Learning Journal Entries

Student Name: Andrei Risea

Student ID: 220300153

Contents

Abbreviations

GS - Goldman Sachs
IB - Investment Banking
UX - User Experience
UI - User Interface

Monday 23rd Sept – Friday 4th October

New team, new environment

Around this time I was just over a month into my new team which is part of the Investment Banking sector. My new team focuses on building and managing apps as well as vendor tool integrations for bankers to use for their daily activities. There is a strong focus on consistency and coding excellence ensuring all code and features pushed out for one, do not break anything and two, provides value to bankers. This often means a slower development process but with an increased focus on user feedback, UX designers, coding excellence and efficiency. Coming from a small team in a completely different area with no front office users, I had the freedom to develop new and cool features fast. These features were often released within a week. Having spent 2 years in that team, I was very accustomed to the fast paced development style.

Upon reflection, moving into the new team felt very uncomfortable at first as it was something new to me and I am the type of person who loves consistency and doesn't look forward to change as I like being good at something. I did not know how to best proceed with work and felt that I could not show the new team what I can do as I was so used to delivering items quickly and I felt as if I was going too slow. I recognise that switching teams was beneficial as it exposed me to something new that challenged me daily. I realised that in my old team I was learning from the wrong end of the development cycle. I was pushing out cool features quickly using skills I already knew but due to the fast paced nature of it, I often pushed out bugs and edge cases. This would often lead to the feature not working as expected and having downtimes due to larger issues. In this case, I would learn from my mistakes and failures. In my new team, what I thought was slow was actually the learning process during the right time; before and during development. This is where I had to take a step back, look at the greater picture, plan and then execute. What I thought was hindering my growth was actually the opposite and allowed me to understand the full development cycle with best practices in mind. I realised soon that I was pushing out good code that had zero or slight bugs as compared to a much larger proportion in my old fast paced team. Upon deeper consideration, I realised that my initial discomfort with the slower pace was from my desire to show that I can deliver and to prove myself. Now, I see that a thorough, user-centered approach improves quality and thereby improves long-term deliverables. I now understand the importance of investing time upfront to refine requirements, incorporate user feedback, and implement best practices before coding. Consequently, I have grown more patient, strategic, and adaptive which are traits that I think will guide me in future roles.

Moving forward I would not be fast to judge a new team, new environment or project and I would take a step back and see how I should best proceed ensuring that I follow best practices and methodologies. Furthermore, I will strive to find a balance between speed and thoroughness rather than aiming only to produce features rapidly. For example, I will prioritise understanding user needs, planning carefully, and implementing rigorous testing before deployment too to prevent any unexpected consequences which is key in my new team. This can be done through learning more about what me team does and in depth specifically to better understand my role and what are the effects if I push bad changes.

Learning the business

In my old team I did not have a business to support. It was a backend and infrastructure team which meant I was not experienced with how development works for a business facing team. Moving into IB, I realised that I need to learn the business more. For example, why are we building and maintaining these systems for bankers? More specifically, what do bankers use our tools for and what do they allow them to do? For example book building apps and vendor integrations. Having business exposure is why I wanted to join this new team and allows me to learn and develop my abilities as a developer within a bank such as GS. I didn't quite grasp the necessity to understand the business as much as possible when I first joined as it was something completely new and foreign to me. I was too focused on delivering the code and proving my worth so to speak that I avoided learning the exact details of why we are needed and what we help the bankers do.

Upon reflecting, I realised that we, my team and I, are developers building tools and apps to be used by bankers directly. Hence, the work we do can either improve their productivity, or in the event of issues and errors, degrade their productivity which can cause financial repercussions. So, it is important to understand why we are building a new feature as to understand if it is indeed important and useful for the bankers. I also recognise that there is a lot of business logic within the codebase which is important for tasks such as surfacing and aggregating data which requires a certain amount of business knowledge to understand and fully grasp. I also recognise that I felt slightly overwhelmed at the start as there were so many business meetings and discussions that I could not fully grasp as I was missing the business knowledge. Furthermore, I recognise that without fully grasping IB concepts, I may not comprehend the extent of issues that may arise from bad code which is bad for the scope of the team and domain as it could cause production issues. Furthermore, I felt somewhat intimidated by the sheer volume of business-specific meetings and terminology. Looking back, I recognise that this initial uncertainty was actually a catalyst for growth pointing me in the relevant directions that would improve my knowledge.

