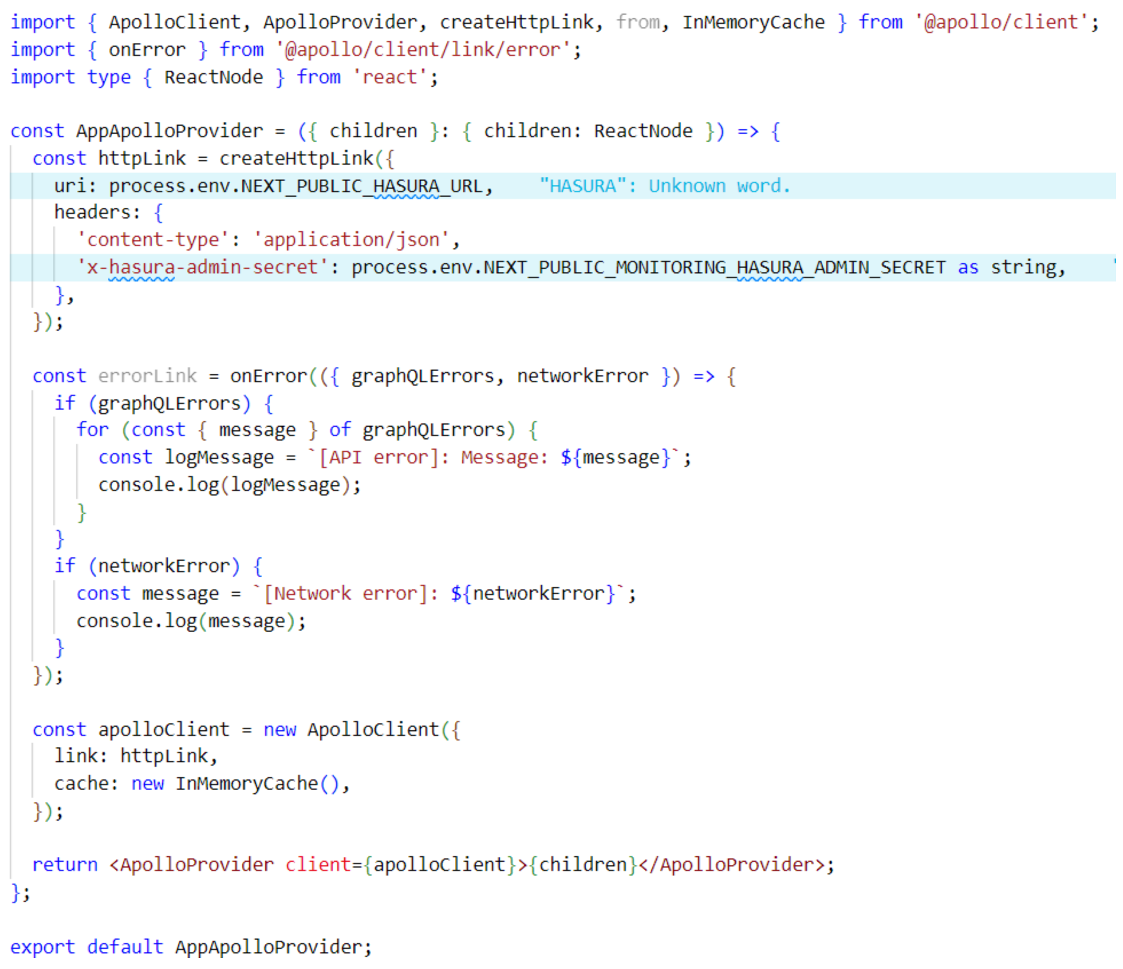


```
const client = new ApolloClient({
  uri: process.env.NEXT_PUBLIC_HASURA_URL,
  headers: {
    "content-type": "application/json",
    "x-hasura-admin-secret": process.env.NEXT_PUBLIC_MONITORING_HASURA_ADMIN_SECRET as string,
  },
  cache: new InMemoryCache(),
});
```

Figure 26. Code snippet showcasing creating a Apollo Client

This code initializes an Apollo Client, a crucial part of GraphQL communication. The uri specifies the endpoint where the Hasura GraphQL API is hosted, allowing the component to interact with the connected database. Additionally, it sets up headers, including the content type and an admin secret key for secure communication. The cache parameter involves the management of data storage and retrieval, enhancing the efficiency of GraphQL interactions.

As for OpusFlow InsightHub, a standalone web application outside the OpusFlow system, the integration relies on encapsulating the entire app with AppApolloProvider.



```
import { ApolloClient, ApolloProvider, createHttpLink, from, InMemoryCache } from '@apollo/client';
import { onError } from '@apollo/client/link/error';
import type { ReactNode } from 'react';

const AppApolloProvider = ({ children }: { children: ReactNode }) => {
  const httpLink = createHttpLink({
    uri: process.env.NEXT_PUBLIC_HASURA_URL,    "HASURA": Unknown word.
    headers: {
      'content-type': 'application/json',
      'x-hasura-admin-secret': process.env.NEXT_PUBLIC_MONITORING_HASURA_ADMIN_SECRET as string,
    },
  });

  const errorLink = onError(({ graphQLErrors, networkError }) => {
    if (graphQLErrors) {
      for (const { message } of graphQLErrors) {
        const logMessage = `[API error]: Message: ${message}`;
        console.log(logMessage);
      }
    }
    if (networkError) {
      const message = `[Network error]: ${networkError}`;
      console.log(message);
    }
  });

  const apolloClient = new ApolloClient({
    link: httpLink,
    cache: new InMemoryCache(),
  });

  return <ApolloProvider client={apolloClient}>{children}</ApolloProvider>;
};

export default AppApolloProvider;
```

Figure 27. Code snippet showcasing the AppApolloProvider component

The AppApolloProvider serves as the backbone for GraphQL communication, connecting the application to the specified Hasura GraphQL endpoint. Here's a breakdown of its key functionalities:

HTTP Link Configuration:

- The component establishes an HTTP link to the specified Hasura GraphQL endpoint (**process.env.NEXT_PUBLIC_HASURA_URL**). This link serves as the communication bridge between InsightHub and the Hasura Cloud, allowing GraphQL queries and mutations.

Error Handling:

- The AppApolloProvider incorporates robust error-handling mechanisms using the onError function from Apollo Client. This ensures graceful handling of GraphQL errors and network issues, preventing disruptions in the application flow.

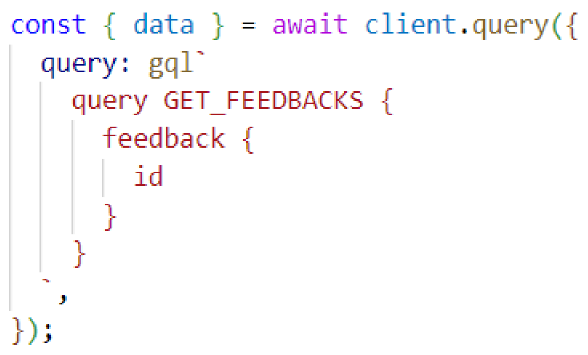
Apollo Client Initialization:

- The AppApolloProvider initializes an instance of the Apollo Client, combining the HTTP link and error link. This client becomes the core mechanism for GraphQL interactions, managing the flow of data between InsightHub and the Hasura Cloud.

Finally, the component wraps the entire InsightHub application with the ApolloProvider, ensuring that every component, page, and functionality within InsightHub gains seamless access to the configured Apollo Client for GraphQL interactions.

Interacting with the Database using GraphQL

After the configuration process, to demonstrate the use of GraphQL, the following code snippet showcasing interaction with the database. For instance, suppose there is a requirement for feedback-related information.



```
const { data } = await client.query({
  query: gql`
    query GET_FEEDBACKS {
      feedback {
        id
      }
    }
  `,
});
```

Figure 28. Example code snippet showcasing interaction with database

Below is a detailed analysis of the code:

1. **Sending a Request:** The code initiates a request to the database, seeking specific information. The **client.query** method executes a GraphQL query, sending a request to the GraphQL server and expecting a response based on the specified query.

2. **Specifying Request Details:** It structures the request details in a specified format, essentially saying, 'I want the IDs of feedback items.' The query is crafted to request specific data from the GraphQL server.

3. **Receiving and Storing the Response:** Once processed by the database, the relevant information is sent back. The response is stored in a variable called **data**. This variable contains the result of the query, holding the response received from the GraphQL server, including the requested information.

4.4.4 AWS S3 Bucket

Setup and Integration

The setup of the AWS S3 bucket for the project was handled by the company supervisor to ensure the security and confidentiality of sensitive data. The S3 bucket is configured with strict access controls, utilizing private configurations to restrict unauthorized access to its contents. Specifically, only authorized users or systems with the correct credentials are permitted to interact with or retrieve data from the S3 bucket. This restriction aims to strengthen the overall security of the project by minimizing the risk of unauthorized access to sensitive data stored in the AWS S3 bucket.

Following the setup, the S3 bucket URL is provided with the bucket key and bucket secret for the environment key. The bucket URL serves as the API endpoint for connecting the project to the AWS S3 bucket, enabling the project to interact with and access the contents of the S3 bucket, which is: *"https://s3.eu-central-1.amazonaws.com/opusflow-monitoring/monitoring".* Here's the breakdown of information about the URL:

1. **Region:** "eu-central-1" in the URL represents the AWS region where the S3 bucket is located. In this case, it is the EU (Frankfurt) region.
2. **Bucket Name:** "opusflow-monitoring" is the unique identifier for the AWS S3 bucket within the specified region.
3. **Path:** "/monitoring" at the end of the URL represents the path within the S3 bucket where the project will interact.

Secure File Interaction with AWS S3

To establish a secure connection between the AWS S3 bucket and the project, controlled and temporary access to specific S3 objects is ensured through the use of pre-signed URLs.

What is pre-signed URL?

A pre-signed URL is a specific URL generated by the application, granting users temporary access to perform specific actions on a designated S3 object. This access can be either for reading or writing an object, and the URL includes parameters set by your application. Three crucial parameters define the limitations of access:

1. **Bucket:** Represents the S3 bucket where the object is stored or will be stored.
2. **Key:** Signifies the name of the object within the specified bucket.
3. **Expires:** Indicates the duration of time the URL remains valid.

Expiration mechanism

Once the specified expiration time elapses, the user loses the ability to interact with the designated object. AWS ensures that access is only granted to correctly signed pre-signed URLs, reinforcing security measures by allowing access exclusively to the S3 bucket owner.

Limitations of Pre-Signed URLs

It's crucial to note that the permissions granted by a pre-signed URL are specific to the actions defined during its creation. For instance, if a pre-signed URL is generated for a GET (Read) action, attempting to use it for a PUT (Write) action will be unsuccessful.

Use of Pre-Signed URLs in application

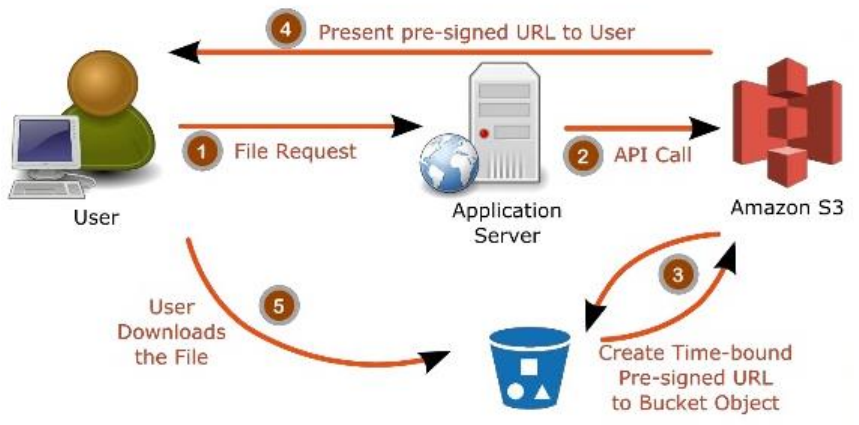

4 Present pre-signed URL to User
1 File Request
2 API Call
User
Application Server
Amazon S3
3
Create Time-bound Pre-signed URL to Bucket Object
5
User Downloads the File

Figure 29. Utilizing Pre-Signed URLs for Controlled Interaction with AWS S3 Bucket

For the file upload process to the S3 bucket, the project utilizes the API endpoints provided by Next.js. These endpoints are responsible for handling the upload function and require specific content parameters, namely the file type and file name. For instance, the following code represents the function call to the API endpoint for uploading feedback media files.

```
const response = await fetch("/api/feedback/upload", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ file_type: fileType, file_name: fileName }),
});
const { url } = await response.json();
return url;
```

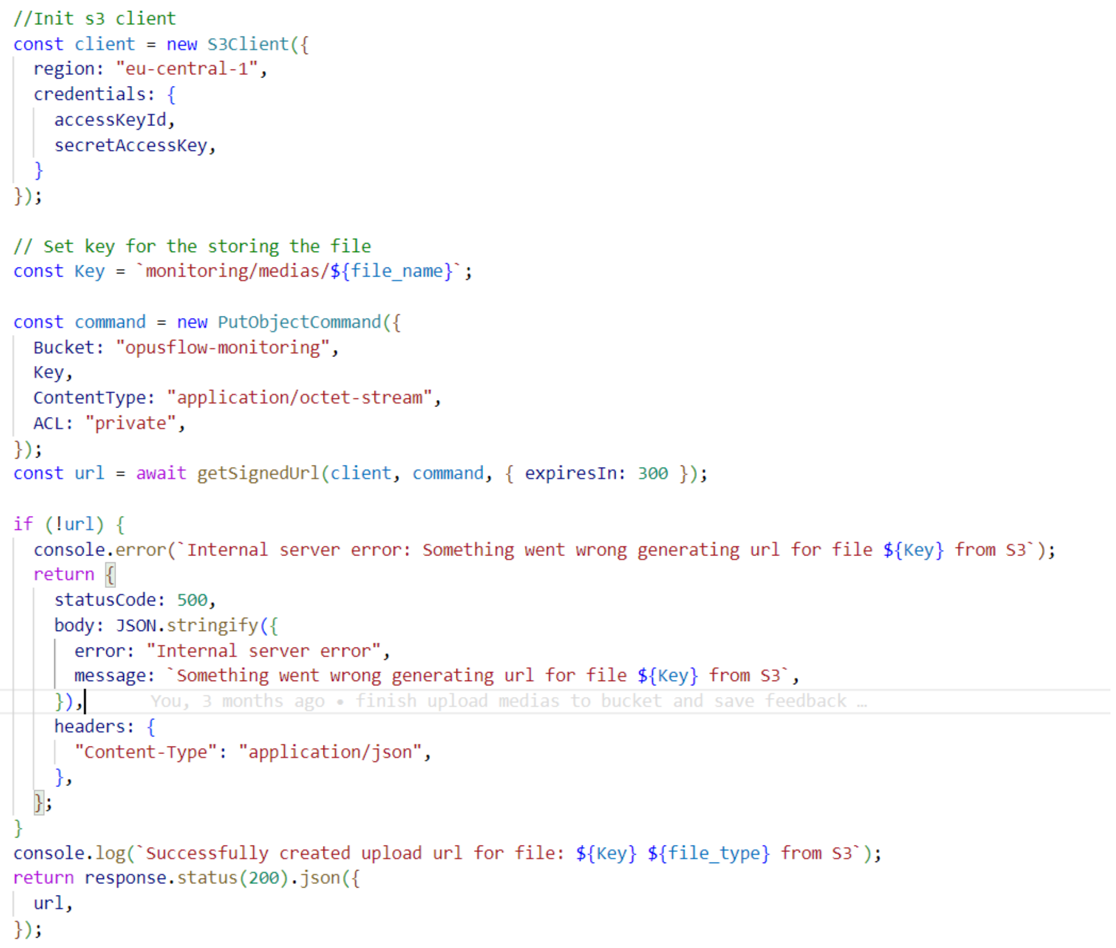
Figure 30. Code snippet showcasing a request to the server to upload a file

Below is a detailed analysis of the code:

1. **API Endpoint:** The **fetch** function is employed to make a POST request to the "/api/feedback/upload" endpoint. This endpoint is dedicated to handling file uploads.
2. **Request Configuration:** The request is configured with the necessary headers, specifying that the content type is JSON.
3. **Request Body:** The body of the request contains a JSON object with crucial information about the file, including its type (**file_type**) and name (**file_name**).
4. **Response Handling:** Upon a successful request, the response is processed. The JSON response is destructured to extract the **url** property, representing the location where the file has been uploaded to in the AWS S3 bucket.

5. **Returning the URL:** The extracted URL is then returned, providing a reference to the location of the uploaded file.

When the API call is made from the client side to the server side of Next.js, it triggers the function responsible for obtaining the pre-signed URL.



```
//Init s3 client
const client = new S3Client({
  region: "eu-central-1",
  credentials: {
    accessKeyId,
    secretAccessKey,
  }
});

// Set key for the storing the file
const Key = `monitoring/medias/${file_name}`;

const command = new PutObjectCommand({
  Bucket: "opusflow-monitoring",
  Key,
  ContentType: "application/octet-stream",
  ACL: "private",
});
const url = await getSignedUrl(client, command, { expiresIn: 300 });

if (!url) {
  console.error(`Internal server error: Something went wrong generating url for file ${Key} from S3`);
  return {
    statusCode: 500,
    body: JSON.stringify({
      error: "Internal server error",
      message: `Something went wrong generating url for file ${Key} from S3`,
    }),       You, 3 months ago • finish upload medias to bucket and save feedback …
    headers: {
      "Content-Type": "application/json",
    },
  };
}
console.log(`Successfully created upload url for file: ${Key} ${file_type} from S3`);
return response.status(200).json({
  url,
});
```

Figure 31. Code snippet showcasing API calls to AWS S3

Below is a detailed analysis of the code:

1. **S3 Client Initialization:** An S3 client is initialized with AWS credentials (access key and secret access key) and the specified region.

2. **Set Object Key:** The object key is set, defining the path and name under which the file will be stored in the S3 bucket.

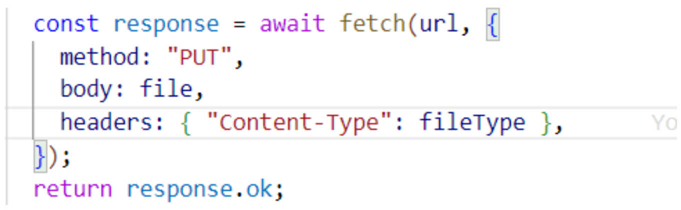
3. **Create PutObject Command:** A **PutObjectCommand** is created with essential details such as the bucket name, object key, content type, and ACL (Access Control List).

4. **Generate Pre-signed URL:** The **getSignedUrl** function is used to obtain a pre-signed URL for the specified **PutObjectCommand**. This URL is time-limited to ensure secure, temporary access.

5. **Error Handling:** If an issue arises during the URL generation, an error response is provided, indicating an internal server error.

6. **Successful URL Generation:** Upon successful URL generation, a JSON response is sent with the pre-signed URL.

The reason for obtaining the pre-signed URL is that it serves as a secure, time-limited token, granting temporary access to upload the file to a specific location in the AWS S3 bucket. Following the acquisition of the pre-signed URL, the subsequent step involves initiating a PUT request. This PUT request utilizes the pre-signed URL to securely upload the file content to the specified location in the S3 bucket.

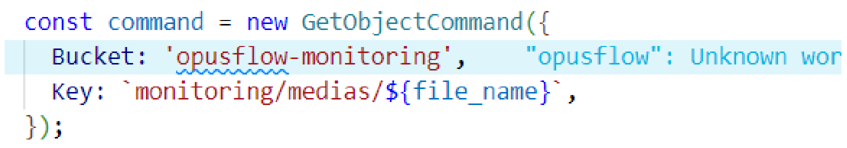


```
const response = await fetch(url, {
  method: "PUT",
  body: file,
  headers: { "Content-Type": fileType },        Yo
});
return response.ok;
```

Figure 32. Code snippet showcasing upload file to S3 bucket

The response from the S3 upload is checked, and the result is returned, indicating the success or failure of the upload operation.

For the file download process from the S3 bucket, the procedure is similar to the upload process. It starts with making API calls to the API endpoint for downloading files from the client side. Then, the function for downloading files from the server makes API calls to AWS S3 to obtain the pre-signed URL. The difference between the download and upload processes is outlined in the following section.



```
const command = new GetObjectCommand({
  Bucket: 'opusflow-monitoring',     "opusflow": Unknown wor
  Key: `monitoring/medias/${file_name}`,
});
```

Figure 33. Code snippet showcasing of creating a GetObjectCommand for file retrieval from AWS S3.

Below is a detailed analysis of the code:

1. **Create GetObject Command:** A **GetObjectCommand** is created with essential details such as the bucket name and object key (file path and name).

2. **Generate Pre-signed URL for Reading:** The **getSignedUrl** function is used to obtain a pre-signed URL specifically for reading (GET request) the specified object.

In the project, the pre-signed URL obtained during the download process serves as a media source. It can be dynamically assigned to the 'src' attribute of an HTML element, enabling the application to seamlessly display images or videos directly sourced from AWS S3. This approach ensures the secure and efficient retrieval of files from AWS S3 using pre-signed URLs. The generated URLs have a limited validity period to enhance security, and they enable the seamless integration of media content within the application.

4.4.5 Authentication & Authorization

Authentication in the project is managed differently for the SnapFlow Feedback component integrated into OpusFlow and the standalone OpusFlow InsightHub web application.

SnapFlow Feedback (Integrated with OpusFlow):

- **Tool Used:** Clerk
- **Authentication & Authorization Details:**
    - Clerk is utilized as the tool for authorization and authentication.
    - Only users registered in the OpusFlow system have access to the SnapFlow Feedback component.
    - Access is limited to registered OpusFlow users, ensuring that only authenticated users can provide feedback about the system.
    - OpusFlow employs a robust set of functions to ensure the security of API endpoints, such as media uploads to the S3 bucket .
    - Access is exclusively granted to logged-in users possessing valid tokens, effectively thwarting potential API spam and mitigating security risks associated with unauthorized access.

OpusFlow InsightHub:

- **Authentication Approach:**
    - For the InsightHub web application, a simplified authentication procedure is implemented.
    - This authentication is designed for the demo version and will transition to Clerk, as discussed in the recommendation section.
    - The authentication page includes a form with fields for email and password.
- **Key Features**:
    - The authentication code snippet includes a form for email and password.
    - Form validation is implemented using Yup.
    - A demo account's default values are provided for ease of testing.
    - The user is informed of login success or failure through messages.
- **Authorization and Guard**:
    - Authorization measures prohibit unauthorized users from accessing certain parts of the project, such as project details.
    - The AuthGuard component serves as a protective layer, wrapping the entire application layout.

- When an unauthenticated user attempts to access protected content, they are redirected to the login page with the appropriate return URL.
- The guard checks the authentication status using the useAuthContext hook and redirects users accordingly.
- This ensures that only authenticated users can view the detailed content of the project within OpusFlow InsightHub

This authentication approach ensures that users of the OpusFlow InsightHub web application, particularly those accessing the demo version, can securely log in with the provided credentials. For detail code snippet, refer to Appendix G.

4.4.6 Snapflow Feedback

Integration with OpusFlow

SnapFlow Feedback is a the component integrated into the OpusFlow system, which adopts a robust monorepo structure. The monorepo serves as a foundational architecture, consolidating multiple projects or applications into a single repository. This centralized approach simplifies development, streamlines code sharing, and ensures cohesive management of various components within the system.