Evaluating this experience, transitioning to a business facing IB team revealed the important value of domain knowledge. Understanding how bankers use our tools started making my work clearer. Initially, I focused solely on delivering code, which slowed my grasp of business context. I now see that aligning technical solutions with core business needs is essential for genuine impactful and reliable outcomes.

Moving forward, I plan to be more knowledgeable in the business. I will achieve this through doing my own reading into concepts specific to IB. I should also improve my communication skills to leverage people in my team who have been in the IB space more than me so that I can learn from them too. I will plan to ask more questions and try to apply knowledge learnt to the code itself to see how business logic and code fits in together. This will improve my skills as a business facing developer. In addition, I plan to set aside dedicated time each week for reading internal business documentation and attending workshops if available. I will also seek regular feedback from experienced colleagues to confirm that my interpretations of business requirements are accurate and to refine my understanding of domain-specific nuances that I may overlook.

Monday 7th October- Friday 18th October

Issue with frontend project

In my team, I was tasked with adding some tooltips onto some of our grids as when the page width was shrunk, the table would shrink and the data would not be visible. So, upon hovering on a table cell, the user would still be able to see the data allowing them to have multiple pages on the screen at one time. Following best practices, I looked to see how other applications had a tooltip component implemented that I could use. I found one, so I implemented it and instantly faced some issues. The tooltip was hidden behind the grid. My next step was to increase the Z score of the tooltip bringing it forward but that did not work. I tried moving the grid back behind the tooltip but this did not work either. I kept on debugging and narrowed the issue to CSS. I spent days trying to get the tooltips to work and to be shown trying everything and investigating deep into the grid source code to see if there was anything. After more than a week I spoke to someone on my team and he said to ignore how other applications have done it as they were old and outdated and to just use some external components rather than doing what has already been done by other teams. I just added the code to the frontend and without any CSS changes the tooltip worked instantly.

I was frustrated that I spent so long trying to get the GS implementation working. Upon reflection, I realised that this delay and bottleneck could have been significantly reduced if I communicated exactly what was wrong with my team explaining exactly what I had tried and did. I acknowledge I felt embarrassed to ask for help for such a little thing which was the thinking that caused the delay. Upon deeper consideration, my reluctance to ask for help was due to a misunderstanding of what it means to be a team player. I initially aligned seeking advice with a lack of ability on my part, but now I understand that collaboration can prevent delays, strengthen team relations and improve skills. In addition, I realised that sometimes it is not worth spending more than a day after exhausting all possible solutions trying to make something work. There were more tasks I could have been doing that would have had more of an impact.

This experience has widened my perspective on the development process. I now see that effective communication and the willingness to pivot approaches when stuck in a bottleneck are as crucial as the fundamental technical skills. I now understand that leveraging the team’s knowledge can transform a large setback into a learning opportunity preventing a similar case from occuring again.

Moving forward I will improve my communication skills in situations like this. If debugging an error, I will provide my team with a full list of exhausted attempts to solve this and verbalise the extent of the issue better regardless of the sise and importance of a feature. However, I will balance over and under using the team. After speaking to members of the team and mentors, they echo the fact that sometimes it's better to use something new and unseen before rather than trying to play around with existing old code as it is usually outdated and follows old best practices as well as old versions of packages too. Also, I will also look into setting hard time deadlines for feature progression. For example if I encounter a small error that can be solved by taking a completely different approach, I would dedicate a fixed time to solving it using the current approach such as a day before switching approaches to ensure continuous development. I can follow a common practice such as allocating 2 days of full concentration on debugging and problem solving the issue at hand and if after the 2 days, nothing has progressed only then would I change course. As I progress, I'll keep track of the complexities I encounter and the time spent on each attempt and will do regular self retrospectives to ensure that I'm not getting trapped in unproductive cycles which will only negatively impact both my team and I.

Integration into Java

One of my biggest strengths that I've been informed on by my old and new team is my frontend skills. This is what I spent nearly 2 years building upon in my old team so I am very comfortable with how React and Typescript works. However, moving into my new team I have had to learn Java. I have never used and touched Java code apart from the bare basics during a Year 1 Semester 1 module. This has caused me to be very slow in pushing out Java code changes as well as writing good, efficient code and has usually resulted in pushing sub optimal code that I have to go back and spend more time fixing and debugging.