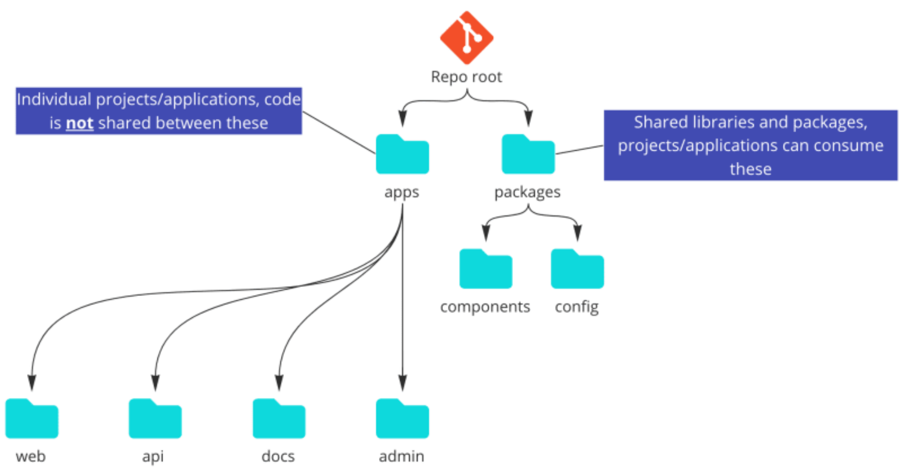

Repo root
Individual projects/applications, code is **not** shared between these
Shared libraries and packages, projects/applications can consume these
apps
packages
components
config
web
api
docs
admin

Figure 34. Monorepo structure

OpusFlow's monorepo is organized into distinct folders, with the "apps" directory containing the entire project, including OpusFlow and other projects. The "packages" folder, a key element, houses libraries and shared components designed for cross-project utilization.

SnapFlow Feedback will be developed within the "components" folder of the "packages" directory. SnapFlow. This intentional placement underscores its potential for deployment in future applications beyond OpusFlow, aligning with the scalable and reusable coding principles of the monorepo design.

To ensure consistent operation without disrupting user activities, the component is strategically added to the root "_app" file of OpusFlow. This file, serving as a pivotal point for application initialization and important configurations, aligns with the current setup of Next.js, contributing to the seamless execution of SnapFlow Feedback

File organization

In SnapFlow Feedback, A high-quality coding standards was maintained by following a carefully structured approach, as outlined in the quality management section. The illustration below demonstrates the organization of components, highlighting a deliberate distribution of code across various files and folders. This strategic separation of concerns ensures a clean and well-structured codebase.

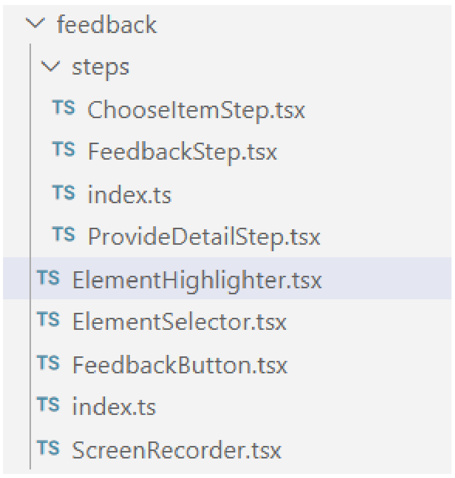

```
feedback
  steps
    ChooseItemStep.tsx
    FeedbackStep.tsx
    index.ts
    ProvideDetailStep.tsx
  ElementHighlighter.tsx
  ElementSelector.tsx
  FeedbackButton.tsx
  index.ts
  ScreenRecorder.tsx
```

Figure 35. SnapFlow Feedback Component Folder Structure

In the "steps" folder, there are three files: ChooseItemStep, FeedbackStep, and ProvideDetailStep. Each file corresponds to a specific step in the feedback submission process, adhering to the design philosophy of creating concise functions with singular responsibilities. This approach not only enhances maintainability but also aligns with the widely-accepted software design principle of "Don't Repeat Yourself" (DRY), mitigating unnecessary code duplication.

In addition to these steps, the functionality for highlighting the selected component is distributed across two files: ElementHighLighter and ElementSelector. A separate file, ScreenRecorder, houses the functions related to screen recording. This modular breakdown ensures a clear separation of concerns and enhances the overall readability and maintainability of the code.

The central component, FeedbackButton, serves as the core element of SnapFlow Feedback, encapsulating imports from various files to construct the complete SnapFlow Feedback Button popover form. This component exemplifies the structured formatting principles employed in the project, minimizing deep nesting for enhanced readability. Moreover, the inclusion of standardized

Features breakdown

SnapFlow Feedback component is designed to enhance user accessibility. Developed as a FAB (Floating Action Button) positioned in the right corner of the screen, users can conveniently access the feedback button from any location within the application. Clicking on this button triggers a popover form, as illustrated below, streamlining the feedback submission process and promoting user engagement.

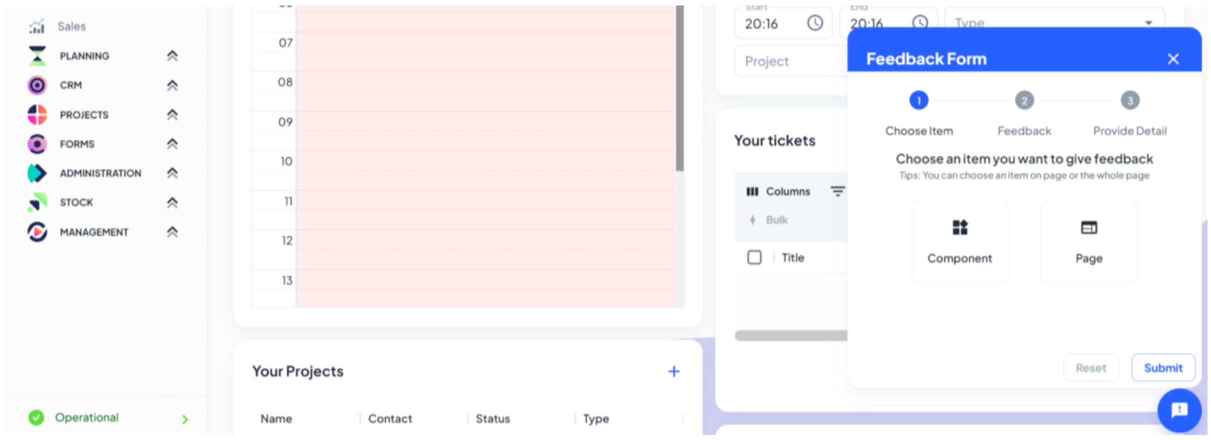


Figure 36. SnapFlow Feedback - Popover Form Triggered

To enhancing the user's capability to provide feedback across various pages and components, SnapFlow Feedback incorporates a screenshot function. This feature empowers users by allowing them to choose between capturing a specific component or an entire page.

The development of the component selection function drew inspiration from the user-friendly feature found in the Chrome browser, known as "select an element to inspect." Refer to the illustration below for a visual representation of the component screenshot functionality after user select the component button in the step 1 choose item.

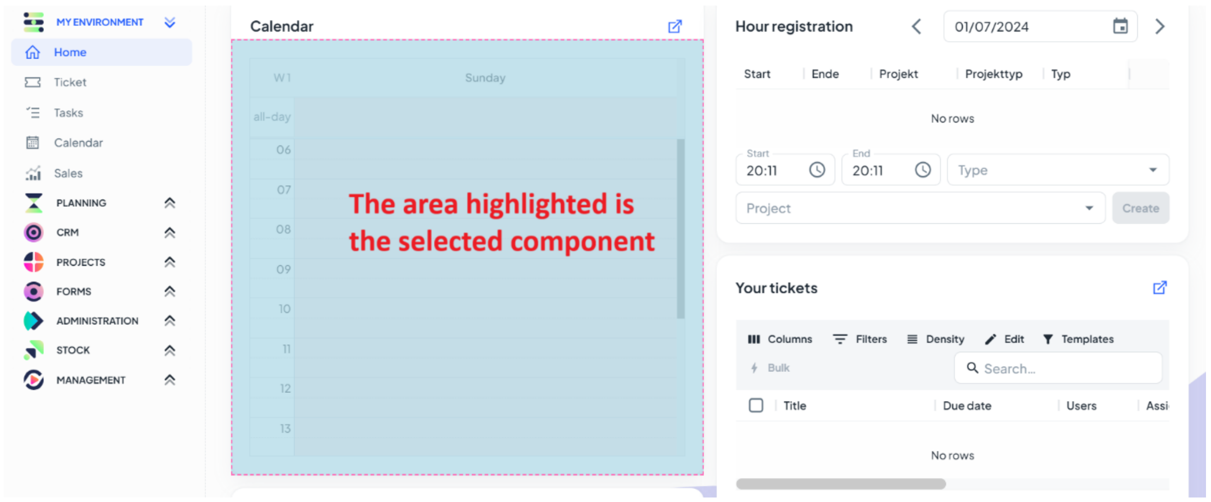
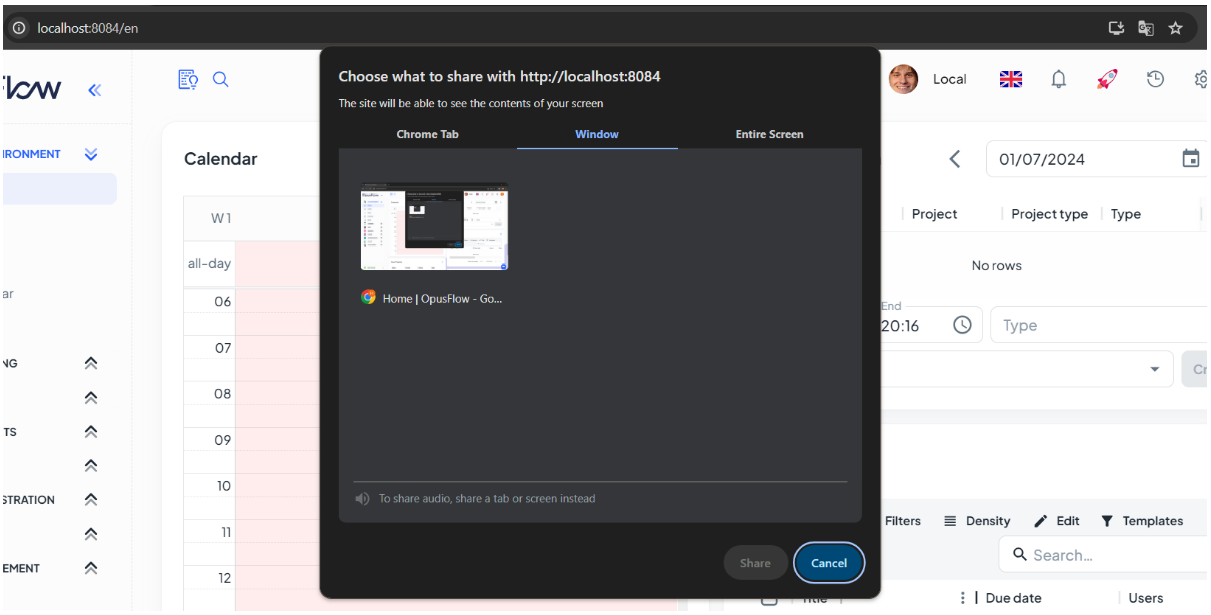


Figure 37. SnapFlow Feedback - Component Selected Triggered

Upon hovering the mouse over a component, a custom highlighter draws a blue rectangle with a pink dashed border around the selected component. This implementation aims to provide an intuitive and visually enhanced user experience during the feedback submission process. For page captures, the screenshot function simply captures the entire page.

Another pivotal feature of SnapFlow Feedback is its screen recording functionality, designed to empower users to provide contextual information along with their feedback, as outlined in Step 3 of the wire design. The screen recording feature is a bespoke development, created entirely from scratch without relying on external libraries. By incorporating the MediaRecorder from MDN references, SnapFlow Feedback ensures a robust and dependable recording mechanism. This enables seamless integration with modern browsers, ensuring users have consistent and reliable screen recording capabilities. When a user initiates the screen recording by clicking the record button, a browser popup prompts them to select the screen they wish to record, as illustrated below.


localhost:8084/en

Choose what to share with http://localhost:8084

The site will be able to see the contents of your screen

Figure 38. SnapFlow Feedback – Screen Record Button Triggered

Once the user selects the desired screen, an notification box appears in the middle-bottom of the screen, indicating that the screen is actively recording.

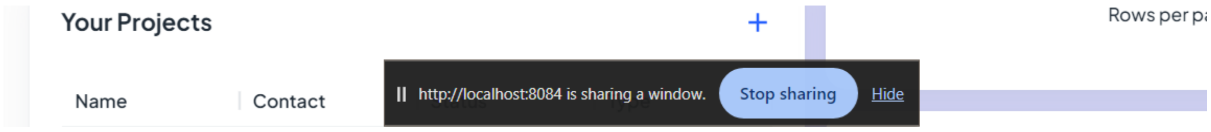

http://localhost:8084 is sharing a window.

Figure 39. SnapFlow Feedback – Screen Record Mode

Users can end their recording session by selecting the 'Stop Sharing' button or when reaching the 30-second limit imposed by the screen recording feature.

Upon completing the final step of the feedback form, all media files are uploaded to the AWS S3 bucket, and their corresponding links are included in the feedback data. Then feedback data is sent to GraphQL to initiate the task creation process. Refer to Appendix H for compressive detail.

Error Handling

The SnapShot Feedback compoent employs the Yup validation schema to ensure users submit complete and valid feedback. Required fields such as type, element, and issue are checked for completeness and adherence to specified constraints. If any validation fails, the submission is halted, providing immediate feedback to users. Special attention is given to video submissions, requiring at least one video file for comprehensive feedback. This approach enhances the user experience by guiding them to correct any missing or incorrectly formatted information in real-time. By leveraging Yup, the system minimizes the risk of incomplete or erroneous feedback submissions, ensuring the reliability and accuracy of captured data.

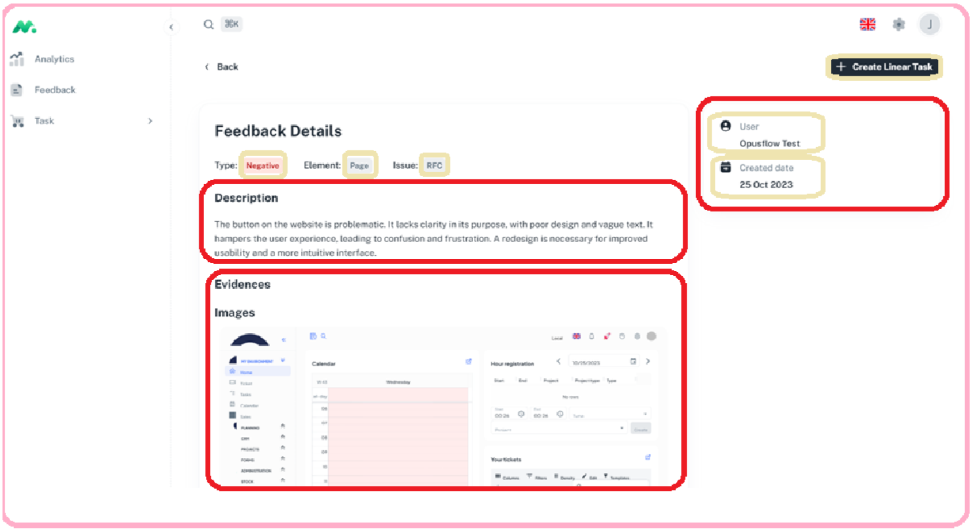
4.4.7 OpusFlow InsightHub

File organization

OpusFlow InsightHub is a dedicated web application designed to manage feedback independently. With a focus on meeting specific requirements and packed with features, InsightHub acts as a centralized hub. It facilitates in-depth analysis of feedback and the effortless generation of tasks. This section will explore its fundamental functionalities and distinctive components.

The project's foundation, built upon the company's the 'Minimal' repository within the monorepo. The core structure of the project is as follows:

- **__tests__:** Manage automatic test file of the application.
- **src folder:** The heart of the application, containing the source code with distinct subfolders for various functionalities.
    - **app folder:** Manage all pages of the application.
        - *api folder:* Handles API-related functionality and API endpoints.
        - *auth folder:* Manages authentication-related pages.
    - *components:* Houses reusable UI components.
    - *hooks:* Contains custom hooks.
    - *layouts:* Manages the layout components.
    - *routes:* Handles routing configurations.
    - *sections:* Contains different sections of the application.
    - *utils:* Houses utility functions.
    - *types:* Manages TypeScript types.



Type: Negative Element: Page Issue: RFC

User
Opusflow Test

Created date
25 Oct 2023

Description

The button on the website is problematic. It lacks clarity in its purpose, with poor design and vague text. It hampers the user experience, leading to confusion and frustration. A redesign is necessary for improved usability and a more intuitive interface.

Evidences

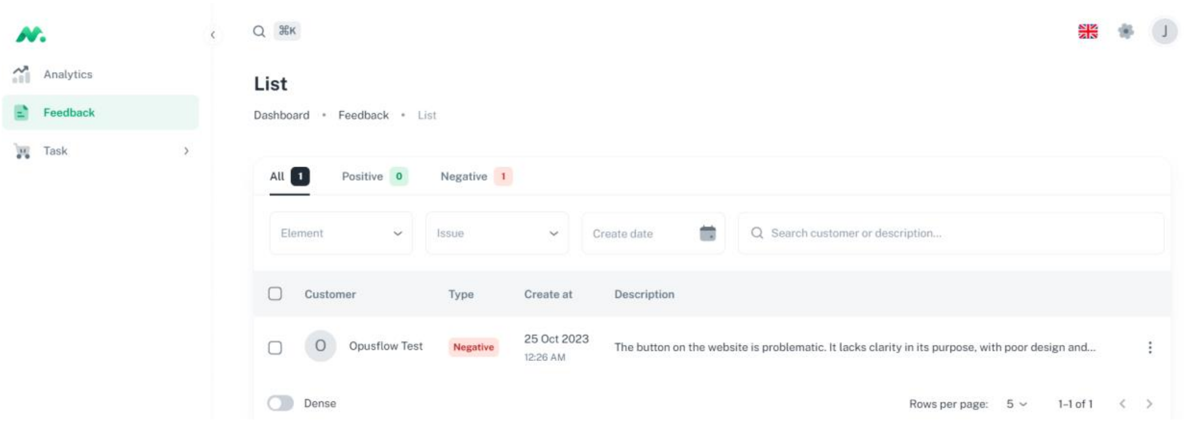
Images

Figure 40. Illustration of a Page Structure

For a visual representation, the illustration above demonstrates how the structure of a page is constructed. The pink border signifies the page, the red border represents the section, and the yellow border denotes the components.

Features breakdown

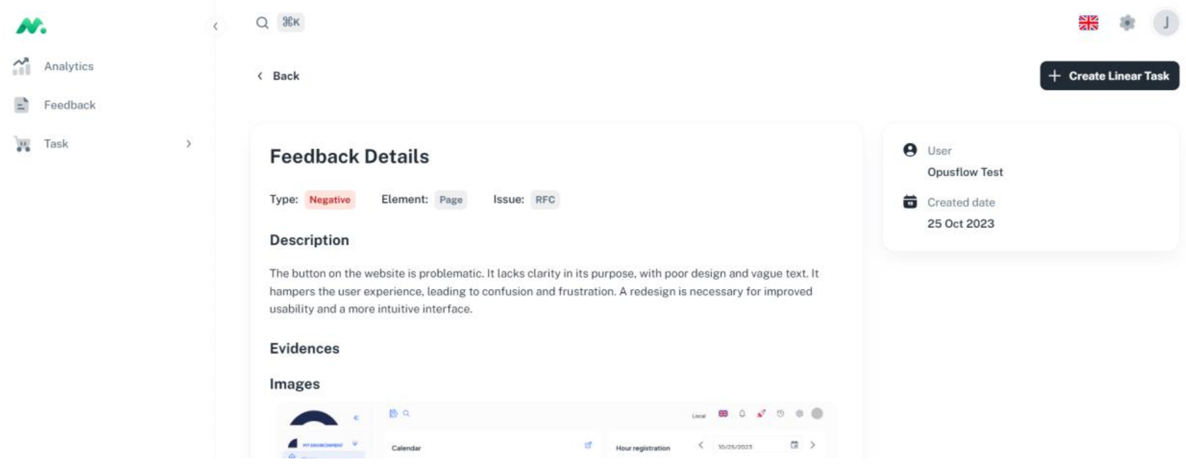
The implementation of the Centralized Feedback Dashboard in InsightHub revolves around creating a dynamic and consolidated viewing platform. This unified space empowers users to effortlessly access, manage, and review all feedback, fostering a streamlined approach to information handling. To enhance usability, advanced filtering and sorting capabilities are seamlessly integrated, allowing users to efficiently organize feedback based on various parameters.



Customer	Type	Create at	Description
Opusflow Test	Negative	25 Oct 2023 12:26 AM	The button on the website is problematic. It lacks clarity in its purpose, with poor design and...

Figure 41. OpusFlow InsightHub - Feedback Dashboard Page

The Feedback Detail Page in InsightHub ensures a detailed and organized display of each feedback entry. Purposefully designed, it presents all essential information in a structured manner, allowing users to quickly access and understand each feedback submission. The user-friendly interface enhances overall usability, emphasizing simplicity and accessibility in design.



Feedback Details

Type: Negative Element: Page Issue: RFC

Description

The button on the website is problematic. It lacks clarity in its purpose, with poor design and vague text. It hampers the user experience, leading to confusion and frustration. A redesign is necessary for improved usability and a more intuitive interface.

Evidences

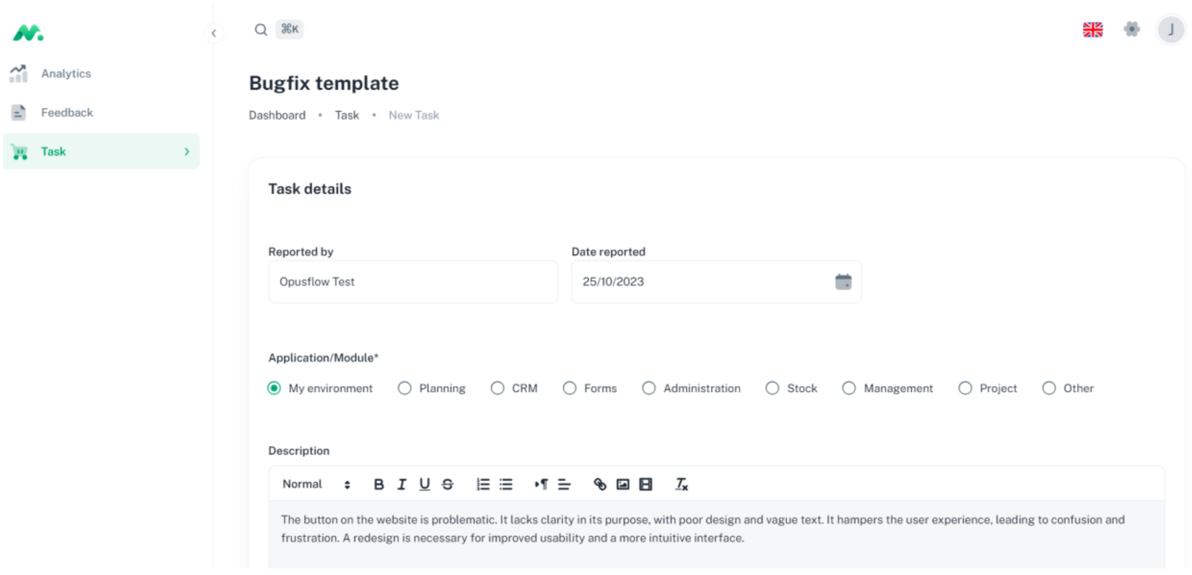
Images

User
Opusflow Test

Created date
25 Oct 2023

Figure 42. OpusFlow InsightHub - Feedback Detail Page

Moving to the Task Initiation Page, InsightHub's implementation enables users to effortlessly create tasks within the Linear application. This page provides pre-filled forms based on feedback data, streamlining the task initiation process significantly. Request for Change (RFC) and Bug Report form are automatically selected based on feedback issue. This integration aligns with the design's vision, enhancing the efficiency of converting feedback into actionable tasks.