Upon reflection, I've realised that sometimes I prefer to stay in my own bubble of frontend as that is what is most comfortable to me and in doing so, prevents me from learning and integrating myself more into Java. For example there were some times where I was able to select some backlog tickets to start working on and I chose the frontend tickets over the backend ones as that is what I was comfortable with. Upon reflecting, this is bad for me and the team. For myself it is bad because I won't learn the language and best practices fast enough and bad for my team because they experience delays in development. I do also recognise that I sometimes feel as if I am not doing enough when doing the backend tickets which are slow which is why I may seem to prioritise frontend tickets that I can get out reasonably fast to show my team that I can perform. However, I recognise that being slow at the start is normal and that I should embrace it and not be fearful of how it makes me look in front of my team and I have to start slow to become better for myself and for my team. Upon further reflection, the reason we use Java so heavily over other languages such as Python is that it is more mature, easier to maintain, develop, build and test. Hence, it is important I learn and understand the reasoning behind why the language is used for our use case so that it could give me an overview of the complexities of my team and to attempt to align a business use case with a language to understand how that marriage comes to be.

Moving forward, I plan to be more involved in the backend and the Java world so that I can become a better, more well rounded full stack developer. I will begin to prioritise more of the backend tickets Furthermore, I plan to focus on learning and developing my skills at the start of my time in the new team so that I can contribute a greater amount in the coming months in terms of full stack abilities. I also plan on upholding and learning Java best practices through delving into some Java courses and exploring other code bases and open source external code bases in Java. I think exposing myself to more Java, as much as I dislike trying something new, will be incredibly beneficial to not only myself, but my team too as they will benefit from a stronger, well rounded developer. My manager has also told me countless times that if I want to try something new I can just go to him and he will switch my projects around to help me align myself with my specific goals, so moving forward I will also be more transparent and open with my manager to let him know how I want to proceed.

Monday 21st October – Friday 1st November

Testing

Although having been in the firm for 2 years, I had little exposure to testing as in my old team we did minimal tests and any test was done in Python which is slightly different to Java. In short, I had begun writing some tests for a new feature I had written in Java to test the logic and flow. I was introduced to unit and integration tests and how they differ, their importance, and when best to use each by my team members. I then wrote some tests that would cover the functionality of a new date picker dropdown feature I had added. The tests passed and the feature was deployed. However, there were slight edge cases that caused some visual bugs and logic errors, particularly when users selected invalid dates. I quickly debugged and solved the issue and wrote more test cases to cover these scenarios.

Upon reflection, I understand that I was new to testing in general, particularly in Java, where I didn't fully appreciate the need for comprehensive and broad coverage. I felt embarrassed asking for help, as after two years in the firm, I assumed it was expected that I could write robust tests from the get go. In hindsight, I realise my hesitation to seek guidance only lengthened the learning curve, potentially introducing unnecessary risk to the live app used by bankers. Furthermore, I didn't fully recognise how business-facing applications demand extra diligence and care, as user-facing bugs can damage stakeholder confidence and disrupt core business activities.

Evaluating this experience, I see that insufficient testing not only caused frustration but highlighted the importance of proactively involving my team to explore best practices and verify test scenarios early on. I now recognise that unit and integration tests must encompass real-world data variations, such as invalid dates, to avoid similar issues.

Moving forward, I will communicate more openly about my weaknesses, as this transparency fosters both personal growth and stronger deliverables. I plan to adopt a more structured testing strategy, potentially integrating a "test pyramid" approach or Test-Driven Development for complex business features to systematically cover both normal and edge cases. Additionally, I will look into improving our testing documentation so the team can quickly identify what has or hasn't been covered. By setting clearer milestones for testing such as peer reviews by day two of development and edge-case testing by day three, I can catch errors before they reach production. In the future, I also intend to incorporate user feedback into my test scenarios, ensuring the tests reflect actual usage patterns rather than theoretical ones. This will help confirm that the user experience remains smooth, even for edge cases, and minimise the risk of embarrassing bugs in a live app. Ultimately, this learning has shown me that deep, upfront testing is not just a technical exercise but an ongoing commitment to the business needs and user trust that define a successful app.