Bugfix template

Dashboard • Task • New Task

Task details

Reported by	Opusflow Test
Date reported	25/10/2023

Application/Module*

[✓ My environment] Planning CRM Forms Administration Stock Management Project Other

Description

Normal

The button on the website is problematic. It lacks clarity in its purpose, with poor design and vague text. It hampers the user experience, leading to confusion and frustration. A redesign is necessary for improved usability and a more intuitive interface.

Figure 43. OpusFlow InsightHub - Bugfix Template Page

The data filled in the form will be used to create a Linear task by retrieving and validating video URLs from feedback data. The form data is converted into Markdown format. The final step involves creating a Linear task using the formatted data. Refer to Appendix I for compressive detail.

Error Handling

The process of handling errors within the system is facilitated through two key components: the Yup library and the useForm hook from the react-hook-form library. Following is the break down of the error-handling mechanism operates:

1. **Yup Schema:**
    - Rules for each form field are defined using Yup in schema.
    - The emphasis is on ensuring all required fields have accurate information.
    - User-friendly error messages are built into the schema for clear feedback on incomplete mandatory fields.
2. **react-hook-form (useForm Hook):**
    - The useForm hook manages the form, handling both state and validation.
    - Integration with the Yup schema is seamless, thanks to yupResolver.
    - This hook keeps track of field values, errors, and submission status.
    - If validation issues arise during submission, the useForm hook provides direct access to detailed error information, making error messaging precise and enhancing the user experience.

4.5 Testing

Ensuring the robustness and reliability of the project involved a comprehensive testing strategy covering various aspects of the application, aligning with the specified requirements. The testing process comprised both unit testing, focusing on specific features, and end-to-end (e2e) testing, validating key use cases integral to the application's functionality. Refer to Appendix K for complete test details.

4.5.1 Unit Testing

Unit testing focused on scrutinizing specific features within OpusFlow InsightHub & SnapFlow Feedback, encompassing critical functionalities such as capturing screenshots, recording screens, creating feedback entries, viewing feedback lists, and generating tasks based on feedback. This meticulous approach facilitated a thorough examination of each feature's performance, allowing for the detection of potential issues or deviations from the defined requirements.

Below is a summary table illustrating the unit testing outcomes for these specific features:

Test No.	Rq. No.	Description	Test Case	Expected Result	Actual Result	Status
UT1	F2	Verify the application's ability to capture screenshots.	Load a page and initiate screenshot capture.	Screenshot is captured successfully.	Screenshot captured successfully.	Pass
UT2	F2	Validate the functionality of screen recording.	Navigate to a page and start recording.	Screen recording is initiated without errors.	Screen recording initiated without errors.	Pass
UT3	F1	Test the creation of feedback entries.	Submit feedback via the feedback module.	Feedback entry is created successfully.	Feedback entry created successfully.	Pass
UT4	F4	Ensure the accurate display of the feedback list.	Access the feedback list page.	List of feedback entries is displayed.	List of feedback entries displayed correctly.	Pass
UT5	F3	Validate the creation of tasks based on feedback.	Generate a task using feedback information.	Task is created successfully in the Linear app.	Task created successfully in the Linear app.	Pass

In this table:

- "Feature" describes the specific functionality being tested.
- "Description" provides a brief overview of the feature being tested.
- "Test Case" outlines the steps taken during testing.
- "Expected Result" details the anticipated outcome of the test.

- "Actual Result" records the observed result during testing.
- "Status" indicates whether the test passed or encountered issues.

Results from unit testing revealed the robustness of individual features, with all functionalities performing as expected. The validation of core actions, such as creating feedback entries and initiating tasks, demonstrated the reliability of these features within the application.

4.5.2 E2E Testing

The comprehensive end-to-end (e2e) testing phase focused on validating the functionality of two pivotal use cases integral to the project's success.

Test Case No. 1: User Feedback Submission

Test Scenario: Successful User Feedback Submission

- **Outcome:** Passed
    - *Observation:* A confirmation message "Feedback submitted successfully" is displayed.
    - *Observation:* The submitted feedback is accurately recorded in the system.

Test Scenario: Unsuccessful User Feedback Submission

- **Outcome:** Passed
    - *Observation:* An error message is received for missing information.
    - *Observation:* The system does not record incomplete feedback, showcasing proper validation.

Test Case No. 2: Feedback Analysis

Test Scenario: Successful Feedback Analysis and Task Creation

- **Outcome:** Passed
    - *Observation:* Successful access to the feedback dashboard.
    - *Observation:* Filters and sorting options function as expected.
    - *Observation:* Clear display of feedback trends and summary.
    - *Observation:* Tasks are created in the Linear app without issues.

Test Scenario: Unsuccessful Feedback Analysis

- **Outcome:** Passed
    - *Observation:* No errors are encountered during feedback analysis.
    - *Observation:* Any issues with filtering and sorting were not observed.
    - *Observation:* Successful creation of tasks in the Linear app.

5. Results

5.1 Evaluation of Requirements

In reviewing the functional and nonfunctional requirements for the entire project, the evaluation reflects a comprehensive assessment of adherence to specified criteria.

Functional Requirements:

1. **F1 - Submission of Feedback**
    - **Evaluation:** Successfully validated through unit testing (UT3) and confirmed in the SnapFlow Feedback design with a 3-step form.
    - **Status:** Pass
2. **F2 - Capturing Context of Feedback**
    - **Evaluation:** Unit testing (UT1 and UT2) validated the application's ability to capture screenshots and initiate screen recording, confirming adherence to the requirement.
    - **Status:** Pass
3. **F3 - Task Initiation in Linear App**
    - **Evaluation:** Unit testing (UT5) ensured the creation of tasks based on feedback in the Linear app, confirming fulfillment of the requirement.
    - **Status:** Pass
4. **F4 - Centralized Feedback Dashboard**
    - **Evaluation:** Unit testing (UT4) verified the accurate display of the feedback list, ensuring compliance with the requirement.
    - **Status:** Pass
5. **F5 - Feedback Correlation View**
    - **Evaluation:** Successfully confirmed during e2e testing, validating the implementation of a mechanism to visualize associations between different user feedback.
    - **Status:** Pass

Non-functional Requirements:

1. **NF1 - System Performance**
    - **Evaluation:** Confirmed through the application's usage, ensuring the feedback module responds to user interactions within the specified time frame of 0.5 seconds.
    - **Status:** Pass
2. **NF2 - System Usability**

- **Evaluation:** Confirmed through the SnapFlow Feedback design, featuring a 3-step form for feedback submission, meeting the criterion of 3-5 interactions.
- **Status:** Pass

3. **NF3 - Data Security**
    - **Evaluation:** Verified through the utilization of AWS S3, media files are stored in a secure manner with a private setup, guaranteeing the protection of sensitive data. Enhanced security is further ensured by requiring API-related keys for both SnapFlow Feedback and OpusFlow InsightHub. However, several security vulnerabilities exists in the authentication and authorization process of OpusFlow InsightHub, especially concerning API endpoints. The current solution is limited to the local machine, making it imperative to address this issue before the release.
    - **Status:** Not Pass

5.2 Reflections

5.2.1 Working methods

The decision to use Kanban, supplemented by Notion, had several benefits and drawbacks:

Pros:

1. Effective Use of Notion's Kanban Template: The project leverages the potential of the Notion tool, particularly its Kanban template, enhancing task management and accountability.
2. Enhanced Flexibility: Kanban's inherent flexibility allows for various changes and adjustments throughout the project, based on feedback and input from different stakeholders.
3. Visualization of Workflow: The project's progress is easily visualized through Kanban's systematic structure, beneficial for tracking the status of various tasks and features.
4. Continuous Task Delivery: Kanban's ability to handle continuous task delivery ensures a steady workflow and progress towards project completion.
5. Granular Task Management: The task breakdown methodology of Kanban facilitates a more detailed view of tasks and an effective prioritization strategy.

Cons:

1. Learning Curve: There could be a steep learning curve for those unfamiliar with Kanban or Notion, affecting their ability to understand and effectively utilize these tools.
2. Potential for Overload: While Kanban emphasizes continuous delivery, without proper management, this could potentially lead to task overload, affecting the quality of work.
3. Less Structured than Scrum: Compared to Scrum, Kanban lacks predefined roles and rituals, potentially leading to confusion or lack of clarity.

4. Dependence on Technology: Given that Notion is a digital tool, access to the Kanban board requires a stable internet connection. In unstable or low-connectivity situations, this could potentially disrupt the workflow.

5.2.2. Challenges and Lessons Learned

One major obstacle that significantly impacted the original project's timeline and deliverables was the delayed communication with the supervisor. The absence of clear guidance and a well-defined structure for the report resulted in a misalignment between the project's development and reporting phases. Consequently, the initial version of the project concept, submitted for review, revealed numerous issues and inconsistencies, necessitating extensive revisions.

The lack of early and consistent communication channels impeded the timely understanding of expectations and reporting requirements. This misalignment created a bottleneck in the final stages, where the need for substantial report revisions coincided with the remaining development tasks and bug fixes.

A crucial takeaway from this experience underscores the paramount importance of proactive and consistent communication with project stakeholders, particularly supervisors or mentors. Early engagement and clarification of expectations could have provided a clearer roadmap for both project development and reporting, mitigating the last-minute rush and potential project shortcomings.

Moving forward, a commitment to more robust project management practices will be implemented, encompassing regular check-ins, milestone reviews, and clear reporting guidelines. This approach aims to ensure a smoother and more coordinated execution of future projects, minimizing potential roadblocks arising from miscommunication or unclear expectations.

5.3 Limitations

During the development of the project, certain limitations were identified, requiring careful consideration for future enhancements. These challenges, while navigated successfully, offer valuable insights into areas for potential improvement:

5.3.1 html2canvas library constraints

- **Issue Description:** The html2canvas library used for capturing screenshots within OpusFlow InsightHub faced challenges due to its lack of updates. This limitation hinders its ability to capture elements with new CSS properties or intricate styles.

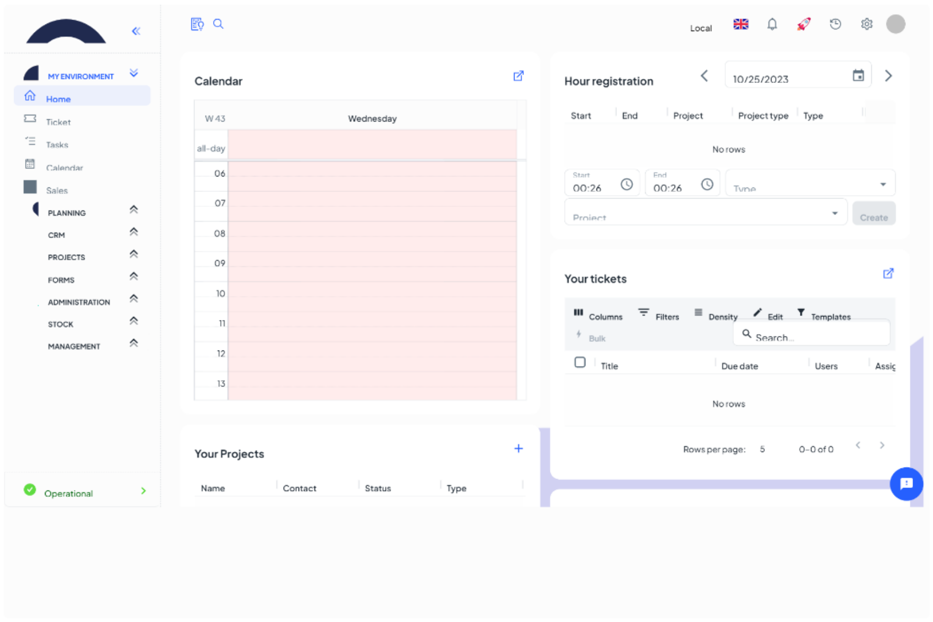
- **Background:** html2canvas operates by rendering DOM elements into a canvas and converting it into an image. However, it may struggle with certain CSS properties or complex designs introduced after the last library update.

- **Impact:** This limitation becomes evident when attempting to screenshot components within OpusFlow that utilize new CSS features or styling. Despite the absence of direct replacements for html2canvas, the screen recording feature effectively captures the essence of user feedback.

5.3.2 Integration challenges with Linear for video files in task description

- **Issue Description:** An integration challenge surfaced during the Linear task creation process, where videos attached to tasks were not visible in the task description. Additionally, the URL link to the video stored in Linear's private cloud was inaccessible. Here is an example where the embedded version of the video does not display within the Linear app.

- **Actual Result:** abc
- **Screenshot:**



- **Videos:**

https://uploads.linear.app/9c8c43f5-8ba9-458b-bb5b-907a165a8f8e/f99e65ce-788c-496f-b4f4-5d1af24d2ec6/dbc5e907-63f7-49bc-8834-03cdcc53629e

Figure 44. Video Embed Issue in Linear App

- **Background:** Linear, as the task management platform, may have limitations in displaying embedded videos directly in the task description, an aspect controlled by the Linear system itself.

- **Impact:** Although the Linear Support Desk acknowledged this issue, no immediate solution was available within the Linear ecosystem. A potential workaround involves exploring alternative hosting solutions for images or videos outside the Linear Private Cloud, ensuring seamless integration and visibility of multimedia content within tasks.

5.3.3 Lacking secure authentication & authorization in OpusFlow InsightHub

- **Issue Description:** The current authentication mechanism in OpusFlow InsightHub lacks robust security measures, relying on a basic form for email and password. The API endpoints are not protected to unauthorize called from outside of the system.
- **Background:** The existing authentication process, initially designed for the demo version, is scheduled for an upgrade to Clerk, along with an adjustment to the authentication method for OpusFlow's API.
- **Impact:** Insufficient authentication security exposes the InsightHub application to potential breaches, compromising user data and eroding trust. As the system evolves or is intended for use, enhancing authentication measures becomes paramount to ensure a resilient and secure user environment.

6. Conclusion

In response to the research question—how a solution can be developed to efficiently collect user feedback and translate it into meaningful improvements for the current system—the comprehensive project, encompassing SnapFlow Feedback and OpusFlow InsightHub, has successfully addressed critical needs within the OpusFlow ERP system.

The integration of SnapFlow Feedback directly with OpusFlow fulfills users' demand for a straightforward and effective feedback process. Simultaneously, OpusFlow InsightHub empowers the consultancy team by providing a centralized dashboard for viewing, filtering, and sorting user feedback data. This collaborative approach allows for the initiation of tasks based on user feedback and ensures that all collected feedback contributes to actionable insights.

While the project has encountered challenges, notably with the html2canvas library, integration issues with Linear, and underdeveloped authentication and authorization in OpusFlow InsightHub, these limitations serve as valuable insights for potential enhancements in future iterations. The commitment to ongoing improvements and the integration of user feedback into the development process will be crucial for ensuring OpusFlow's continuous evolution and alignment with user expectations.

It's important to note that the proposed solution has been developed but is not yet integrated into the current OpusFlow system due to various limitations that need to be addressed before being introduced to OpusFlow users. Despite these challenges, the project's success in addressing identified needs and laying the foundation for an adaptive and user-centric system underscores its ability to leverage innovative solutions and technology integration. This success not only enhances the overall user experience but also facilitates continuous improvements within the OpusFlow ERP system.

7. Recommendation

To strengthen the current solution and overcome identified limitations, consider the following recommendations:

1. **Enhance Feedback Detail:** Consider including the page information (the url of the page) in the feedback details. This additional data will provide a better contextual overview of where the feedback was captured, aiding in a more comprehensive understanding of the feedback context.

2. **Implement Clerk to OpusFlow InsightHub:** Explore the integration of Clerk into OpusFlow InsightHub for a robust and secure authentication and authorization setup. Clerk aligns seamlessly with OpusFlow's technology stack and offers an easy-to-manage solution for controlling access to the application. This step will enhance the overall security posture of OpusFlow InsightHub, ensuring a safer and more reliable user experience.

3. **Apply OpusFlow's Authentication Method to OpusFlow InsightHub**: Extend the proven authentication method utilized by OpusFlow to OpusFlow InsightHub, ensuring consistent and secure user access across both platforms. Additionally, apply this authentication approach to safeguard API endpoints, fortifying the overall security framework of OpusFlow InsightHub and fostering a reliable user experience.

References

This section contains a list of resources that were referenced in the writing of this document.

1. Haije, E. G. (2023, September 6). How to utilise user feedback for software development. Mopinion. https://mopinion.com/user-feedback-for-software-development/#:~:text=User%20feedback%20supplies%20you%20with,improve%20the%20overall%20user%20experience
2. Kanika. (2023, October 20). Website Feedback: Questions, Best Practices, Types, Use Cases & More. Website Feedback: Questions, Best Practices, Types, Use Cases & More. https://www.zonkafeedback.com/blog/feedback-on-website#:~:text=Website%20feedback%20helps%20to%20get,functionality%2C%20and%20overall%20user%20experience
3. Team, A. E. C. (n.d.). A detailed comparison between Kanban and Scrum. https://business.adobe.com/blog/basics/kanban-vs-scrum#:~:text=Kanban%20is%20centered%20around%20visualizing,on%20delivering%20chunks%20of%20items
4. Why use Kanban Method? | Kanban Tool. (n.d.). Kanban Tool. https://kanbantool.com/kanban-method

Bibliography

This section contains a list of resources that were accessed but not referenced in the writing of this document.

1. Gopal, T. (2022, June 28). Hasura Design Philosophy. Hasura GraphQL Engine Blog. https://hasura.io/blog/how-hasura-works/
2. Using presigned URLs to identify per-requester usage of Amazon S3 | Amazon Web Services. (2022, October 19). Amazon Web Services. https://aws.amazon.com/blogs/storage/using-presigned-urls-to-identify-per-requester-usage-of-amazon-s3/
3. Working with the GraphQL API - Linear API. (n.d.). https://developers.linear.app/docs/graphql/working-with-the-graphql-api
4. Gaurav, S. (2023, February 28). Git Branching Strategy- scaler Topics. Scaler Topics. https://www.scaler.com/topics/git/git-branching-strategy/
5. Aws. (n.d.). GitHub - aws/aws-sdk-js: AWS SDK for JavaScript in the browser and Node.js. GitHub. https://github.com/aws/aws-sdk-js
6. Axios. (n.d.). https://axios-http.com/
7. Home. (n.d.). React Hook Form - Simple React Forms Validation. https://www.react-hook-form.com/
8. html2canvas - Screenshots with JavaScript. (n.d.). https://html2canvas.hertzen.com/
9. Introduction to Apollo client. (n.d.). Apollo Docs. https://www.apollographql.com/docs/react/
10. JavaScript with syntax for types. (n.d.). https://www.typescriptlang.org/
11. Jquense. (n.d.). GitHub - jquense/yup: Dead simple Object schema validation. GitHub. https://github.com/jquense/yup
12. Linear. (n.d.). GitHub - linear/linear: Tools, SDK's and plugins for Linear. GitHub. https://github.com/linear/linear#readme
13. Material UI: React components that implement Material Design. (n.d.). https://mui.com/material-ui/
14. Next.Js by Vercel - the React framework. (n.d.). https://nextjs.org/
15. Node-Fetch. (n.d.). GitHub - node-fetch/node-fetch: A light-weight module that brings the Fetch API to Node.js. GitHub. https://github.com/node-fetch/node-fetch
16. React. (n.d.). https://react.dev/
17. Yet another react lightbox. (n.d.). Yet Another React Lightbox. https://yet-another-react-lightbox.com/
18. Vogel, J. (n.d.). ICT Research Methods — Methods Pack for research in ICT. ICT Research Methods. https://ictresearchmethods.nl/dot-framework/
19. Graduation. (n.d.). Figma. https://www.figma.com/file/AlKJxcDl8JMVJprCvEVHYN/Graduation?type=design&node-id=6%3A612&mode=design&t=mEFqxgl3j4YwukUG-1

Appendix A: Detailed transcripts of interview with QA manager

Huy Ton (Student): Good day, Naief. Thank you for taking the time to discuss the user feedback process in OpusFlow. Let's dive into the questions. Could you provide an overview of the current process for managing user feedback in OpusFlow?

Naief Bitar (QA Manager): Certainly, Huy. When a customer issues a bug report or makes a request for the development of a feature, they contact the consultancy through email, phone calls, or the Hellonext platform. The consultancy team then works with the client to gain more details about the feedback, examining the issues. If the issue or feature is deemed necessary, the consultancy team creates a task on Linear, a third-party application we use as our task management platform.

Huy Ton (Student): Thank you, Naief. Now, let's talk about who are the key stakeholders involved in the user feedback process, and what roles do they play?

Naief Bitar (QA Manager): The key stakeholders involved in the user feedback process include the customer, the consultancy team, the development team, and the management and QA team. The consultancy team is primarily responsible for interacting with customers, gathering feedback details, and creating tasks on Linear. The development team handles the actual implementation of tasks, while the management and QA team ensures that the tasks are valid and ready for production release.

Huy Ton (Student): Thank you for clarifying that. Now, regarding external tools, is there any integrated into the feedback process, and how does the consultancy team utilize it to manage tasks based on customer feedback?

Naief Bitar (QA Manager): Absolutely. As mentioned, we use Linear as our external tool for managing tasks. Linear helps us keep track of task statuses, from development to completion. The consultancy team creates tasks on Linear, and this tool facilitates communication and collaboration across teams involved in the feedback handling process.

Huy Ton (Student): Perfect. Can you elaborate on how a task progresses from the initial feedback stage to completion?

Naief Bitar (QA Manager): Certainly. After the consultancy team creates a task on Linear, the development team follows the Linear Method for handling tasks. This involves writing detailed issues, breaking them down, assigning a single owner, and prioritizing them. The team works in 1 or 2-week cycles, providing flexibility and effective time management. Regular updates are made to the main branch, and the implementation is made available in production. The QA team then tests the task to ensure it runs smoothly in the production environment before notifying the consultancy that the feedback has been successfully addressed.

Huy Ton (Student): Thank you for your time.

Appendix B: Detailed transcripts of interview with CTO

Huy Ton (Student): Good day, Joey. Thank you for taking the time to discuss my graduation project. To start, could you provide an overview of OpusFlow, its target audience, and the specific sector it serves, such as solar panel installers and wind turbine maintenance?

Joey Teunissen (CTO): Of course, Huy. OpusFlow is a B2B Enterprise ERP SaaS primarily serving companies in the sustainable sector, including businesses involved in solar panel installation and wind turbine maintenance. Our target audience consists of these companies, and they are the end users of our ERP system.

Huy Ton: Great, that provides some valuable context. Now, let's dive into the feedback mechanism. What you expect from a feedback mechanism within OpusFlow, and what are the primary objectives we aim to achieve through this project?

CTO: The need for a feedback mechanism arises from our desire to enhance the user experience (UI/UX) and functionalities of the OpusFlow system. Our main objectives include simplifying the feedback process for end users, especially those in the sustainable sector, while ensuring its effectiveness. We want users to be able to provide feedback on a per-page or per-component level conveniently.

Huy Ton: That makes sense. To delve deeper into user feedback, could you elaborate on the key aspects of the user feedback process that should be simple yet effective, especially for companies in the sustainable sector?

CTO: Certainly. The key aspects of the user feedback process should be user-friendly and straightforward. We want users to have the option of providing quick feedback, such as a simple "happy/no happy" response, but also the flexibility to provide more detailed feedback when they have the time. This simplicity and ease of use are essential for our target audience.

Huy Ton: Thank you for explaining that. Moving on, you mentioned the importance of allowing users to provide feedback on a per-page or per-component level within OpusFlow. Can you explain why this level of granularity is essential for the project?

CTO: Allowing users to provide feedback at such a granular level is crucial because it simplifies the feedback process. For instance, if a specific component, such as a table sorting feature, is not working as expected, we don't want users to go through the hassle of creating a new, global "ticket" and manually describing where it went wrong. We prefer a system where users can identify issues at the component or page level, making it easier for them to provide feedback.

Huy Ton: That's clear. Now, let's discuss the system design and reusability aspect. What are your expectations regarding the design of the feedback system, particularly in terms of abstraction and reusability between different pages and components?

CTO: Our expectation is to create a feedback system that is abstract and can be reused seamlessly between pages and components without the need for custom code per level. The aim is to make it user-friendly and not reliant on technical expertise. For instance, if a button is added to a component or page, it should automatically provide information about where an issue occurred, making it easier for users to provide feedback.

Huy Ton: Thank you for elaborating on that. Now, let's shift our focus to presenting feedback to stakeholders. How do you envision the presentation of collected feedback? What specific insights or metrics are you looking to gain from this feedback?

CTO: We envision a dashboard or system that can present collected feedback in a clear and actionable manner. We are interested in insights that help us prioritize and improve existing functionalities based on user feedback.

Huy Ton: That sounds important for the project. Moving on, you mentioned the potential integration with error tracking tools like Sentry and LogRocket. Can you elaborate on how this integration can benefit OpusFlow and its users?

CTO: Integrating error tracking tools like Sentry and LogRocket can be highly beneficial. It allows us to track and report errors within the system, providing insights into what's going wrong. The challenge is aligning these automated error reports with manual feedback from customers. The data from both sources can provide a clearer view of issues, helping OpusFlow make improvements and prioritize effectively.

Huy Ton: Thank you for explaining that. You mentioned that OpusFlow currently has LogRocket set up but does not actively use it for tracking errors within the system. Could you provide more details about OpusFlow's current setup with LogRocket and its approach to error tracking and feedback handling?

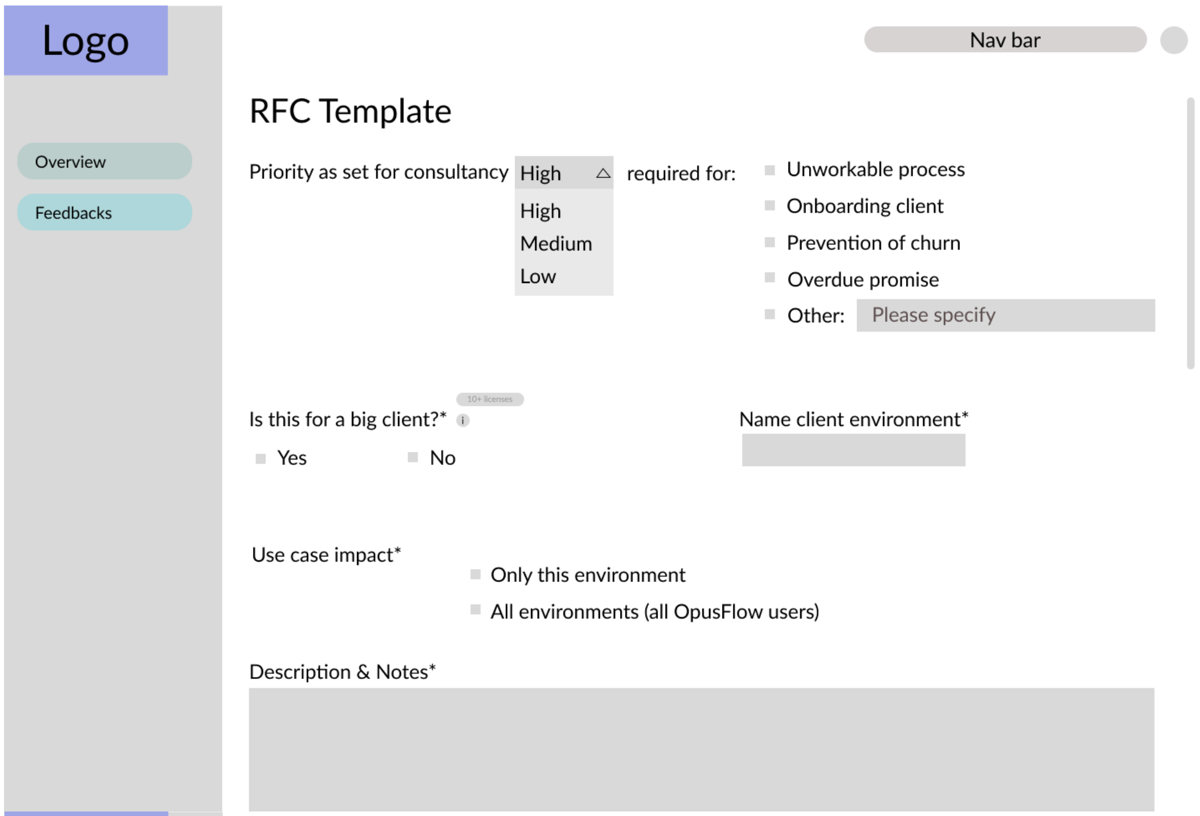
CTO: Currently, we have LogRocket set up, but it's underutilized for tracking errors. Additionally, we do not have a dedicated feedback module within OpusFlow. These are areas we hope to improve through this project.

Huy Ton: I appreciate you sharing those insights. Finally, could you provide any long-term goals or expectations you have for this feedback module within OpusFlow?

CTO: Our long-term goals involve implementing an easy-to-use feedback system for end customers, which can also be used internally to prioritize and enhance existing functionalities. We also aim to create a scalable system that can keep track of error logs and align this data with manual feedback to improve OpusFlow continually.

Huy Ton: Thank you, Joey, for your detailed responses. This information will be invaluable for shaping the project requirements for my graduation project.

Appendix C: Wireframe of RFC Task Template



Logo

Overview

Feedbacks

Nav bar

RFC Template

Priority as set for consultancy High

High
Medium
Low

required for:

- Unworkable process
- Onboarding client
- Prevention of churn
- Overdue promise
- Other: Please specify

Is this for a big client?* i 10+ licenses

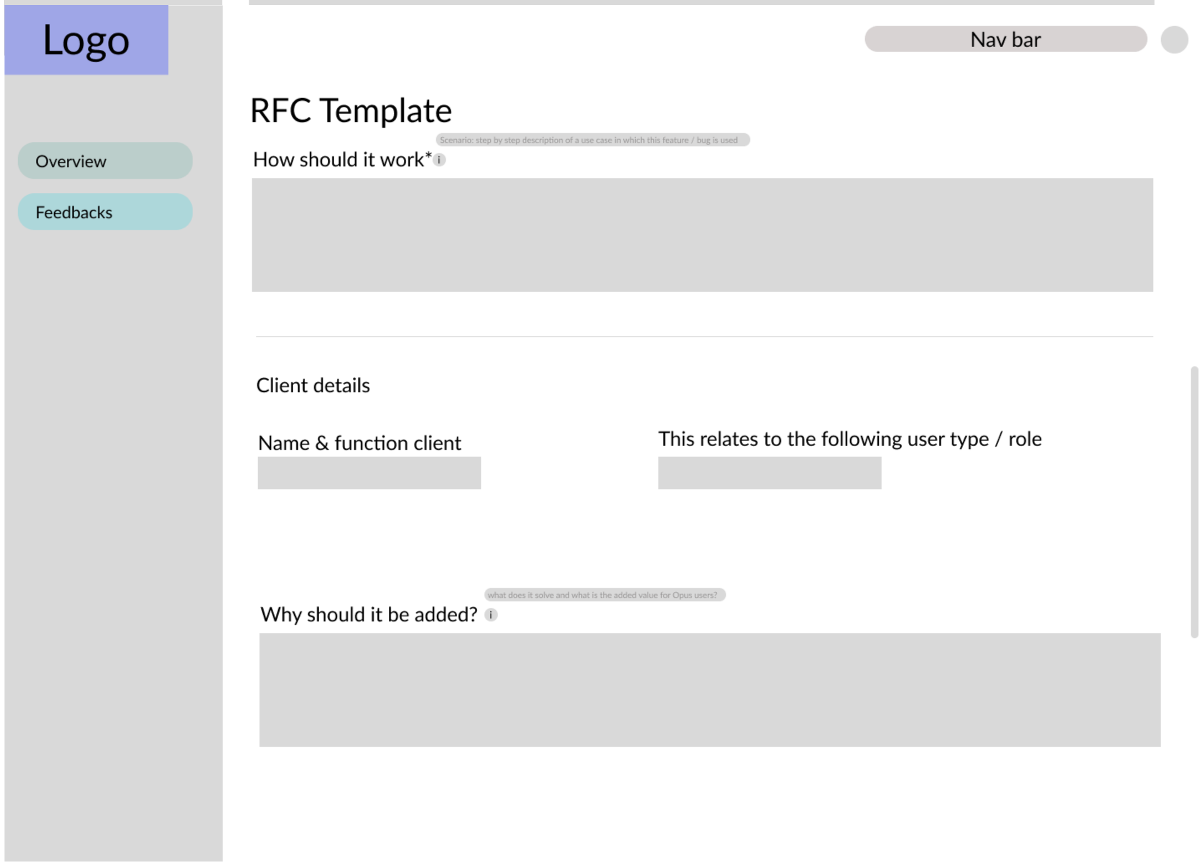
Yes No

Name client environment*

Use case impact*

- Only this environment
- All environments (all OpusFlow users)

Description & Notes*



Logo

Overview

Feedbacks

Nav bar

RFC Template

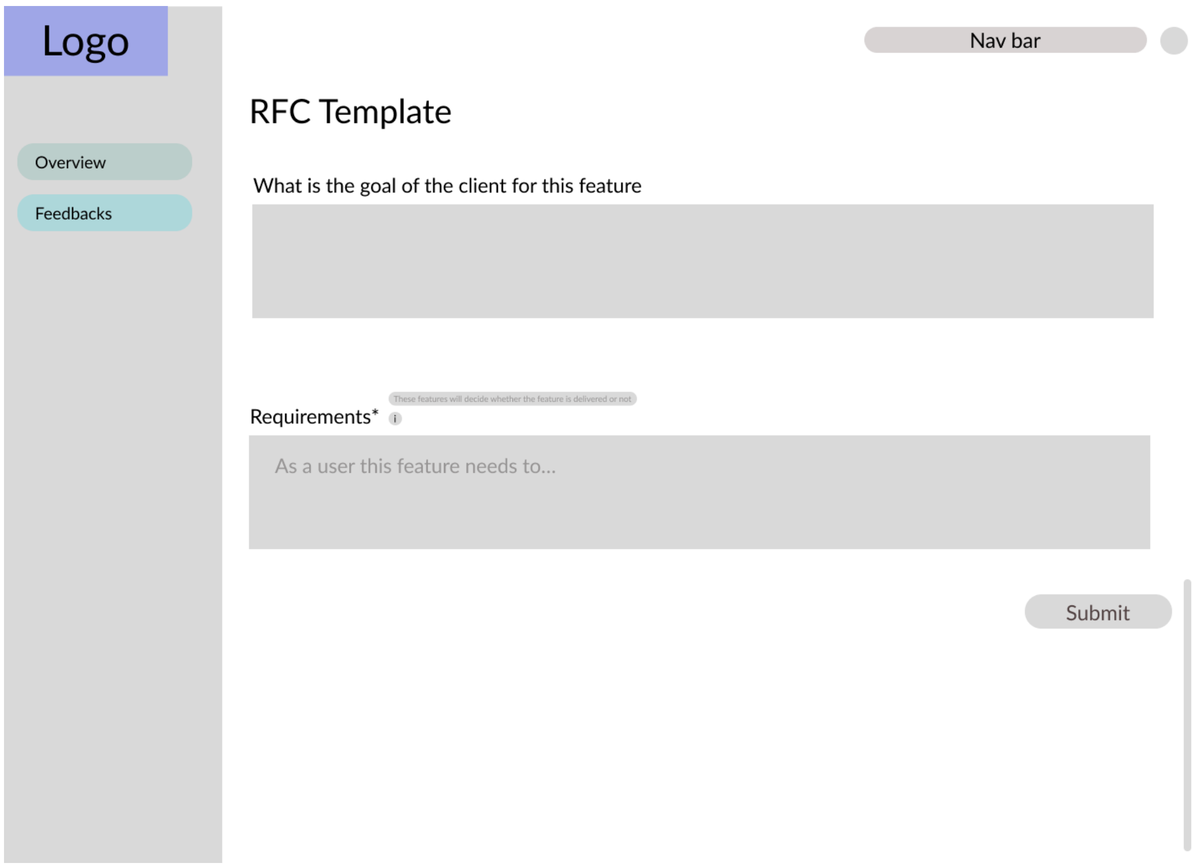
How should it work* i Scenario: step by step description of a use case in which this feature / bug is used

Client details

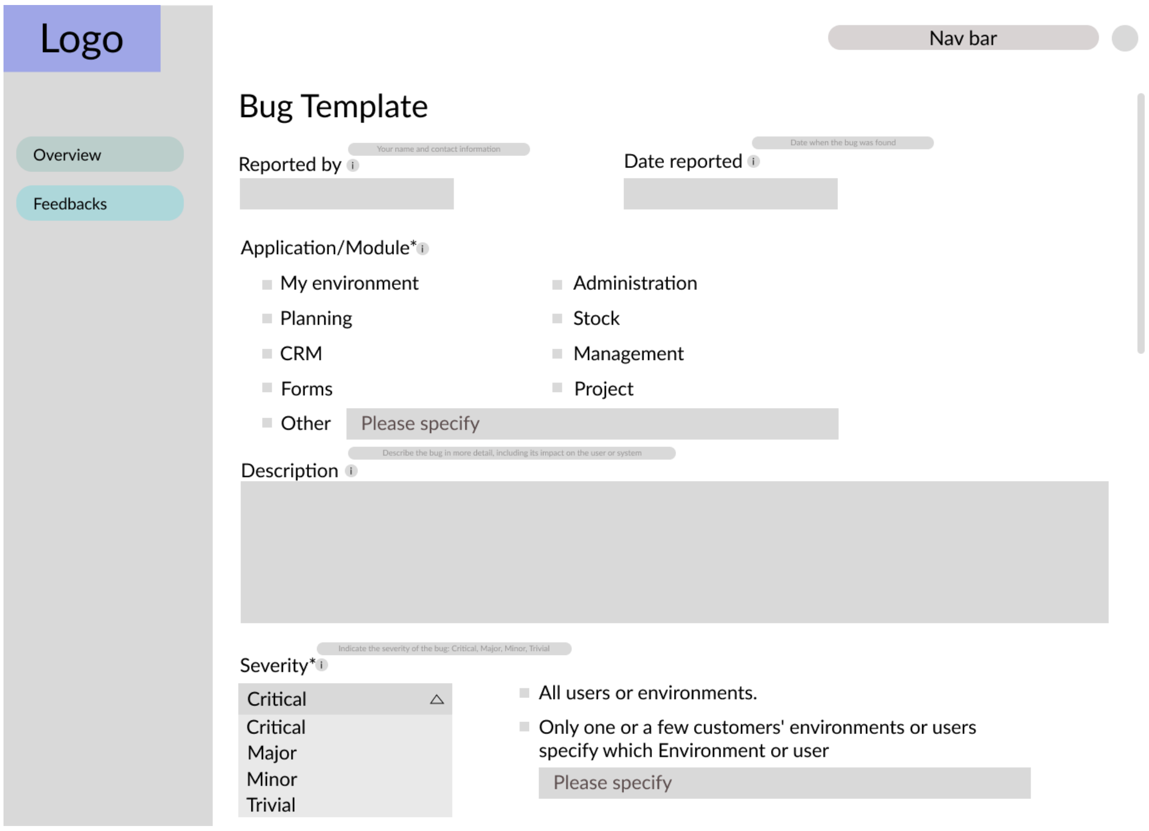
Name & function client

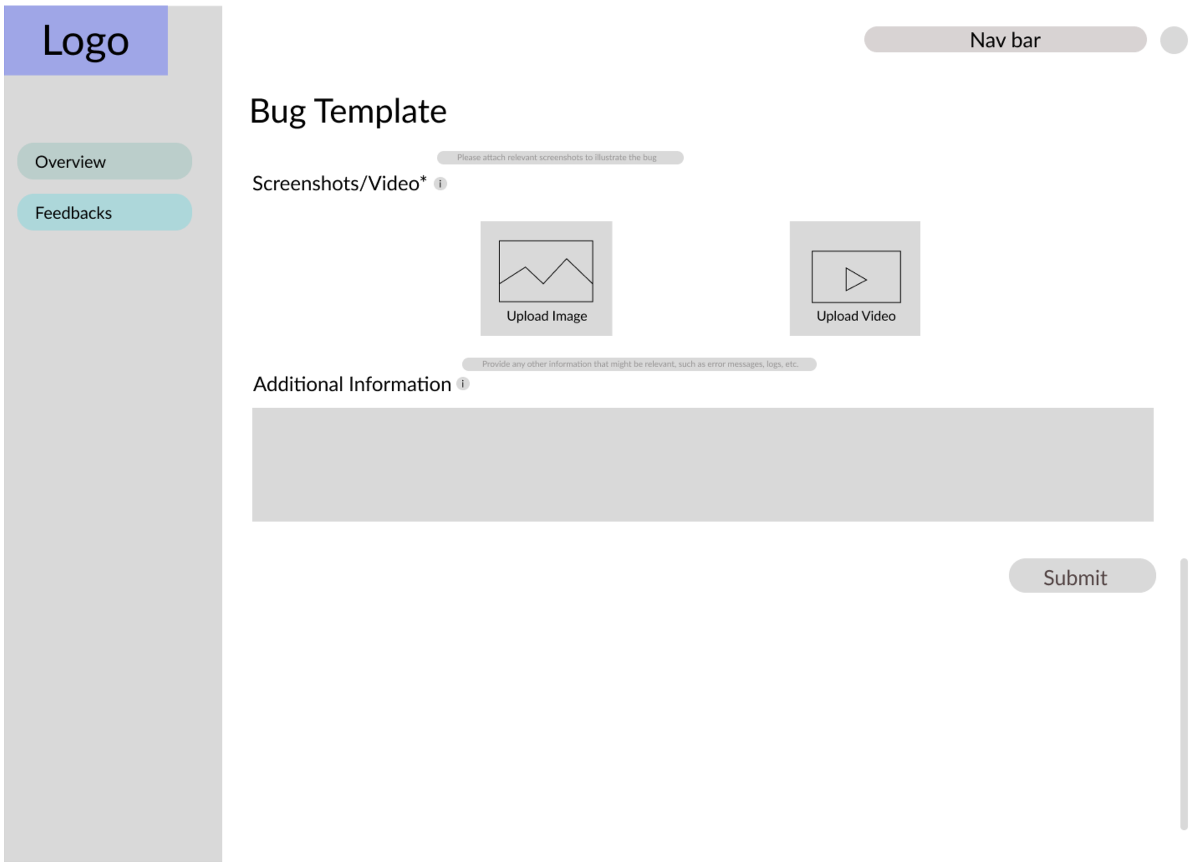
This relates to the following user type / role

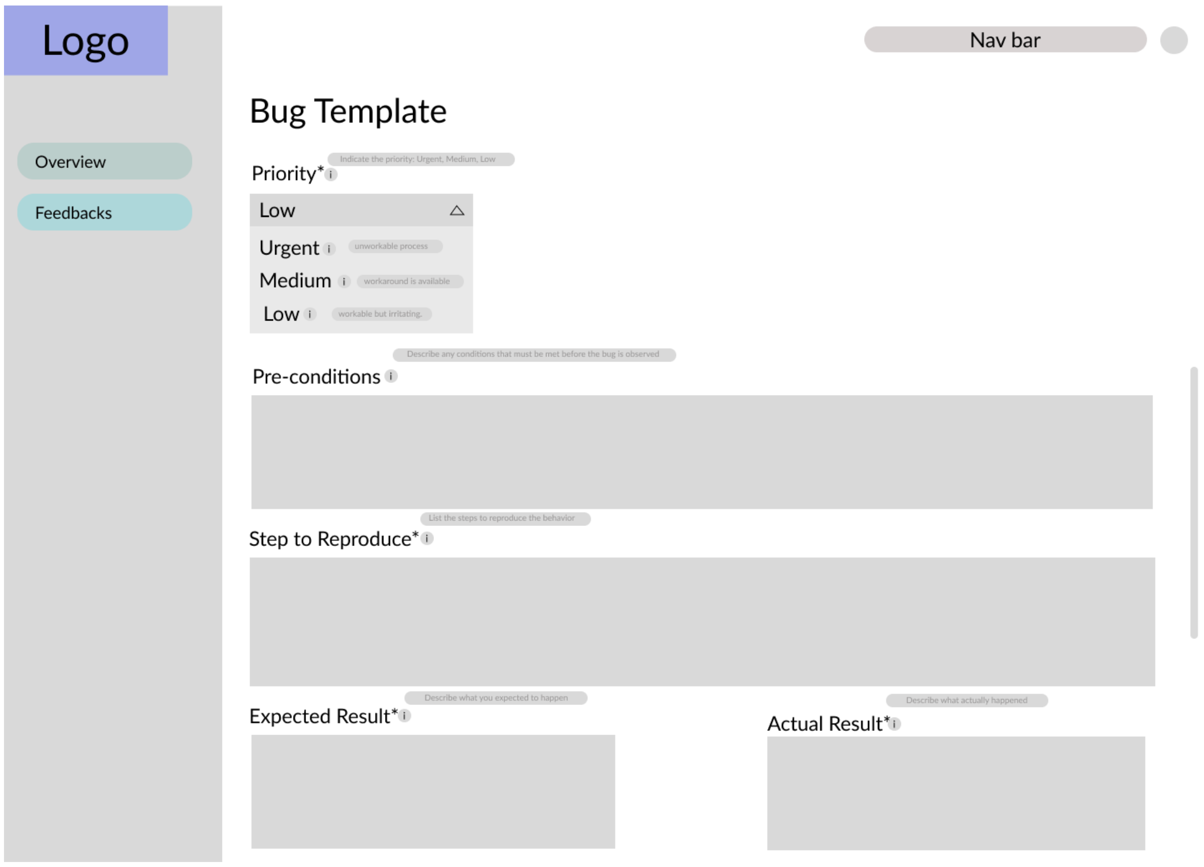
Why should it be added? i what does it solve and what is the added value for Opus users?


Logo
Nav bar
RFC Template
Overview
Feedbacks
What is the goal of the client for this feature
These features will decide whether the feature is delivered or not
Requirements*
As a user this feature needs to...
Submit

Appendix D: Wireframe of Bug Report Task Template


Logo
Nav bar
Overview
Feedbacks
Bug Template
Your name and contact information
Reported by
Date when the bug was found
Date reported
Application/Module*
My environment
Administration
Planning
Stock
CRM
Management
Forms
Project
Other
Please specify
Describe the bug in more detail, including its impact on the user or system
Description
Indicate the severity of the bug: Critical, Major, Minor, Trivial
Severity*
Critical
Critical
Major
Minor
Trivial
All users or environments.
Only one or a few customers' environments or users specify which Environment or user
Please specify


Logo
Nav bar
Overview
Feedbacks
Bug Template
Please attach relevant screenshots to illustrate the bug
Screenshots/Video*
Upload Image
Upload Video
Provide any other information that might be relevant, such as error messages, logs, etc.
Additional Information
Submit



Bug Template

Priority* i — Indicate the priority: Urgent, Medium, Low

Low

- Urgent i — unworkable process
- Medium i — workaround is available
- Low i — workable but irritating.