Stakeholder issues

In IB engineering at GS, there are around 5 major stakeholders that contribute to the development of an app that we build or new features we implement. This means that there will always be some disagreement between the stakeholders and so a solution would have to be discussed and agreed upon which usually entails some compromise for every stakeholder apart from compliance which makes situations like these hard when compliance do have issues. I’ve never had this structure and amount of stakeholders in my previous team so it felt weird being part of one. I was accustomed to just building fast with what my manager and I thought was best for a feature or app. I never really planned and discussed features with as many people as we do now in IB so, experiencing these back and forth debates felt awkward to me at first. Especially when we were discussing the simplest concerns and elements of an app that I thought was pointless at the start.

Before joining one of these meetings in my new team, I would just sprint through the work without giving it a detailed thought. Upon reflection, I realise that this isn't the best practice for development. It actually may be that it won’t benefit anyone as we are pushing code that directly impacts the bankers and their productivity meaning if anything was to go wrong, it could impact client relations and current deals being worked on. I also recognise that I felt embarrassed to voice my concerns and opinions as I was very new to the team and the domain itself which meant that I was just doing what was agreed upon regardless of if I felt differently about it. Upon reflection this is detrimental to both my team and I as without a transparent environment, we won't be able to foster a productive and supportive environment. Reflecting on this situation, I realise that what initially seemed like unnecessary debates over minor details is actually a critical mechanism to ensure quality, compliance, and user satisfaction. Understanding the rationale behind these discussions such as protecting the bank’s reputation, ensuring legal compliance, and catering to user needs, helps me appreciate the care and precision that goes into even the smallest feature.

Evaluating this experience, it becomes clear that aligning a project to multiple different opinions and people ensures quality, regulatory adherence, and user satisfaction, essential for any user facing solution. The discomfort I felt stemmed from not fully grasping how high the stakes are when deploying code for business focused environments. Understanding that each stakeholder, particularly compliance, adds valuable checks has shifted my perspective. What I once perceived as red tape is actually a mechanism to protect the bank’s reputation and maintain legal integrity.

Moving forward I should be more open to voicing my concerns regardless of the opinion itself as this is what will help me grow and learn the space quicker. I should also prepare more for meetings like this beforehand so I am looped into the current discussions even if they are directly outside my scope of projects. This will allow me to understand the wider scope of such stakeholder meetings. Having seen firsthand that debate leads to stronger decisions, I am committed to embracing these discussions as growth opportunities. I will plan to engage confidently in stakeholder meetings ensuring to be prepared with context and thoughtful input which will help me evolve as a more adaptive and strategic developer.

Monday 4th November – Friday 15th November

Bug in my old team

In my old team, the main book of work was this tool that helps track programmes and initiatives in the division that senior management would do. It is a tool that is heavily used by senior management and was one of my most used projects. This tool was heavily tested by me and I ensured the functionality and logic would work. However, the code failed one day on a data quality issue from the response of an API. My old manager informed me of this but at the time I was busy with my new team commitments and deadlines so had no time to debug, but I did provide some context to a new joiner in my old team who was tasked with debugging the issue. However, the new joiner would keep coming back to me asking for help and guidance around the code. The code was complex handling so many different edge cases and logic that the person could not really understand the flow. Although I had provided flow charts explaining the basic flows, the person could not understand it. So, I had to devote some more time walking them through the code block at a time trying to explain what everything does as it was all bespoke and custom to the use case. This meant that I was context switching a lot between ongoing deadlines in my new team and bugs in my old which overwhelmed me and increased the pressure.

Upon reflection I realised that although the code could not be simplified, I could have at least made it more readable by structuring the code and adding comments and more documentation following best practices. Although I had a basic flow diagram documented, for the complexity of the code, I recognised that it would have been better to write an in depth document for the code allowing any new joiner to understand the code and get started without a huge barrier. I also recognise that I felt pressured to deliver to both my old and new team and wish I had communicated better with my old team explaining that I have new commitments. Though, I am conscious that I should have provided better documentation and information. Upon reflection, I recognise that my reluctance to thoroughly document and structure the code earlier began from focusing too heavily on immediate functionality rather than long-term maintainability. This experience overall helped me understand that even well-tested code becomes a barrier if newcomers can’t navigate it without constant hand-holding.

Evaluating this experience, I see that the actual cost of insufficient documentation extends beyond single bugs as it can also disrupt project timelines and place unnecessary stress on multiple team members and other teams too if the main developer left. I also understand that domain specific knowledge can be a significant obstacle if it remains only in the mind of the original developer otherwise known as tribal knowledge. Had I invested more time in documenting the rationale behind key logic blocks or detailing how APIs interact with our tool, the new joiner could have ramped up more smoothly. This lack of detail emphasises the need for a broader commitment to knowledge transfers to maintain a collaborative software development.