Pre-conditions i — Describe any conditions that must be met before the bug is observed

Step to Reproduce* i — List the steps to reproduce the behavior

Expected Result* i — Describe what you expected to happen

Actual Result* i — Describe what actually happened

Appendix E: RFC Task Template

1. Priority as set for consultancy *:

- High, required for:
    - Unworkable process
    - Onboarding client
    - Prevention of churn
    - Overdue promise
    - Other:
- Medium, need to have but:
    - Not immediately required
    - Can currently be fixed through other means
    - Feedback on UX- or UI flow
    - Other:
- Low, nice to have:
    - Nice to have

2. Is this for a big client? (10+ licenses) *

3. Name client environment *:

4. Use case impacts *:

- Only this environment

- All environments (all OpusFlow users)

5. Description & notes *:

6. How should it work? (Scenario: step by step description of a use case in which this feature / bug is used) *:

Client details:

1. Name & function client:

2. This relates to the following usertype / role (think of sales, administration etc):

3. Why should it be added? (what does it solve and what is the added value for Opus users?):

4. What is the goal of the client for this feature:

5. Requirements set by the client and the consultant for this feature (these features will decide whether the feature is delivered or not):

- As a user this feature needs to…

Appendix F: Bugfix Task Template

1. Reported by*: (Your name and contact information)

2. Date reported*: (Date when the bug was found)

3. Application/Module*:

- My environment
- Planning
- CRM
- Projects
- Forms
- Administration
- Stock
- Management
- Other: Please specify

4. Description*: (Describe the bug in more detail, including its impact on the user or system)

5. Severity*: (Indicate the severity of the bug: Critical, Major, Minor, Trivial)

- All users or environments.
- Only one or a few customers' environments or users specify which Environment or user: Please specify

6. Priority*: (Indicate the priority: **Urgent**, Medium, Low)

- **Urgent**, unworkable process.
- Medium, workaround is available.
- Low, workable but irritating.

7. Pre-conditions*: (Describe any conditions that must be met before the bug is observed)

8. Steps to Reproduce*:(List the steps to reproduce the behaviour)

9. Expected Result: (Describe what you expected to happen)

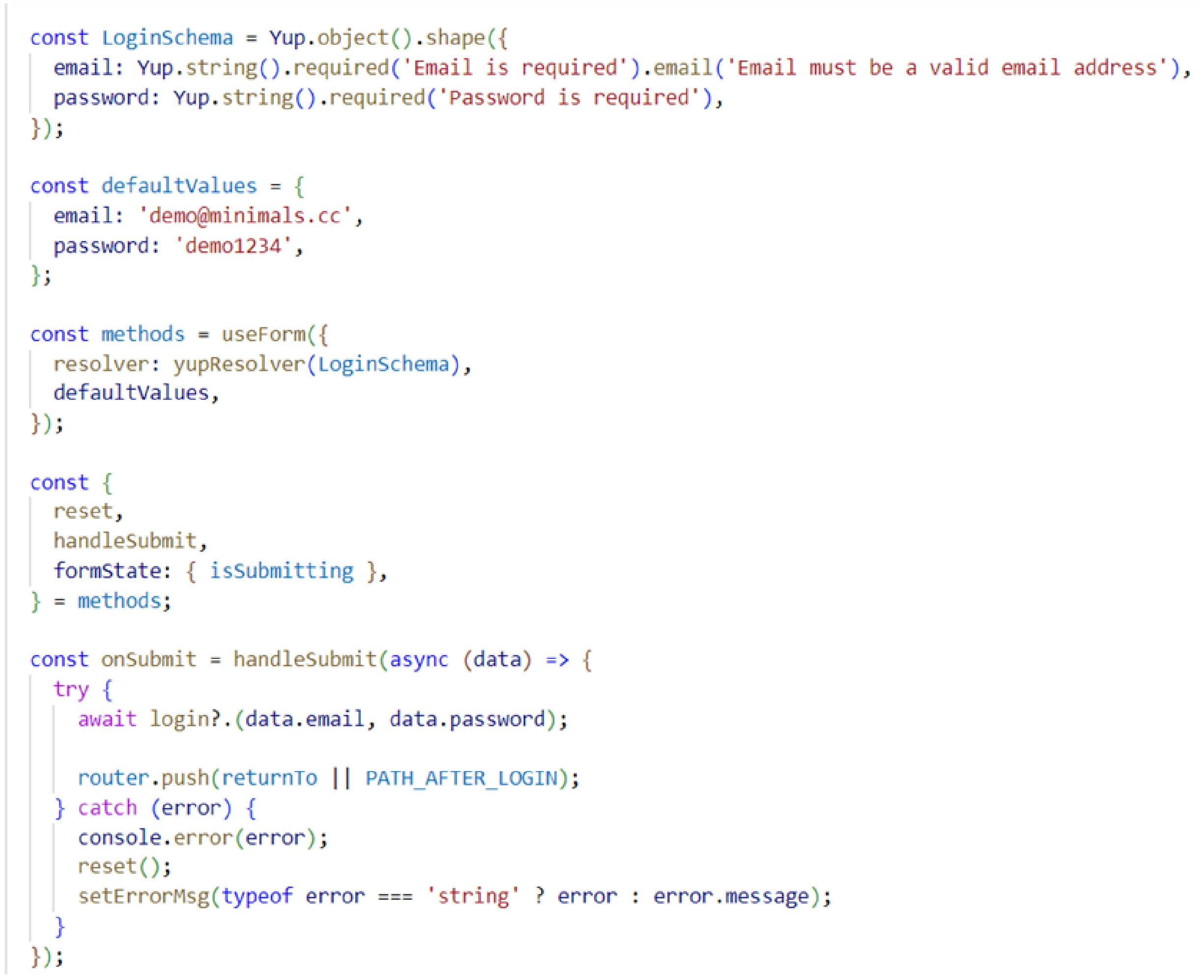
10. Actual Result: (Describe what actually happened)

11. Screenshots/Video*: (Please attach relevant screenshots to illustrate the bug)

12. Additional Information: (Provide any other information that might be relevant, such as error messages, logs, etc.)

Appendix G: Code Snippet for Authentication & Authorization

Authentication



```
const LoginSchema = Yup.object().shape({
  email: Yup.string().required('Email is required').email('Email must be a valid email address'),
  password: Yup.string().required('Password is required'),
});

const defaultValues = {
  email: 'demo@minimals.cc',
  password: 'demo1234',
};

const methods = useForm({
  resolver: yupResolver(LoginSchema),
  defaultValues,
});

const {
  reset,
  handleSubmit,
  formState: { isSubmitting },
} = methods;

const onSubmit = handleSubmit(async (data) => {
  try {
    await login?.(data.email, data.password);

    router.push(returnTo || PATH_AFTER_LOGIN);
  } catch (error) {
    console.error(error);
    reset();
    setErrorMsg(typeof error === 'string' ? error : error.message);
  }
});
```

Here is the break down the key components:

1. **LoginSchema (Yup Schema):**
   - A Yup schema (**LoginSchema**) is defined to validate the structure of the login form data.
   - The schema enforces rules for the 'email' field, requiring it to be a valid email address, and the 'password' field, ensuring it is not empty.
2. **Default Values:**
   - Default values are set for the email and password fields to pre-fill the login form. These values serve as initial placeholders, providing a convenient user experience.
3. **useForm Hook:**
   - The **useForm** hook from **react-hook-form** is employed to manage the form state, validation, and submission logic.

- The **yupResolver** is used to integrate the Yup schema (**LoginSchema**) with the form, enforcing the defined validation rules.

4. **Form Methods:**
   - Various form methods are extracted from the **useForm** hook, including **reset**, **handleSubmit**, and access to the form state (**isSubmitting**).

5. **OnSubmit Function:**
   - The **onSubmit** function is invoked when the user submits the login form.
   - Inside the **onSubmit** function, the user's login credentials (email and password) are passed to the **login** function, which presumably handles the authentication.
   - If the login is successful, the user is redirected to the specified route (**returnTo** or a default route after login).
   - In case of an error during login, the **reset** method is called to clear the form, and an error message is set to be displayed to the user. Any caught error is logged to the console.

Authorization

Here is the break down the key components:

1. **Router and Authentication:**
   - Utilizes **useRouter** and **useAuthContext** hooks for routing and authentication information.

2. **State:**
   - Manages a **checked** state to track authentication verification.

3. **Check Function:**
   - Checks authentication status.
   - Redirects unauthenticated users to the login page with return URL.
   - Sets **checked** to true for authenticated users.

4. **Effect Hook:**
   - Executes the check function on component mount.

5. **Return:**
   - Renders **children** if authentication check is complete; otherwise, renders nothing.

In essence, **AuthGuard** ensures protected content is only accessible to authenticated users by handling authentication checks and redirection logic

```
export default function AuthGuard({ children }: Props) {
  const router = useRouter();

  const { authenticated, method } = useAuthContext();

  const [checked, setChecked] = useState(false);

  const check = useCallback(() => {
    if (!authenticated) {
      const searchParams = new URLSearchParams({
        returnTo: window.location.pathname,
      }).toString();

      const loginPath = loginPaths[method];

      const href = `${loginPath}?${searchParams}`;

      router.replace(href);
    } else {
      setChecked(true);
    }
  }, [authenticated, method, router]);

  useEffect(() => {
    check();
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, []);

  if (!checked) {
    return null;
  }

  return <>{children}</>;
}
```

Appendix H: Code Snippets for Feedback Creation Function

Here is the break down of main components of the submission of feedback in SnapFlow Feedback:

Feedback Schema (Yup Schema):

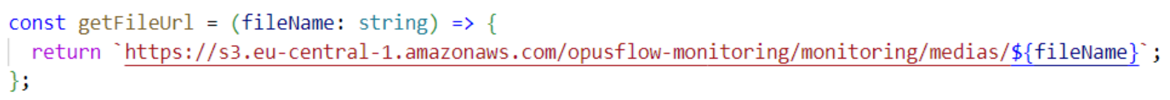
- A Yup schema (**FeedbackSchema**) is defined to validate the structure of the feedback form data.
- Validation rules include making certain fields required, such as 'type,' 'element,' 'issue,' and 'image,' with customized error messages.

```
const FeedbackSchema = object().shape({
  type: string().required("This field is required"),
  element: string().required("This field is required"),
  description: string(),
  issue: string().required("This field is required"),
  image: string().required("This field is required"),
  videos: array().of(mixed()),
});
```

File Upload Functions:

- Functions like **getPresignedUrl** and **uploadFileToS3** handle the process of obtaining pre-signed URLs for file uploads to AWS S3 and uploading files to the specified URLs. The details of obtaining a pre-signed URL and uploading a file to the AWS S3 bucket are explained in the AWS S3 Bucket Configuration.
- **getFileUrl** generates the public URL for a given file name in the S3 bucket.

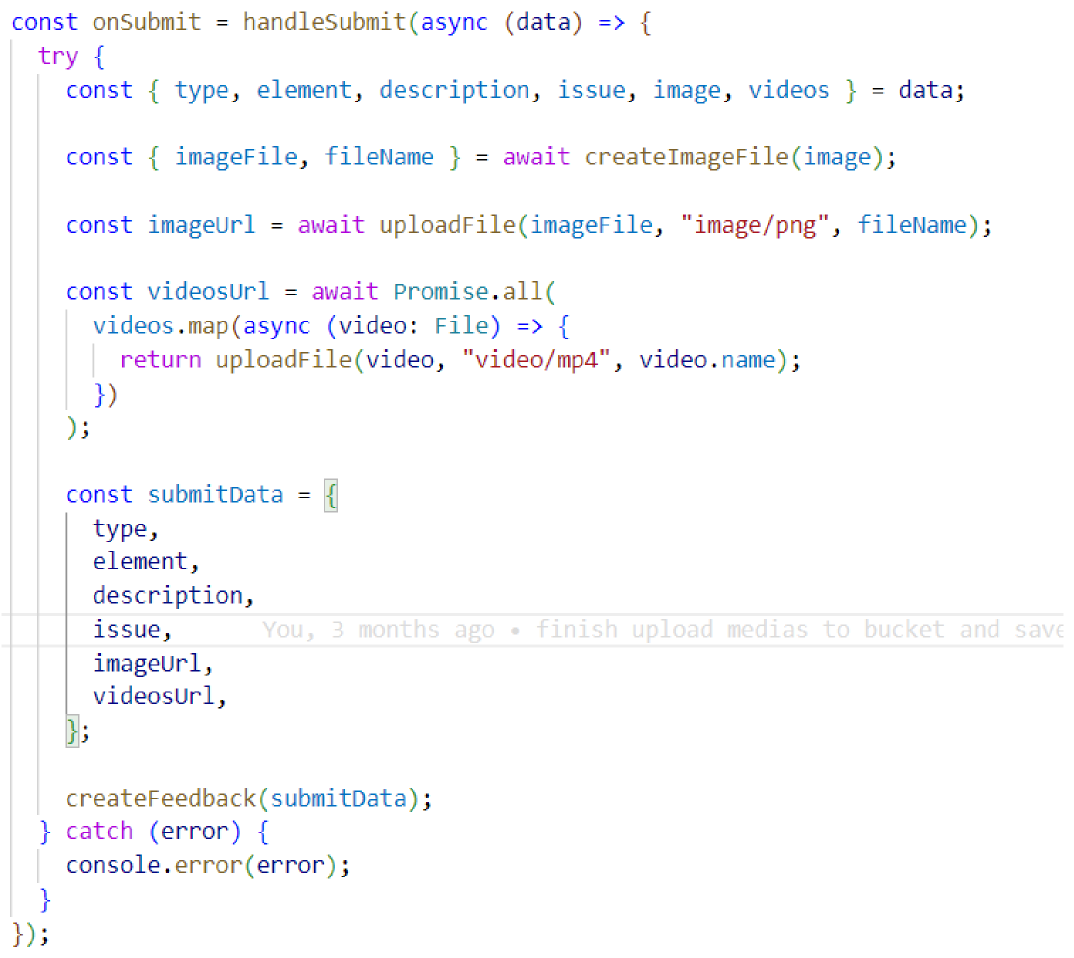
```
const getPresignedUrl = async (fileType: string, fileName: string) => {
  const response = await fetch("/api/feedback/upload", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ file_type: fileType, file_name: fileName }),
  });
  const { url } = await response.json();
  return url;
};

const uploadFileToS3 = async (url: string, file: File, fileType: string) => {
  const response = await fetch(url, {
    method: "PUT",
    body: file,
    headers: { "Content-Type": fileType },
  });
  return response.ok;
};
```



```
const getFileUrl = (fileName: string) => {
  return `https://s3.eu-central-1.amazonaws.com/opusflow-monitoring/monitoring/medias/${fileName}`;
};
```

Feedback Submission Logic:

- The **onSubmit** function is invoked when the user submits the feedback form.
- Within this function, an image file is created from the provided image URL using **createImageFile**.
- The image file is then uploaded to AWS S3, and the resulting image URL is obtained.
- Similarly, each video file in the 'videos' array is uploaded, and an array of video URLs is generated.
- The final data, including type, element, description, issue, image URL, and video URLs, is submitted to the Apollo Client to mutate (create) feedback data using the **CREATE_FEEDBACK** mutation.



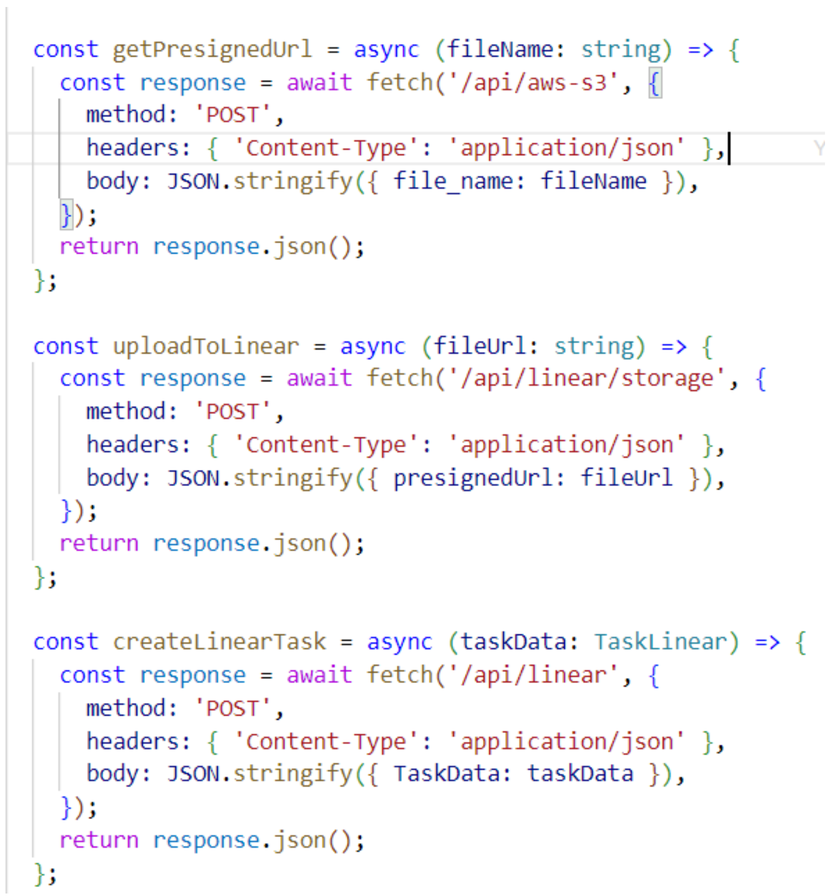
```
const onSubmit = handleSubmit(async (data) => {
  try {
    const { type, element, description, issue, image, videos } = data;

    const { imageFile, fileName } = await createImageFile(image);

    const imageUrl = await uploadFile(imageFile, "image/png", fileName);

    const videosUrl = await Promise.all(
      videos.map(async (video: File) => {
        return uploadFile(video, "video/mp4", video.name);
      })
    );

    const submitData = {
      type,
      element,
      description,
      issue,        You, 3 months ago • finish upload medias to bucket and save
      imageUrl,
      videosUrl,
    };

    createFeedback(submitData);
  } catch (error) {
    console.error(error);
  }
});
```

Appendix I: Code Snippets for Linear Task Creation Function

Here is the break down of main components of the creation of linear task in OpusFlow InsightHub:

File Upload Functions:

- **getPresignedUrl**: This function sends a request to the '/api/aws-s3' endpoint to obtain a pre-signed URL for a given file name.
- **uploadToLinear**: It sends a request to the '/api/linear/storage' endpoint, providing the obtained pre-signed URL, to upload a file to Linear.
- **createLinearTask**: This function sends a request to the '/api/linear' endpoint to create a Linear task, including task data.

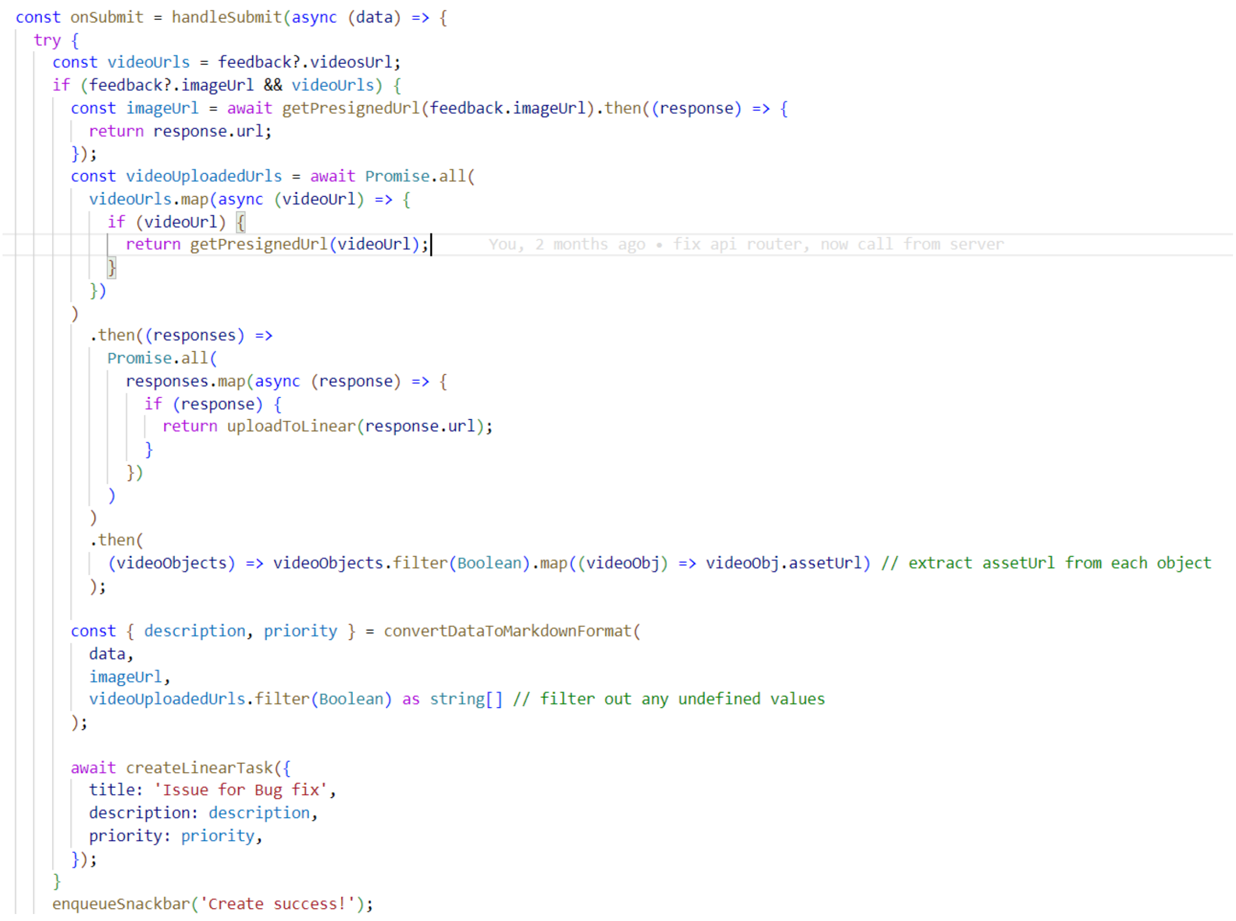


```
const getPresignedUrl = async (fileName: string) => {
  const response = await fetch('/api/aws-s3', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ file_name: fileName }),
  });
  return response.json();
};

const uploadToLinear = async (fileUrl: string) => {
  const response = await fetch('/api/linear/storage', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ presignedUrl: fileUrl }),
  });
  return response.json();
};

const createLinearTask = async (taskData: TaskLinear) => {
  const response = await fetch('/api/linear', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ TaskData: taskData }),
  });
  return response.json();
};
```

Form Submission Logic (onSubmit function):

- Retrieves video URLs from feedback data and checks for the existence of both an image URL (**feedback.imageUrl**) and valid video URLs.
- Uses **getPresignedUrl** to obtain a pre-signed URL for the image and each video in the 'videosUrl' array.
- Utilizes **uploadToLinear** to upload videos to Linear and extract their respective asset URLs.
- Converts form data into Markdown format using **convertDataToMarkdownFormat**.

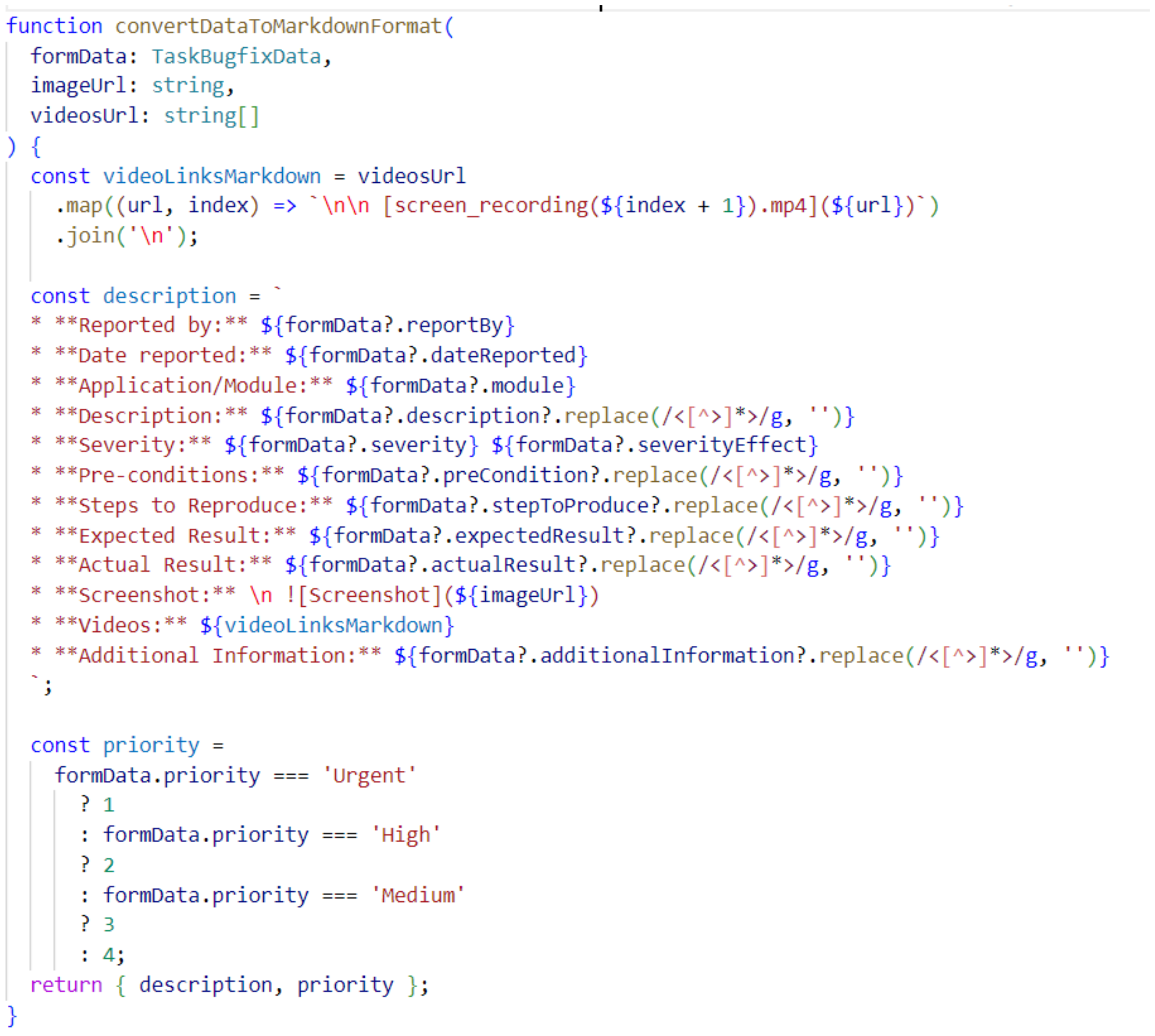
- Calls **createLinearTask** with the formatted data to create a Linear task.
- Displays a success message using **enqueueSnackbar** upon successful task creation.



```
const onSubmit = handleSubmit(async (data) => {
  try {
    const videoUrls = feedback?.videosUrl;
    if (feedback?.imageUrl && videoUrls) {
      const imageUrl = await getPresignedUrl(feedback.imageUrl).then((response) => {
        return response.url;
      });
      const videoUploadedUrls = await Promise.all(
        videoUrls.map(async (videoUrl) => {
          if (videoUrl) {
            return getPresignedUrl(videoUrl);      You, 2 months ago • fix api router, now call from server
          }
        })
      )
        .then((responses) =>
          Promise.all(
            responses.map(async (response) => {
              if (response) {
                return uploadToLinear(response.url);
              }
            })
          )
        )
        .then(
          (videoObjects) => videoObjects.filter(Boolean).map((videoObj) => videoObj.assetUrl) // extract assetUrl from each object
        );

      const { description, priority } = convertDataToMarkdownFormat(
        data,
        imageUrl,
        videoUploadedUrls.filter(Boolean) as string[] // filter out any undefined values
      );

      await createLinearTask({
        title: 'Issue for Bug fix',
        description: description,
        priority: priority,
      });
    }
    enqueueSnackbar('Create success!');
```

Markdown Format Conversion (convertDataToMarkdownFormat function):

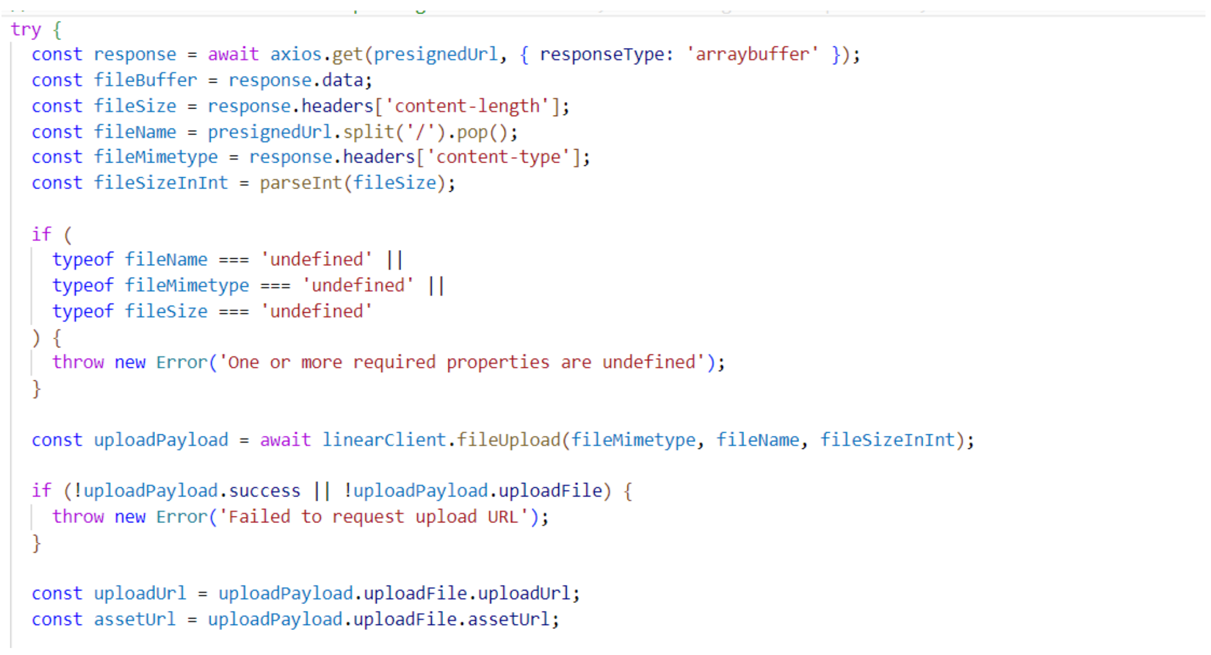
- Takes form data (**TaskBugfixData**), image URL, and video URLs as input.
- Generates Markdown-formatted description including reported by, date reported, module, description, severity, pre-conditions, steps to reproduce, expected and actual results, screenshot, videos, and additional information.
- Translates priority levels ('Urgent', 'High', 'Medium', 'Low') into numerical values.



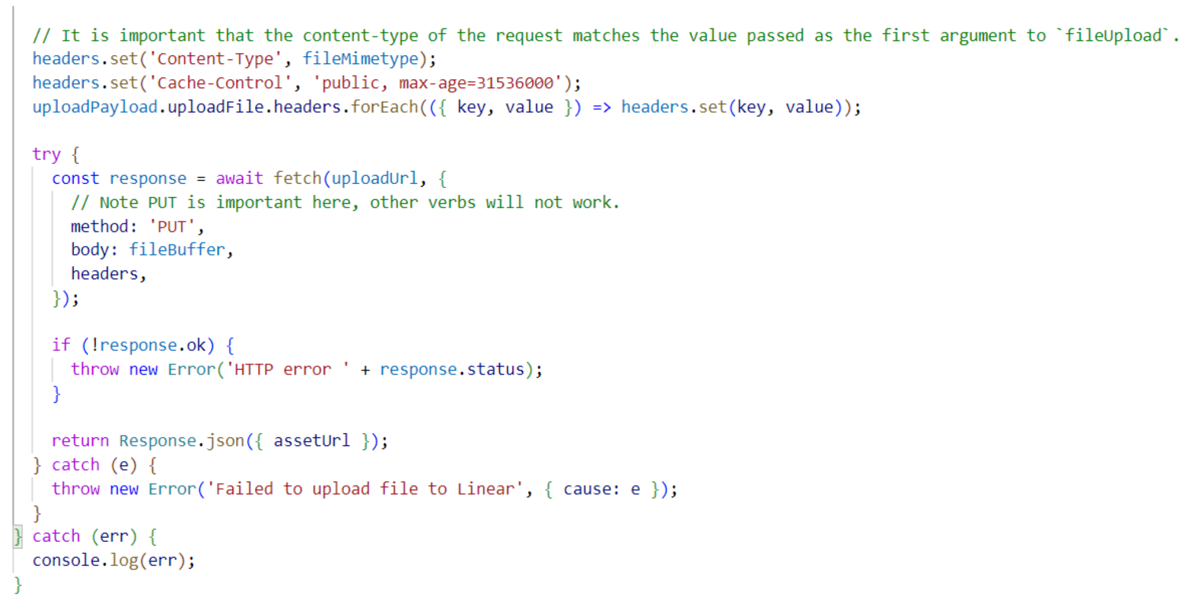
```
function convertDataToMarkdownFormat(
  formData: TaskBugfixData,
  imageUrl: string,
  videosUrl: string[]
) {
  const videoLinksMarkdown = videosUrl
    .map((url, index) => `\n\n [screen_recording(${index + 1}).mp4](${url})`)
    .join('\n');

  const description = `
  * **Reported by:** ${formData?.reportBy}
  * **Date reported:** ${formData?.dateReported}
  * **Application/Module:** ${formData?.module}
  * **Description:** ${formData?.description?.replace(/<[^>]*>/g, '')}
  * **Severity:** ${formData?.severity} ${formData?.severityEffect}
  * **Pre-conditions:** ${formData?.preCondition?.replace(/<[^>]*>/g, '')}
  * **Steps to Reproduce:** ${formData?.stepToProduce?.replace(/<[^>]*>/g, '')}
  * **Expected Result:** ${formData?.expectedResult?.replace(/<[^>]*>/g, '')}
  * **Actual Result:** ${formData?.actualResult?.replace(/<[^>]*>/g, '')}
  * **Screenshot:** \n ![Screenshot](${imageUrl})
  * **Videos:** ${videoLinksMarkdown}
  * **Additional Information:** ${formData?.additionalInformation?.replace(/<[^>]*>/g, '')}
  `;

  const priority =
    formData.priority === 'Urgent'
      ? 1
      : formData.priority === 'High'
      ? 2
      : formData.priority === 'Medium'
      ? 3
      : 4;
  return { description, priority };
}
```

File Upload to Linear API (POST function for file upload to Linear):

- Receives a pre-signed URL from the request body.
- Downloads the file from the S3 pre-signed URL using Axios.
- Requests an upload URL from the Linear SDK.
- Uploads the file to the specified Linear upload URL using a PUT request and returns the asset URL.



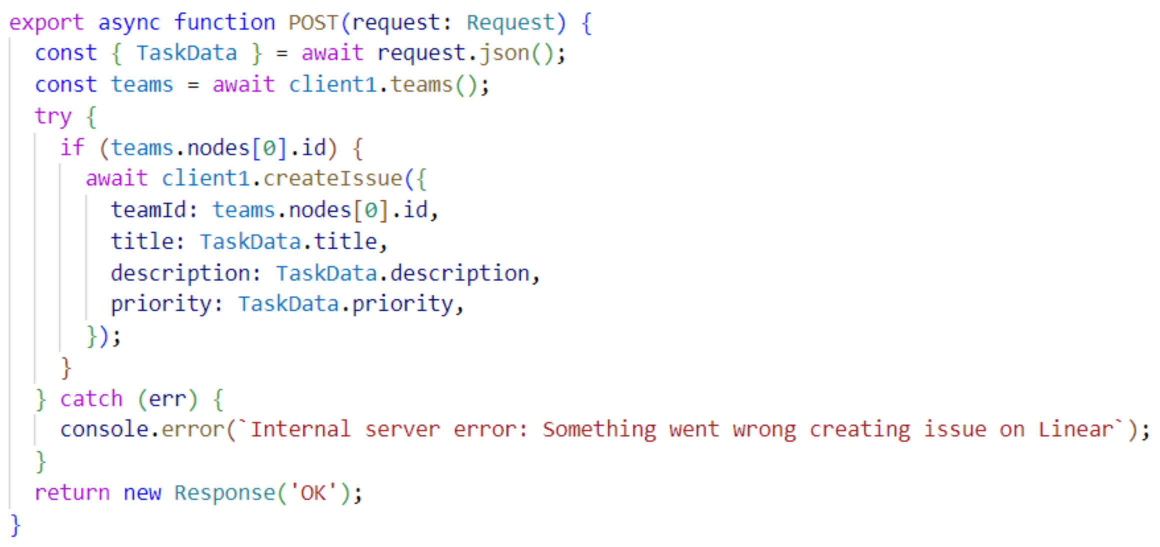
```
try {
  const response = await axios.get(presignedUrl, { responseType: 'arraybuffer' });
  const fileBuffer = response.data;
  const fileSize = response.headers['content-length'];
  const fileName = presignedUrl.split('/').pop();
  const fileMimetype = response.headers['content-type'];
  const fileSizeInInt = parseInt(fileSize);

  if (
    typeof fileName === 'undefined' ||
    typeof fileMimetype === 'undefined' ||
    typeof fileSize === 'undefined'
  ) {
    throw new Error('One or more required properties are undefined');
  }

  const uploadPayload = await linearClient.fileUpload(fileMimetype, fileName, fileSizeInInt);

  if (!uploadPayload.success || !uploadPayload.uploadFile) {
    throw new Error('Failed to request upload URL');
  }

  const uploadUrl = uploadPayload.uploadFile.uploadUrl;
  const assetUrl = uploadPayload.uploadFile.assetUrl;
```



```
  // It is important that the content-type of the request matches the value passed as the first argument to `fileUpload`.
  headers.set('Content-Type', fileMimetype);
  headers.set('Cache-Control', 'public, max-age=31536000');
  uploadPayload.uploadFile.headers.forEach(({ key, value }) => headers.set(key, value));

  try {
    const response = await fetch(uploadUrl, {
      // Note PUT is important here, other verbs will not work.
      method: 'PUT',
      body: fileBuffer,
      headers,
    });

    if (!response.ok) {
      throw new Error('HTTP error ' + response.status);
    }

    return Response.json({ assetUrl });
  } catch (e) {
    throw new Error('Failed to upload file to Linear', { cause: e });
  }
} catch (err) {
  console.log(err);
}
```

Create Task API (POST function for creating a task on Linear):

- Receives task data from the request body.
- Retrieves the team ID from the Linear SDK.
- Creates a new issue on Linear using the provided task title, description, and priority within the specified team.



```
export async function POST(request: Request) {
  const { TaskData } = await request.json();
  const teams = await client1.teams();
  try {
    if (teams.nodes[0].id) {
      await client1.createIssue({
        teamId: teams.nodes[0].id,
        title: TaskData.title,
        description: TaskData.description,
        priority: TaskData.priority,
      });
    }
  } catch (err) {
    console.error(`Internal server error: Something went wrong creating issue on Linear`);
  }
  return new Response('OK');
}
```

Appendix K: Testing Details

This section offers detailed insights into the tests documented in [the testing section of the report](). Automated testing will be executed using the Jest tool, and manual testing will be personally conducted. All tests listed below are performed with the following standardized setup:

- **Hardware:**
  - Dell XPS 13
  - Intel Core i7
  - 16GB RAM
- **Software:**
  - Operating System: Windows 10
  - Browser: Google Chrome (Version 120.0.6099.225)
  - Snipping Tool: The Snipping Tool is a built-in screen capture utility in Windows operating systems. It allows users to take screenshots of all or part of their screen, providing flexibility in capturing specific areas, full screens, or individual windows. In this testing scenario, the Snipping Tool is utilized to capture selected components and pages for later comparison.
- **Test Data:**
  - Sample user accounts provided by ProfitFlow B.V.
  - Environment keys for project setup provided by ProfitFlow B.V.
- **Pre-Condition:** Successfully log in to the OpusFlow system.

The subsequent details outline the specifics of each test:

Unit Testing

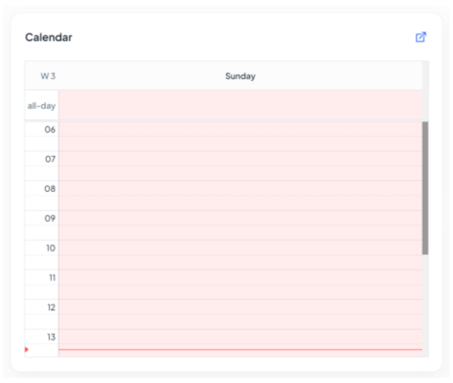
Test 1: UT1- Verify the application's ability to capture screenshots.

Manual Test

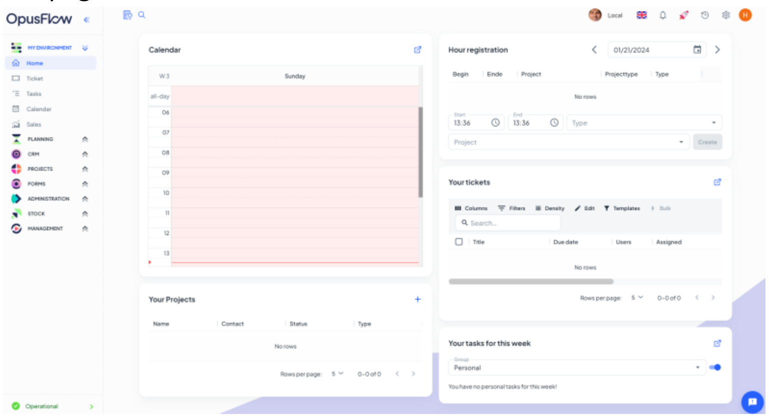
- **Description:** This test the function screenshot function of the SnapFlow Feedback. The test to make sure that the seleted component/page are correctly take the snapshot.
- **Test Steps:**
  - Navigate to the index page of OpusFlow.
  - Click on the feedback button located in the right corner.
  - Select the "Component" option to specify the element for feedback in step 1 of the feedback form.
  - In step 2 of the feedback form, click on the captured image to view the screenshot in lightbox mode.

- Save the image and use the Snipping Tool to capture the previously selected component for comparison.
- Click on the reset button of the form and choose the "Page" option.
- Repeat the steps to open the screenshot in lightbox mode, save the image, and use the Snipping Tool to capture the selected component for comparison.

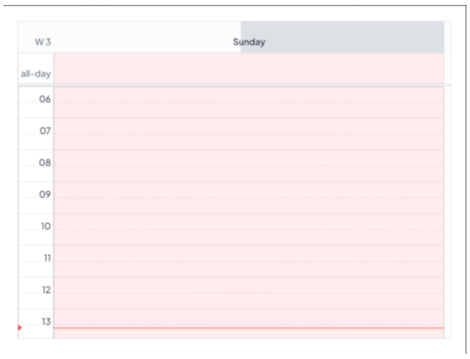
- **Expected Result:** Screenshot capture by using Snipping Tool
  - For component screenshot



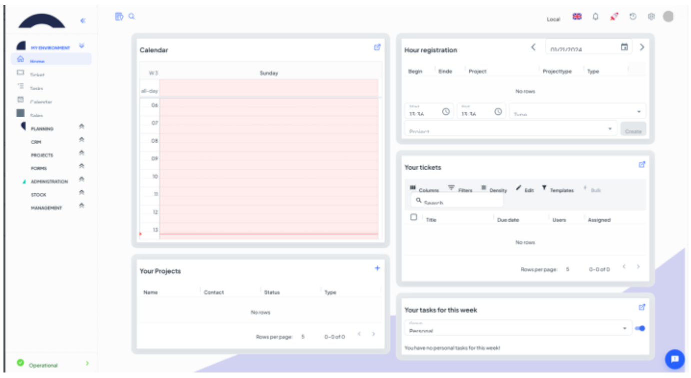
  - For page screenshot



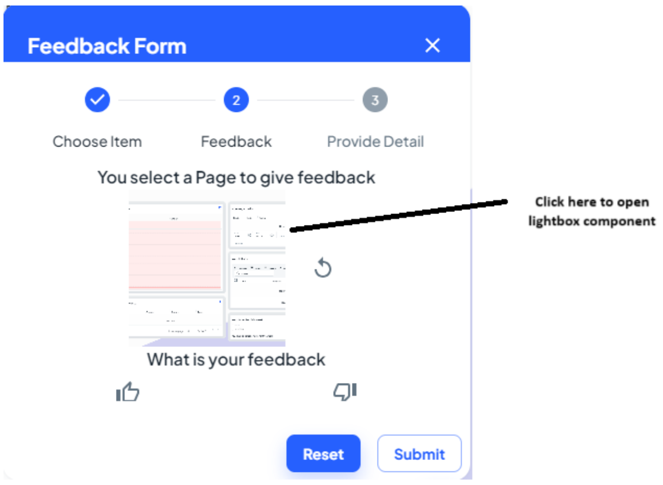
- **Actual Result:** Screenshot capture by SnapFlow Feedback
  - For component screenshot



- For page screenshot



- **Status:** Pass. The screenshot function for both page and component works as expected. A slight difference is observed when comparing the screenshots taken by SnapFlow Feedback and the Snipping Tool. This discrepancy is attributed to the limitations of the html2Canvas library.
- **Notes:** Here is the how to open the lightbox to display the image in lightbox mode



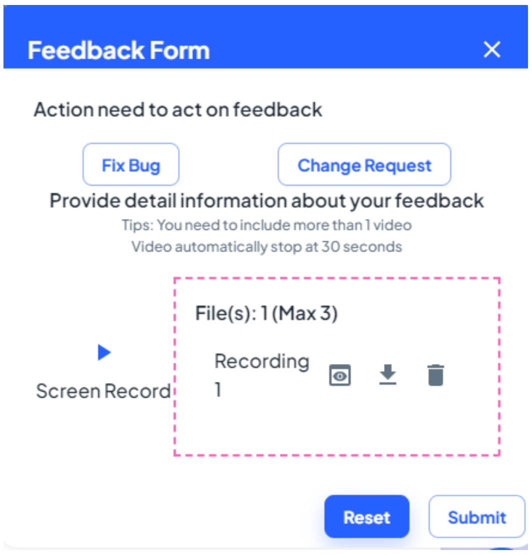
Test 2: UT2- Validate the functionality of screen recording.

Manual Test

- **Description:** This test evaluates the screen recording functionality of the SnapFlow Feedback.
- **Test Steps:**
  - Navigate to any page of the OpusFlow System.
  - Click on the feedback button located in the right corner.
  - Select the "Component" option and choose the negative feedback option in step 2.
  - Choose the option "Fix Bug."

- Scroll down in the form to find the "record" option.
- Press the screen record button and select the current screen to record.
- Press the screen record button again and wait for 30 seconds until the recording automatically stops.

- **Expected Result:**
  - The recording appears after it finishes.
  - The functionality to watch and download the recording works as expected.
- **Actual Result:**
  - Screenshot after finishing the first recording.