Moving forward I will ensure to write better and more structured documentation to allow for anyone, regardless of domain knowledge, to learn and understand the flow of features I build better. I will also focus on making code readable through restructuring and commenting and I understand that when possible, simplifying code is best. I’ll also treat documentation and code clarity as part of the development lifecycle rather than afterthoughts, recognising that clarity upfront prevents future blockers, reduces onboarding delays, and promotes smoother handovers. This renewed focus on maintainability will not only help colleagues working on my code but also enhance my own efficiency if I ever revisit these features in the future.

From a communication standpoint, I will clarify my current workload when assisting other teams or new joiners, managing expectations so that tight deadlines don't become unachievable.

Networking

I consider myself somewhere between an introvert and extrovert. What I have seen happen is prevent myself from taking the steps required to speak to new people. For example, I only just recently spoke to the global head of our team after 2 months of putting it off. This is just one example this week, but usually this is a common theme for me especially looking back into my old team too.

Upon reflection, I recognise a pattern where I delay speaking with colleagues, especially those in senior positions, from feeling anxious or unsure of how they might respond. This hesitance is not because I lack the ability to communicate, but rather because I let worry overshadow my willingness to engage. Upon further reflection, I realise that my hesitation may stem from a fear of appearing uninformed or not meeting expectations. Understanding this root cause clarifies that the barrier is more about my self-perception than any actual external issue. I recognise that by acknowledging this habit, I can better understand the reasons behind my reluctance and begin to see such interactions as opportunities for growth, rather than sources of intimidation. This new awareness is helping me realise that most individuals, regardless of their role, welcome honest conversation, and that my efforts to reach out can build more genuine connections within my professional community.

Moving forward, I plan to start with small, low-pressure exchanges, like talking over a coffee, before gradually approaching more formal, work-related discussions. Each conversation will serve as a chance for me to practice, learn, and gain confidence. To improve my progress, I will also seek feedback from mentors and peers, who can offer support and suggest areas for improvement. Over time, this step-by-step approach should help me develop stronger communication skills, feel more at ease speaking with anyone in the organisation, and ultimately present myself as a more engaged, authentic, and self-assured professional. Furthermore, I will set small communication goals, such as initiating one meaningful conversation with a new colleague each week or so, and reflect on my progress regularly. By doing this, I can track my comfort level, celebrate small victories, and continually challenge myself to expand my abilities to make me a more well rounded team player.

Monday 18th November – Friday 29th November

New feature introduced

I was tasked with incorporating a complex feature for one of our apps. The task was split into two. The backend logic with database interactions and the frontend ticket to combine the design and the logic together. I first started out with the backend and completed it within a couple of days. I then started the frontend which became very complex as I had to account for edge cases such as boundary and extreme inputs. However, I soon noticed that the backend change I wrote was not working. Upon integrating the frontend to it, the backend would fail. This was because the backend code was too simple and tailored to only a few base cases. I started debugging and realised that the problem was much greater and more complex than anticipated. As the only developer on this I had to start debugging and fixing the backend which took over a week to do. This slowed down development and made me context switch from the frontend tasks. This feature was also asked for by users and they would want this feature in production before they start using our app so I was under pressure to deliver this quickly. After a few weeks, the filter was complete and working and was deployed with a over a week's delay.

Upon reflection I realised that this delay could have been prevented if I had tested the backend ticket fully with data in the form that it would be when integrating with the frontend. This was a mistake and an oversight on my behalf which meant that the backend ticket was tested with simplistic data and oversimplified test cases which prevented the bigger error from surfacing. This oversight highlighted a critical gap in my testing approach. I realised that relying solely on simplified test cases provided me with a false sense of security. In a business environment where data intricacies are paramount, I need to anticipate complexity and possible edge cases upfront. I also recognise that I was a little embarrassed that I did not catch this bug meaning that the feature was delayed slightly. This has taught me the importance of having test cases that match and mimic true usage and data that would be the case when integrating with the frontend code. Though, looking back it may also have been beneficial to start working on both the frontend and the backend at the same time as it would have allowed me to develop the backend to the actual data form and structure that I know would work and integrate well with the frontend but I recognise that this would be too overwhelming and involve a lot of context switching.