Feedback Form

Action need to act on feedback

Fix Bug   Change Request

Provide detail information about your feedback

Tips: You need to include more than 1 video

Video automatically stop at 30 seconds

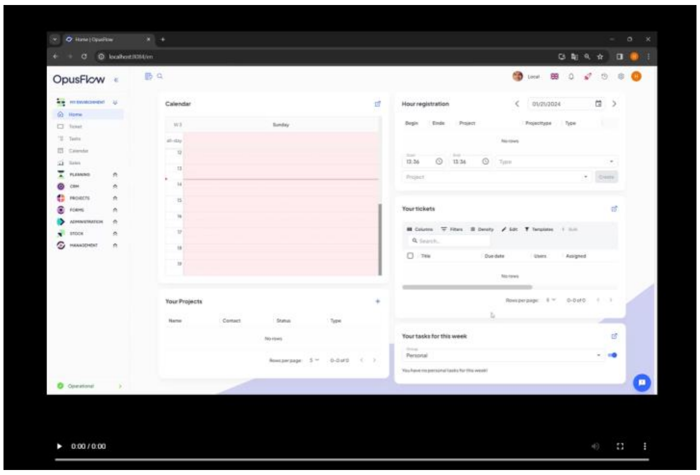
Screen Record

File(s): 1 (Max 3)

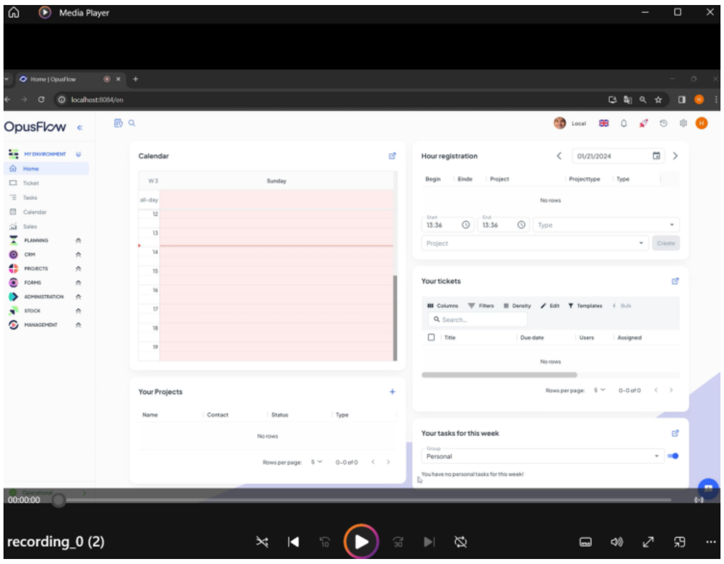
Recording 1

Reset   Submit

  - Screenshot after clicking on the watch button.



  - Screenshot after clicking on the download button and confirming the download



- **Status:** Pass. The function screen record work normally with supported functions like watch the record and download the record also work normally.
- **Notes:** None

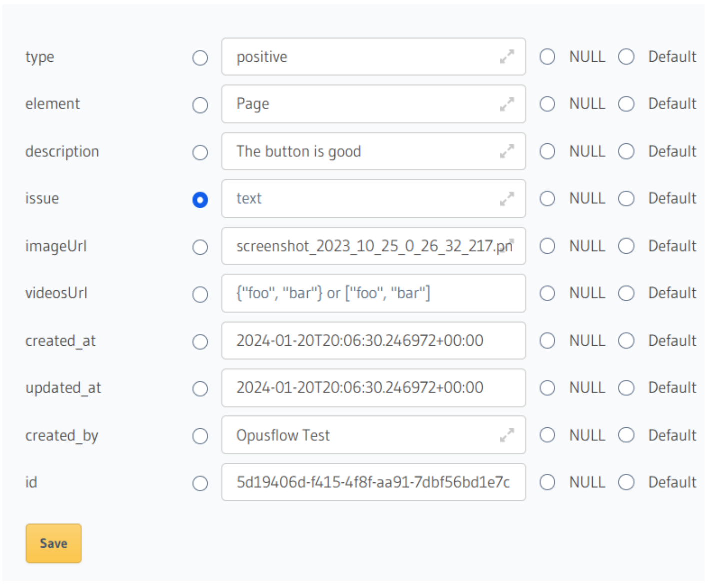
Test 3: UT3- Test the creation of feedback entries.

Manual Test

- **Description:** This test validates the functionality of creating feedback on SnapFlow Feedback.
- **Test Steps:**
  - Click on the feedback button located in the right corner.
  - Choose the "Page" option to provide feedback.
  - Choose the option to give positive feedback.
  - Fill in the required information and submit the form.
  - Click on the feedback button again.
  - Repeat the process of providing feedback with the "Page" option, but this time choose the negative feedback option.
  - Fill in the required information, record the screen, and submit the form.
- **Expected Result:**
  - Two new feedback entries are registered in the database.
  - Both feedback entries are related to the page, with one being positive and the other negative.
  - The timestamp of the created date for positive feedback should be on date 20/01/2024, and for negative feedback, it should be on 21/01/2024.

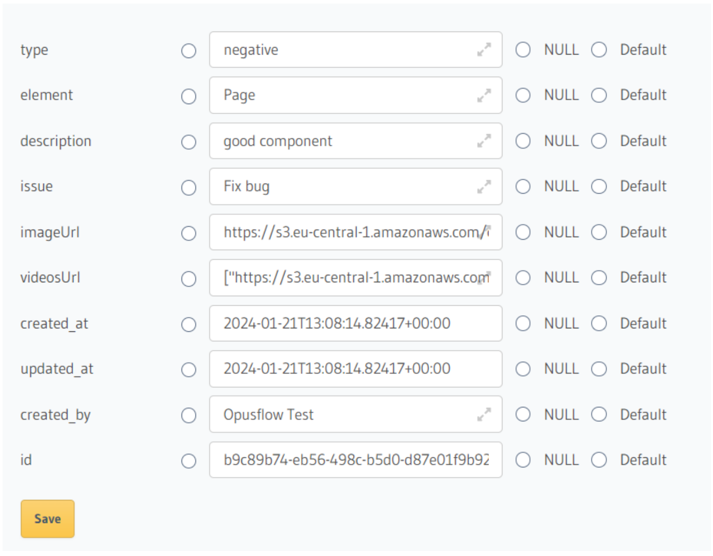
- The required information such as type, element, description, imageUrl should be displayed.

- **Actual Result:**
  - Screenshot for the positive feedback record on date 20/01/2024.



type	positive	NULL	Default
element	Page	NULL	Default
description	The button is good	NULL	Default
issue	text	NULL	Default
imageUrl	screenshot_2023_10_25_0_26_32_217.pn	NULL	Default
videosUrl	{"foo", "bar"} or ["foo", "bar"]	NULL	Default
created_at	2024-01-20T20:06:30.246972+00:00	NULL	Default
updated_at	2024-01-20T20:06:30.246972+00:00	NULL	Default
created_by	Opusflow Test	NULL	Default
id	5d19406d-f415-4f8f-aa91-7dbf56bd1e7c	NULL	Default

  - Screenshot for the positive feedback record on date 21/01/2024.



type	negative	NULL	Default
element	Page	NULL	Default
description	good component	NULL	Default
issue	Fix bug	NULL	Default
imageUrl	https://s3.eu-central-1.amazonaws.com/	NULL	Default
videosUrl	["https://s3.eu-central-1.amazonaws.com	NULL	Default
created_at	2024-01-21T13:08:14.82417+00:00	NULL	Default
updated_at	2024-01-21T13:08:14.82417+00:00	NULL	Default
created_by	Opusflow Test	NULL	Default
id	b9c89b74-eb56-498c-b5d0-d87e01f9b92	NULL	Default

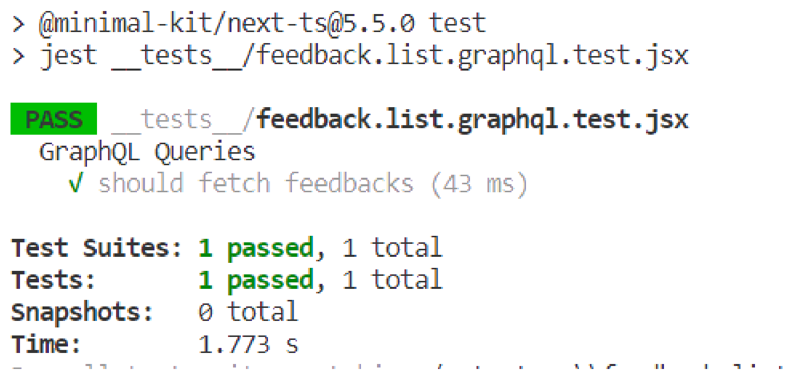
- **Status:** Pass. Both feedback records are present in the database with correct information.
- **Notes:** None

Test 4: UT14- Ensure the accurate display of the feedback list.

Automated Test

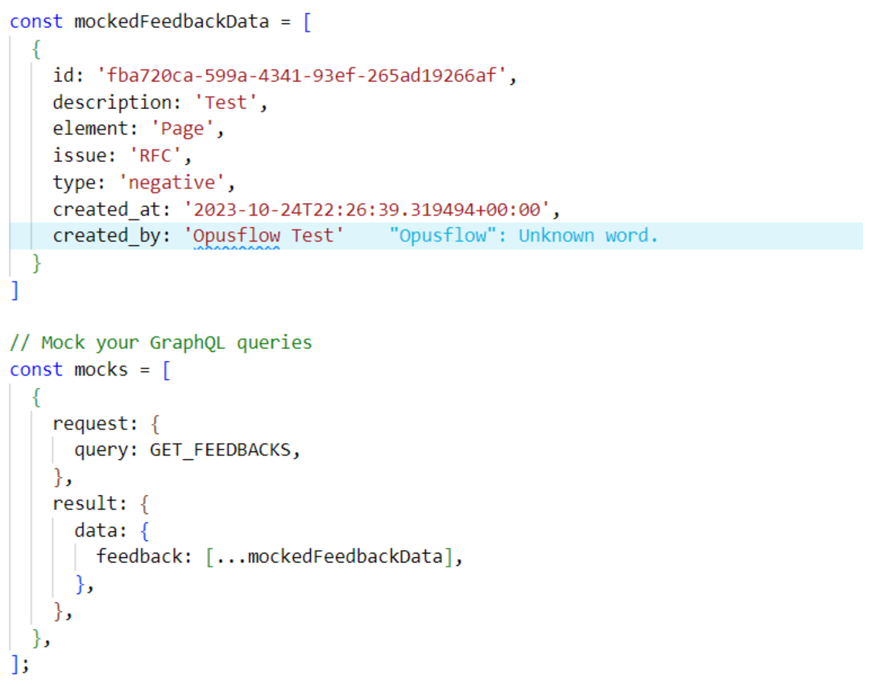
- **Description:** This test ensures that the GraphQL call for getting the feedback list, utilized in the OpusFlow InsightHub dashboard, functions properly.

- **Test Steps:**
  - Use Jest to run the file for testing the GraphQL to get the feedback list: feedback.list.graphql.test.jsx.
  - Execute the following command to run the test: npm run test -- __tests__/feedback.list.graphql.test.jsx.
- **Expected Result:** Automatic test pass.
- **Actual Result:** Screenshot for the result of the automatic test.

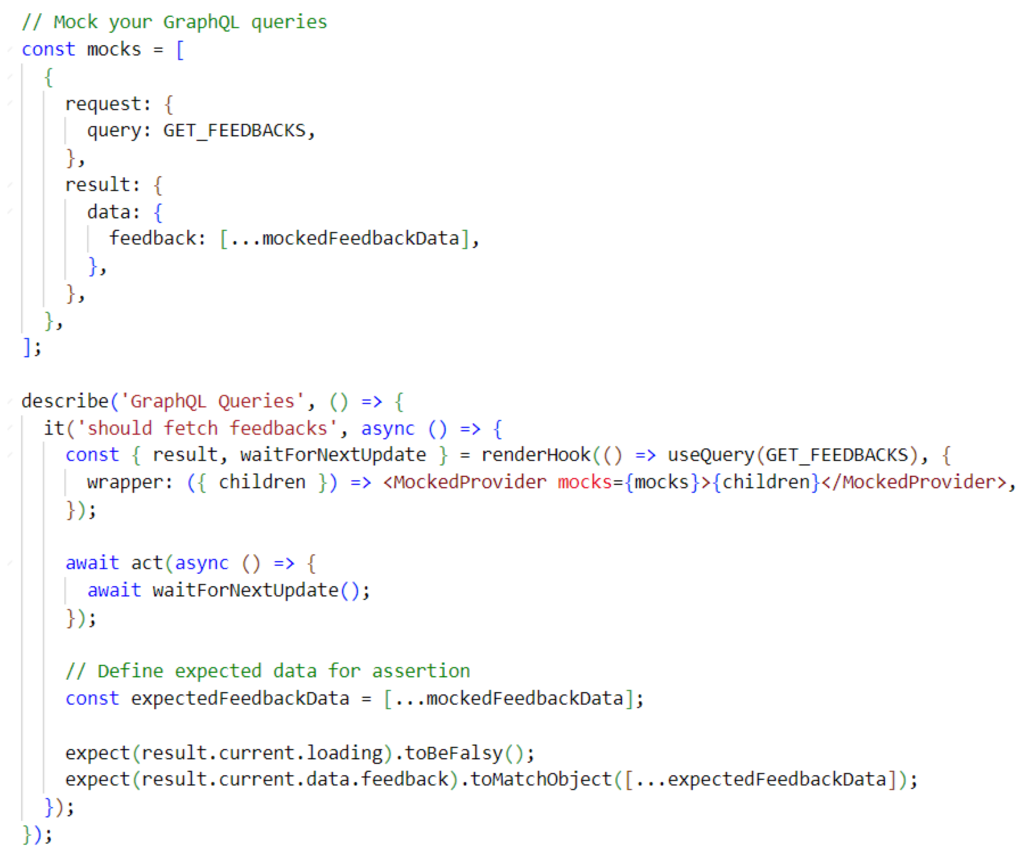


```
> @minimal-kit/next-ts@5.5.0 test
> jest __tests__/feedback.list.graphql.test.jsx

 PASS  __tests__/feedback.list.graphql.test.jsx
  GraphQL Queries
    √ should fetch feedbacks (43 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.773 s
```

- **Status:** Pass
- **Notes:** Test code for the automated test.



```
const mockedFeedbackData = [
  {
    id: 'fba720ca-599a-4341-93ef-265ad19266af',
    description: 'Test',
    element: 'Page',
    issue: 'RFC',
    type: 'negative',
    created_at: '2023-10-24T22:26:39.319494+00:00',
    created_by: 'Opusflow Test'    "Opusflow": Unknown word.
  }
]

// Mock your GraphQL queries
const mocks = [
  {
    request: {
      query: GET_FEEDBACKS,
    },
    result: {
      data: {
        feedback: [...mockedFeedbackData],
      },
    },
  },
];
```



```
// Mock your GraphQL queries
const mocks = [
  {
    request: {
      query: GET_FEEDBACKS,
    },
    result: {
      data: {
        feedback: [...mockedFeedbackData],
      },
    },
  },
];

describe('GraphQL Queries', () => {
  it('should fetch feedbacks', async () => {
    const { result, waitForNextUpdate } = renderHook(() => useQuery(GET_FEEDBACKS), {
      wrapper: ({ children }) => <MockedProvider mocks={mocks}>{children}</MockedProvider>,
    });

    await act(async () => {
      await waitForNextUpdate();
    });

    // Define expected data for assertion
    const expectedFeedbackData = [...mockedFeedbackData];

    expect(result.current.loading).toBeFalsy();
    expect(result.current.data.feedback).toMatchObject([...expectedFeedbackData]);
  });
});
```

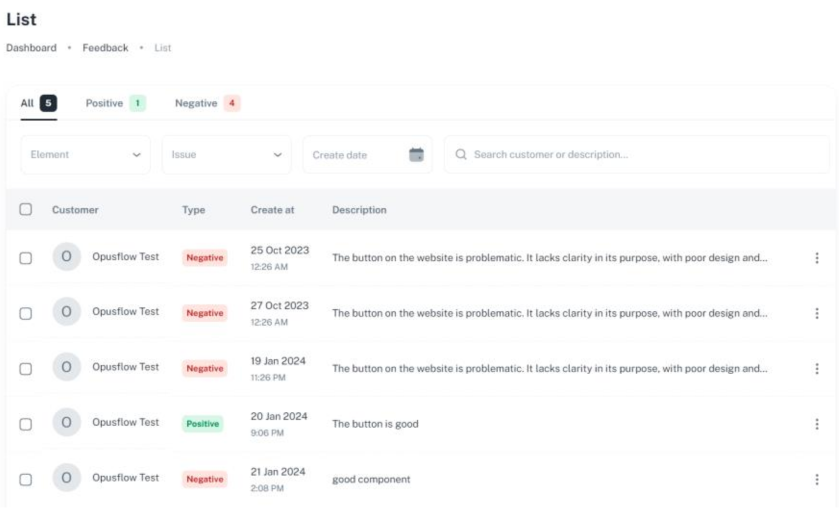
- **Explanation:**
  - The test code uses Jest and the @testing-library/react-hooks library for testing React hooks.
  - It mocks the GraphQL query using the MockedProvider from @apollo/client/testing.
  - The test checks whether the GraphQL query to fetch feedbacks returns the expected data and behaves as intended. It simulates the GraphQL query response with the mocks array.
  - The renderHook function renders the hook, and act is used to wait for the asynchronous operation (fetching data) to complete.
  - Assertions are made to ensure that the loading state is false, and the data returned from the GraphQL query matches the expected data.

Manual Test

- **Description:** This test evaluates the capability of the feedback dashboard in OpusFlow InsightHub to accurately fetch the list of feedback.
- **Test Steps:**
  - Log in to OpusFlow InsightHub using the default user email: demo@minimals.cc / password: demo1234.

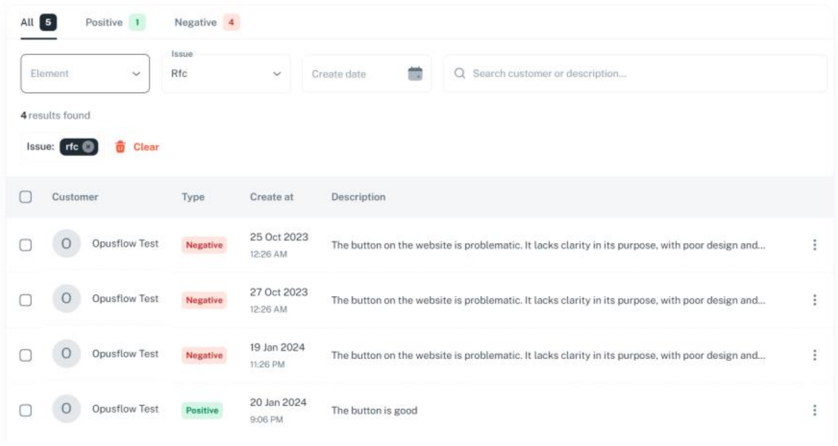
- Navigate to the dashboard page through the left navigation bar.
- Verify that the list of feedback is displayed.
- Use filter function to for filter all the feedback with RFC issue.

- **Expected Result:** The list of feedback is displayed with the ability to filter feedback based on type, element, and issue.
- **Actual Result:**
    - Screenshot of the feedback dashboard.


List
Dashboard • Feedback • List
All 5
Positive 1
Negative 4
Element
Issue
Create date
Search customer or description...

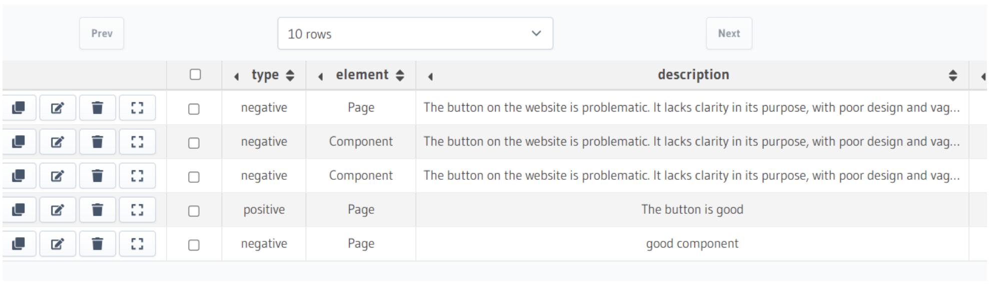
	Customer	Type	Create at	Description
☐	Opusflow Test	Negative	25 Oct 2023 12:26 AM	The button on the website is problematic. It lacks clarity in its purpose, with poor design and...
☐	Opusflow Test	Negative	27 Oct 2023 12:26 AM	The button on the website is problematic. It lacks clarity in its purpose, with poor design and...
☐	Opusflow Test	Negative	19 Jan 2024 11:26 PM	The button on the website is problematic. It lacks clarity in its purpose, with poor design and...
☐	Opusflow Test	Positive	20 Jan 2024 9:06 PM	The button is good
☐	Opusflow Test	Negative	21 Jan 2024 2:08 PM	good component

- Screenshot of the feedback dashboard after applying a filter on the issue.



	Customer	Type	Create at	Description
☐	Opusflow Test	Negative	25 Oct 2023 12:26 AM	The button on the website is problematic. It lacks clarity in its purpose, with poor design and...
☐	Opusflow Test	Negative	27 Oct 2023 12:26 AM	The button on the website is problematic. It lacks clarity in its purpose, with poor design and...
☐	Opusflow Test	Negative	19 Jan 2024 11:26 PM	The button on the website is problematic. It lacks clarity in its purpose, with poor design and...
☐	Opusflow Test	Positive	20 Jan 2024 9:06 PM	The button is good

- **Status:** Pass. The feedbacks are completely displayed and aligned with the records of feedback in the database.
- **Notes:**
    - Screenshot of the total record of feedback in the database.



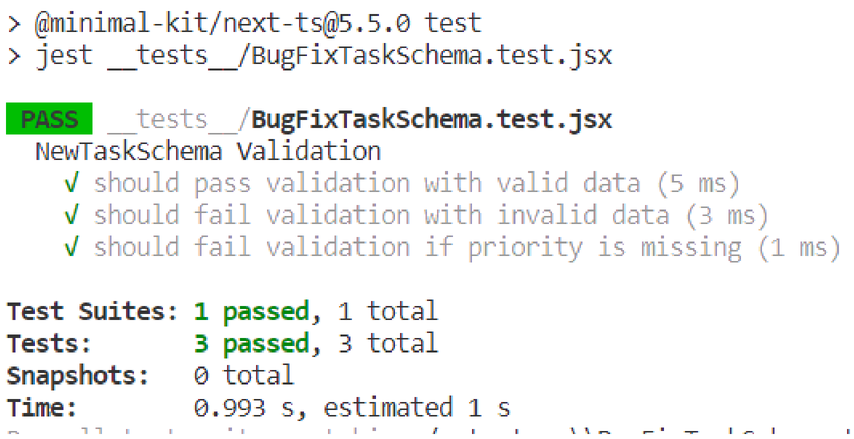
Prev | 10 rows | Next

	☐	type	element	description
	☐	negative	Page	The button on the website is problematic. It lacks clarity in its purpose, with poor design and vag...
	☐	negative	Component	The button on the website is problematic. It lacks clarity in its purpose, with poor design and vag...
	☐	negative	Component	The button on the website is problematic. It lacks clarity in its purpose, with poor design and vag...
	☐	positive	Page	The button is good
	☐	negative	Page	good component

Test 5: UT5- Validate the creation of tasks based on feedback.

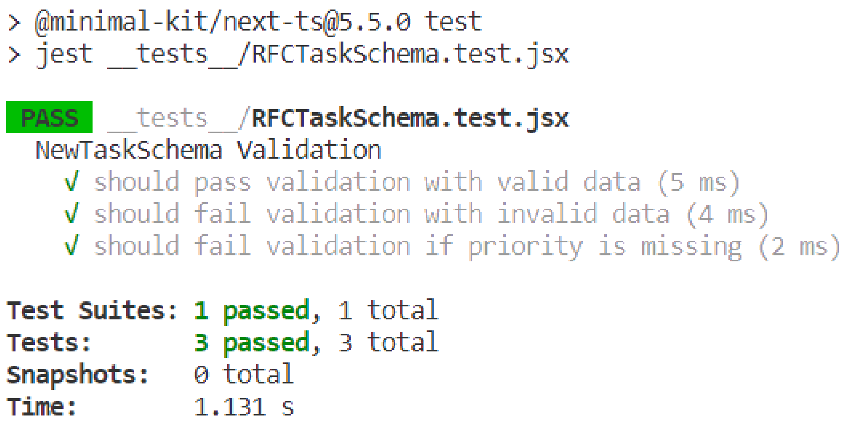
Automated Test

- **Description:**
  - Ensures that the validation forms for creating tasks (RFC and Bugfix) function correctly in OpusFlow InsightHub.
  - Tests the GraphQL process for creating tasks in the database.
- **Test Steps:**
  - Execute the following command to run the test: npm run test -- __tests__/BugFixTaskSchema.test.jsx
  - Execute the following command to run the test: npm run test -- __tests__/RFCTaskSchema.test.jsx
  - Execute the following command to run the test: npm run test -- __tests__/task.graphql.test.jsx
- **Expected Result:** All test should pass.
- **Actual Result:**
  - Screenshot for the result of the automatic test for Bugfix task validation.



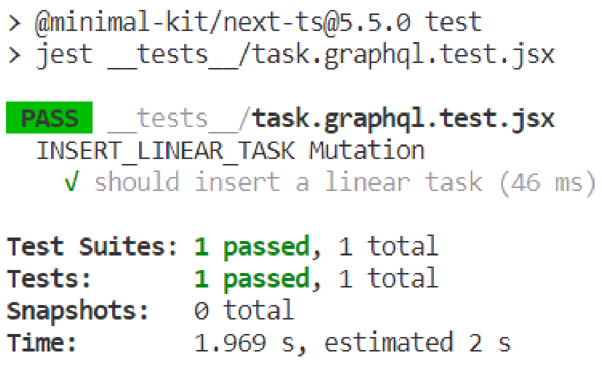
```
> @minimal-kit/next-ts@5.5.0 test
> jest __tests__/BugFixTaskSchema.test.jsx

 PASS  __tests__/BugFixTaskSchema.test.jsx
  NewTaskSchema Validation
    √ should pass validation with valid data (5 ms)
    √ should fail validation with invalid data (3 ms)
    √ should fail validation if priority is missing (1 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.993 s, estimated 1 s
```

  - Screenshot for the result of the automatic test for RFC task validation.



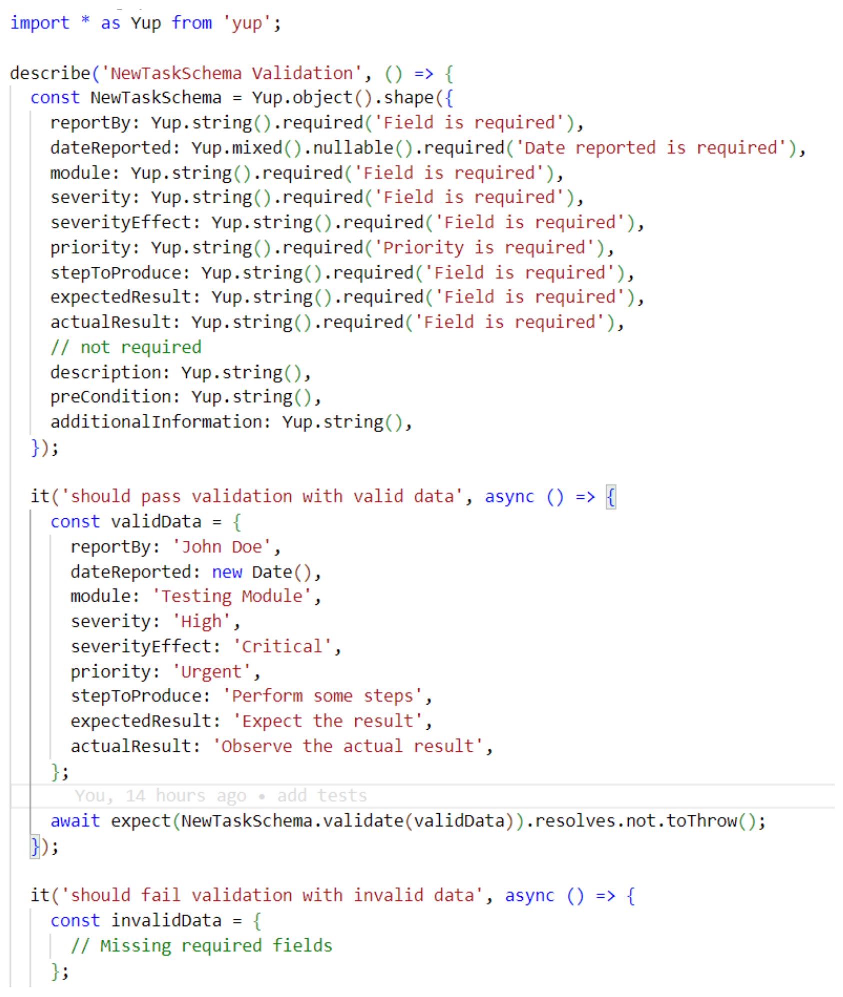
```
> @minimal-kit/next-ts@5.5.0 test
> jest __tests__/RFCTaskSchema.test.jsx

 PASS  __tests__/RFCTaskSchema.test.jsx
  NewTaskSchema Validation
    √ should pass validation with valid data (5 ms)
    √ should fail validation with invalid data (4 ms)
    √ should fail validation if priority is missing (2 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        1.131 s
```

- Screenshot for the result of the automatic test for task creation graphQL.

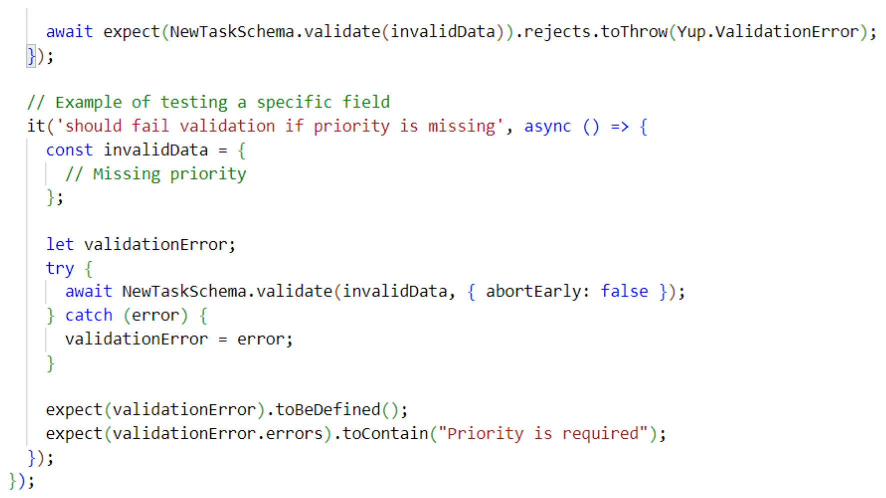


```
> @minimal-kit/next-ts@5.5.0 test
> jest __tests__/task.graphql.test.jsx

 PASS  __tests__/task.graphql.test.jsx
  INSERT_LINEAR_TASK Mutation
    √ should insert a linear task (46 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.969 s, estimated 2 s
```

- **Status:** Pass
- **Notes:**
  - Test code for the automated test for Bugfix task validation.

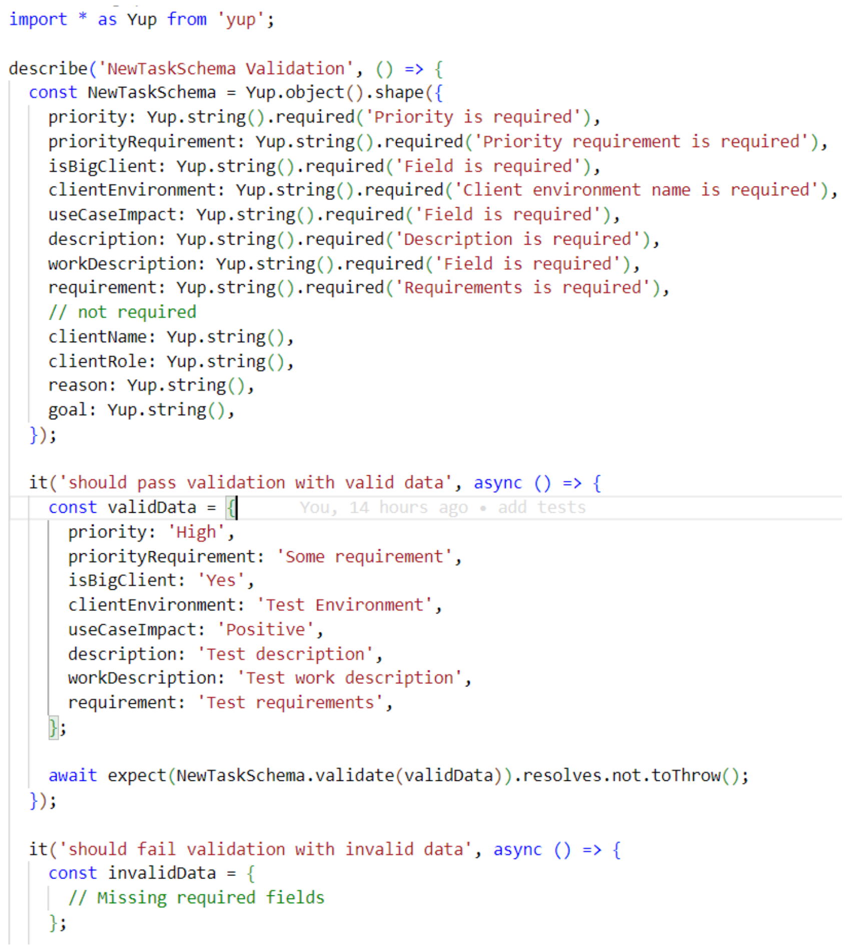


```
import * as Yup from 'yup';

describe('NewTaskSchema Validation', () => {
  const NewTaskSchema = Yup.object().shape({
    reportBy: Yup.string().required('Field is required'),
    dateReported: Yup.mixed().nullable().required('Date reported is required'),
    module: Yup.string().required('Field is required'),
    severity: Yup.string().required('Field is required'),
    severityEffect: Yup.string().required('Field is required'),
    priority: Yup.string().required('Priority is required'),
    stepToProduce: Yup.string().required('Field is required'),
    expectedResult: Yup.string().required('Field is required'),
    actualResult: Yup.string().required('Field is required'),
    // not required
    description: Yup.string(),
    preCondition: Yup.string(),
    additionalInformation: Yup.string(),
  });

  it('should pass validation with valid data', async () => {
    const validData = {
      reportBy: 'John Doe',
      dateReported: new Date(),
      module: 'Testing Module',
      severity: 'High',
      severityEffect: 'Critical',
      priority: 'Urgent',
      stepToProduce: 'Perform some steps',
      expectedResult: 'Expect the result',
      actualResult: 'Observe the actual result',
    };
      You, 14 hours ago • add tests
    await expect(NewTaskSchema.validate(validData)).resolves.not.toThrow();
  });

  it('should fail validation with invalid data', async () => {
    const invalidData = {
      // Missing required fields
    };
```

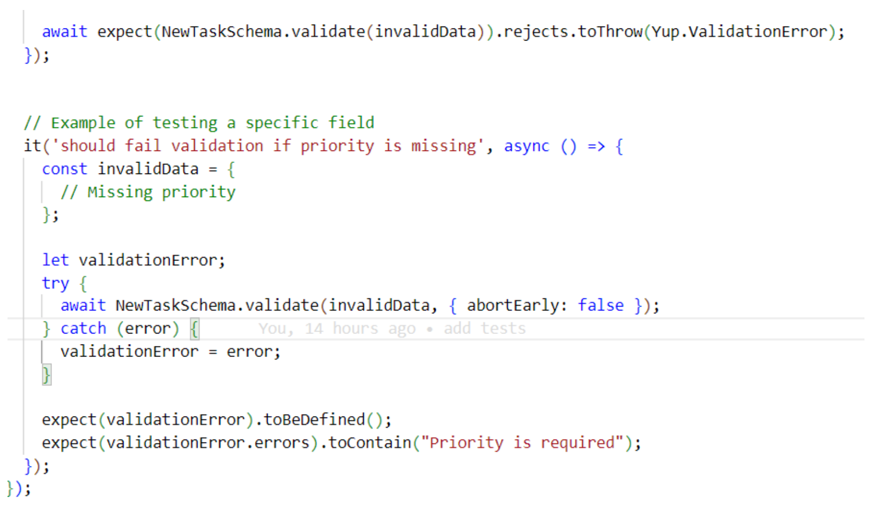


```
    await expect(NewTaskSchema.validate(invalidData)).rejects.toThrow(Yup.ValidationError);
  });

  // Example of testing a specific field
  it('should fail validation if priority is missing', async () => {
    const invalidData = {
      // Missing priority
    };

    let validationError;
    try {
      await NewTaskSchema.validate(invalidData, { abortEarly: false });
    } catch (error) {
      validationError = error;
    }

    expect(validationError).toBeDefined();
    expect(validationError.errors).toContain("Priority is required");
  });
});
```

- Test code for the automated test for RFC task validation.

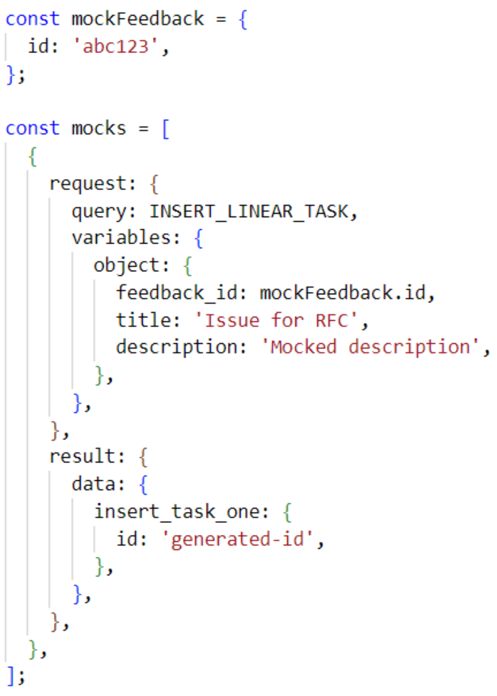


```
import * as Yup from 'yup';

describe('NewTaskSchema Validation', () => {
  const NewTaskSchema = Yup.object().shape({
    priority: Yup.string().required('Priority is required'),
    priorityRequirement: Yup.string().required('Priority requirement is required'),
    isBigClient: Yup.string().required('Field is required'),
    clientEnvironment: Yup.string().required('Client environment name is required'),
    useCaseImpact: Yup.string().required('Field is required'),
    description: Yup.string().required('Description is required'),
    workDescription: Yup.string().required('Field is required'),
    requirement: Yup.string().required('Requirements is required'),
    // not required
    clientName: Yup.string(),
    clientRole: Yup.string(),
    reason: Yup.string(),
    goal: Yup.string(),
  });

  it('should pass validation with valid data', async () => {
    const validData = {      You, 14 hours ago • add tests
      priority: 'High',
      priorityRequirement: 'Some requirement',
      isBigClient: 'Yes',
      clientEnvironment: 'Test Environment',
      useCaseImpact: 'Positive',
      description: 'Test description',
      workDescription: 'Test work description',
      requirement: 'Test requirements',
    };

    await expect(NewTaskSchema.validate(validData)).resolves.not.toThrow();
  });

  it('should fail validation with invalid data', async () => {
    const invalidData = {
      // Missing required fields
    };
```

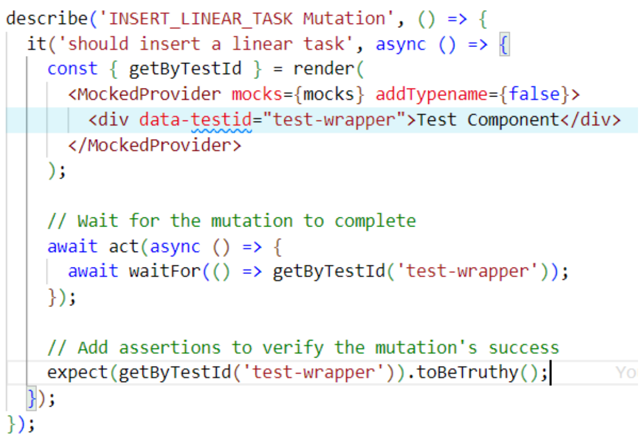


```
    await expect(NewTaskSchema.validate(invalidData)).rejects.toThrow(Yup.ValidationError);
  });


  // Example of testing a specific field
  it('should fail validation if priority is missing', async () => {
    const invalidData = {
      // Missing priority
    };

    let validationError;
    try {
      await NewTaskSchema.validate(invalidData, { abortEarly: false });
    } catch (error) {      You, 14 hours ago • add tests
      validationError = error;
    }

    expect(validationError).toBeDefined();
    expect(validationError.errors).toContain("Priority is required");
  });
});
```

- Test code for the automated test for task creation graphQL.



```javascript
const mockFeedback = {
  id: 'abc123',
};

const mocks = [
  {
    request: {
      query: INSERT_LINEAR_TASK,
      variables: {
        object: {
          feedback_id: mockFeedback.id,
          title: 'Issue for RFC',
          description: 'Mocked description',
        },
      },
    },
    result: {
      data: {
        insert_task_one: {
          id: 'generated-id',
        },
      },
    },
  },
];
```



```javascript
describe('INSERT_LINEAR_TASK Mutation', () => {
  it('should insert a linear task', async () => {
    const { getByTestId } = render(
      <MockedProvider mocks={mocks} addTypename={false}>
        <div data-testid="test-wrapper">Test Component</div>
      </MockedProvider>
    );

    // Wait for the mutation to complete
    await act(async () => {
      await waitFor(() => getByTestId('test-wrapper'));
    });

    // Add assertions to verify the mutation's success
    expect(getByTestId('test-wrapper')).toBeTruthy();
  });
});
```

- **Explanation:**
  - For the test code of Bugfix task validation & RFC task validation:
    - **Test Code Structure:**
      - The test code utilizes the Jest testing framework along with the Yup library for schema validation.
      - Each test case is organized within a **describe** block, providing a clear structure for test suite organization.
    - **Yup Schema:**
      - A Yup schema (**NewTaskSchema**) is defined to specify the expected structure and validation rules for the input data.

- **Validation Scenarios:**
  - **Valid Data:**
    - The test named 'should pass validation with valid data' verifies that the schema validation passes when valid input data is provided.
    - It creates an object (**validData**) with values that adhere to the specified Yup schema.
    - The test ensures that the validation does not throw any errors when processing the valid input.
  - **Invalid Data:**
    - The test named 'should fail validation with invalid data' examines the behavior of the validation when required fields are missing.
    - An object (**invalidData**) with missing required fields is used to test whether the validation throws a **Yup.ValidationError** as expected.
  - **Specific Field Test:**
    - The test named 'should fail validation if priority is missing' focuses on a specific field (**priority**) within the schema.
    - It assesses whether the validation correctly identifies and reports an error when the specified field is absent.

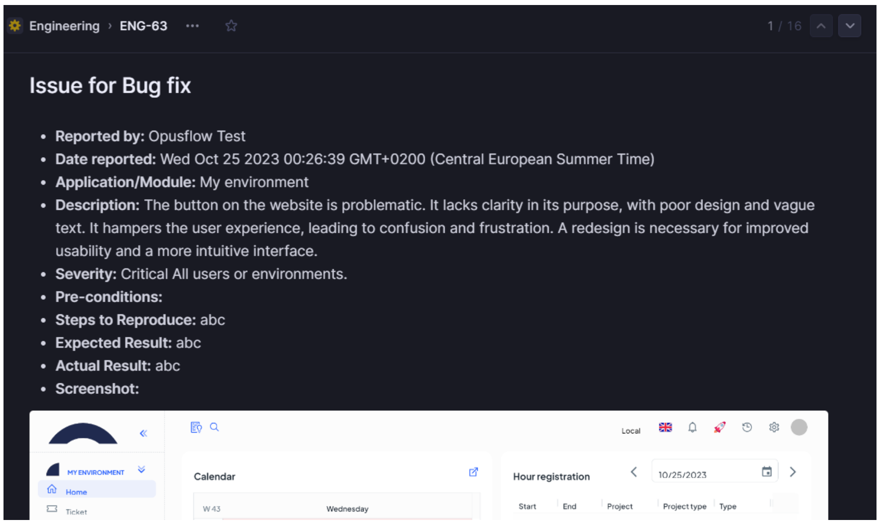
- For the test code of task creation graphql
  - The test focuses on the INSERT_LINEAR_TASK GraphQL mutation.
  - Mocks are defined to simulate the GraphQL server's behavior during the test.
  - A mock feedback object is created with a predefined ID, representing the feedback associated with the inserted linear task.
  - Assertions confirm the success of the GraphQL mutation.
  - Presence of 'test-wrapper' element indicates successful linear task insertion.

Manual Test

- **Description:** Tests the functionality to create a task within the OpusFlow InsightHub application.
- **Test Steps:**

- Log in to OpusFlow InsightHub using the default user email: demo@minimals.cc / password: demo1234.
- Navigate to the feedback dashboard page through the left navigation bar.
- Select a specific feedback and choose "create task" from the action row.
- Fill in the required information and press the create button

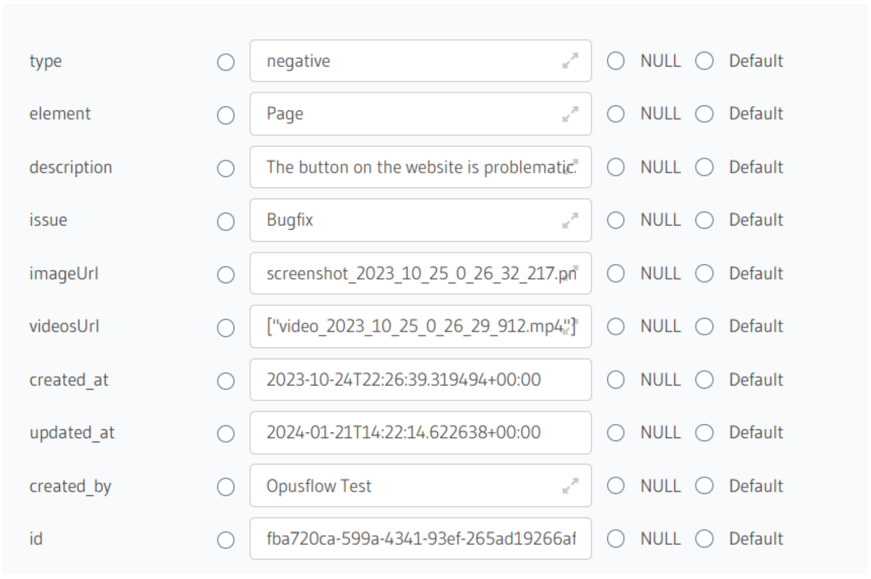
- **Expected Result:** A task will be created on Linear with information aligned with the associated feedback.
- **Actual Result:** Screenshot of the created task on Linear



Issue for Bug fix

- **Reported by:** Opusflow Test
- **Date reported:** Wed Oct 25 2023 00:26:39 GMT+0200 (Central European Summer Time)
- **Application/Module:** My environment
- **Description:** The button on the website is problematic. It lacks clarity in its purpose, with poor design and vague text. It hampers the user experience, leading to confusion and frustration. A redesign is necessary for improved usability and a more intuitive interface.
- **Severity:** Critical All users or environments.
- **Pre-conditions:**
- **Steps to Reproduce:** abc
- **Expected Result:** abc
- **Actual Result:** abc
- **Screenshot:**

- **Status:** Pass. Success create task on Linear.
- **Notes:** Screenshot of the feedback use to create task



type	negative	NULL	Default
element	Page	NULL	Default
description	The button on the website is problematic.	NULL	Default
issue	Bugfix	NULL	Default
imageUrl	screenshot_2023_10_25_0_26_32_217.pn	NULL	Default
videosUrl	["video_2023_10_25_0_26_29_912.mp4"]	NULL	Default
created_at	2023-10-24T22:26:39.319494+00:00	NULL	Default
updated_at	2024-01-21T14:22:14.622638+00:00	NULL	Default
created_by	Opusflow Test	NULL	Default
id	fba720ca-599a-4341-93ef-265ad19266af	NULL	Default