Evaluating this experience, it’s evident that a poor testing approach from my own part for the backend introduced preventable setbacks. Neglecting real world complexity caused delays, and added a lot of pressure to meet user requirements. While the final deployment was a success, the process highlighted the downsides of oversimplifying test cases. The gap between frontend requirements and backend data structures should only emphasise the need for complex test cases as well as robust integration tests. This experience has changed my perspective, emphasising the importance of planning and test driven development.

Moving forward I will now plan and design a frontend data types / data structure alongside any backend API feature requests. This will ensure that I can still split the work into 2, frontend and backend but also ensure that I know what the data types are so that I can better test and code the feature in the backend. This will prevent me from going in blindly and will also mean test cases will be more rigorous. Furthermore, I will create a structured testing strategy that incorporates complex data sets from the onset and maybe even attempt test driven development where I write the tests for a new feature first then build the actual feature around it. Also, I will try to stay calm and prevent context switching when under pressure to prevent becoming overwhelmed which would only delay features and releases more.

Time management

During this period, I had 4 frontend tickets to pursue for the current sprint we were on. I managed to complete the 4 tickets sequentially and get them into a review stage. However, the code reviews were piled one on top of the other meaning I had no more tickets to do and there was now a lot of review work for members of my team. My team managed to slowly review them thoroughly which meant I had 4 tickets to resolve comments and tweaks meaning I had to context switch a lot and it was difficult to take one ticket to completion. However, by the end of the sprint I did manage to complete the 4 tickets, resolve all comments and then push to production.

Upon reflection, I realised that trying to complete all four tickets quickly led to a bottleneck once they hit the review stage. Since all of them needed feedback at the same time, I had to switch repeatedly between tickets, making it hard to maintain focus and finish any one task smoothly. Upon deeper reflection, I recognise that my drive to deliver all four tickets rapidly stemmed from a desire to demonstrate my efficiency and productivity. In hindsight, this approach inadvertently caused more complexity by overloading the review pipeline. This experience showed me the importance of pacing my work and communicating with my team early on, so that code reviews happen steadily rather than piling up. It also helped me understand that smaller, more frequent feedback loops can improve the overall quality of the work and reduce stress for everyone involved in the process.

Evaluating this experience, it’s clear how rushing to complete tasks without scheduling can disrupt the entire team’s workflow. A well-planned approach where each ticket is reviewed promptly after it’s coded not only ensures more thorough feedback but also aligns with the principle of continuous collaboration. Additionally, I see how daily standups could be leveraged more proactively to flag potential review overload early, allowing for better team alignment on pacing and code ownership. This sprint underlined the importance of time management not just for my own workload but for the entire group’s efficiency.

Moving forward, I plan to manage my tasks in smaller, more manageable chunks and seek feedback early. Instead of rushing to finish all items at once, I will aim to complete a few tickets, get them reviewed, then move on. I will also speak openly with my team to align on review times and identify any potential roadblocks instead of reaching review bottlenecks. By doing this, I hope to build a smoother development environment, maintaining a better sense of focus, and helping the team work more efficiently as we move from one sprint to the next. Additionally, I aim to improve my ability to switch smoothly between tasks, developing better note-taking habits, prioritising clear documentation, and maintaining transparent progress updates which is already being done during daily standups.

Monday 2nd December – Friday 13th December

Another bug in my old team

Around this time, I was sent a message by a senior product manager that sat alongside my old team's reporting managing director. He stated that he encountered a bug with a tool that I built that would help him generate some pdfs. He stated that the bug only occurred through a specific path which was to open a previous template for a pdf and then try to add and amend some input parameters. Upon doing so, the tool would error out and fail to generate the pdf. I instantly got to debugging. I first tested out the basic pathway which was to start from scratch and copy the input parameters to see if they were working, and it was fine so I knew it wasn't a data input error. I then decided to look at the network console as upon parameter submission an api would be called and it was clear that a specific API was failing. Reading the error logs made the issue and bug clear as day. It was a dictionary key lookup error in python. The crux of this issue was just a simple for loop which would check every object in a list for at least 2 values. But when loading into a template it would add an extra metadata field to the object so the for loop would incorrectly deem that row as a valid row. The solution was to improve the logic for that loop to ensure that it would specifically check for only the relevant fields to have been populated rather than looking at everything and guessing. I then emailed this detailed debugging information to a dev in my old team who would then go on to fix the issue.

When I got the message I did not know what to feel apart from being annoyed at myself. I also felt slightly embarrassed that a senior member who has used my tools before found a bug that prevented him from moving forward with his daily tasks. I knew I had caused an inconvenience and was somewhat angry at myself for letting this happen. I felt under pressure as I wanted to show the person that I am capable of fixing my mistakes quickly and with due care. Looking at this situation, the bad thing was that I caused an inconvenience to someone and reduced their trust in my tools which also affected my developer reputation. Upon deeper reflection, I realise that my testing approach lacked the depth and variation needed to catch subtle logic errors. My tests focused on the most common input scenarios, neglecting more nuanced pathways like loading previous templates. If I had simulated user workflows more comprehensively, this error might have surfaced before reaching production. This incident evidenced that even a single bug can impact more than just functionality; it can erode stakeholder confidence too. Recognising that my tools directly influence the efficiency and trustworthiness of daily operations taught me that quality assurance isn't only a technical must, it's a professional responsibility to uphold the team's and my own credibilities.

Moving forward I will now implement a more rigorous testing procedure for any tool that I deploy ensuring to have a checklist that will ensure my tests cover all the possible entry points and logic in the code so that every possible scenario has at least one test case that can prove the code works as expected. This won't completely eliminate edge cases but will ensure the vast majority of them are covered as there is always the chance of human error for these tools. In addition I will also ensure to stop writing simple generic code in important logic functions. Simple and generic code is always not as readable as specific and detailed code meaning that when I started debugging I had to remember at the start of the year as to what the method did. Having more specific code will also ensure the code is built for purpose and can handle edge cases by having conditional logic and checks in place to prevent such situations. Through debugging I also learnt that my old code in my old team was not as documented as I would have liked, meaning it was hard to debug quickly. Moving forward, I will ensure to document important functions of a feature to help with future debugging issues and any developer who may have to interact with the code to allow for ease of use and access. Furthermore, I'll regularly reassess and update my testing procedures to ensure they remain aligned with evolving requirements and user behaviors. I'll keep refining documentation standards and coding practices, treating each development cycle as an opportunity to learn, adapt, and enhance the reliability and maintainability of the solutions I provide to my team and our stakeholders.

Difference of opinions

I was working on adjusting some aggregation code in the frontend that is used to calculate some statistics based on data points for users to see and use. This involves some complex business logic cases to ensure the statistics are calculated correctly and are in line with how bankers would manually calculate them. I continued working on this and completed the ticket soon enough adding test cases too to ensure all is working as expected. I submitted it for review and someone in my team preferred not to have such long and simplistic code and favoured a more efficient and shorter code option. I explained that as it handles some complex business logic, I think it would be best to keep the code more readable and easy to understand so that in the eventuality of an error, it is easily debugged. However, the other person suggested that the code could be improved as it was too long and did not fit in with the other style of the code to which I mentioned that the other parts did not handle business logic. In the end, my approach was taken as we were all aware that different developers have different opinions and as long as the tests pass and the functionality works (in situations such as this with strong opinions from both sides) , it is fine to proceed.

Upon reflection, I recognise that this interaction brought to light the complexities of navigating differing coding habits within a team. While I felt confident in my more explicit, verbose approach, primarily to ensure clarity around business logic, my colleague advocated for a more streamlined and concise style. This tension revealed that beyond the correctness of the code, there are often nuanced debates between readability and maintainability across a codebase. I recognise that my communication, although reasoned, could have been more receptive and empathetic, acknowledging that my colleague’s preference stemmed from a genuine desire to improve the overall code quality. I understand the importance of approaching such discussions with openness to alternative viewpoints and a willingness to reconsider and adapt my stance.

Evaluating this scenario, I see that differences of opinion often arise when each developer’s preferred style is rooted in genuine concerns, be it readability, performance, or maintainability. Rather than positioning my approach as definitely correct, I could have encouraged a collaborative dialogue on establishing a middle ground, possibly referencing official style guides or team norms. This ensures that the code remains both understandable and efficient while aligning with the broader codebase style.

Moving forward, I will aim to foster a more collaborative environment when differing opinions on code structure arises. Instead of presenting my perspective as the definitive solution, I will suggest conversations, carefully listen to colleagues' reasoning, and try to find common ground. I plan to refine my communication skills, showing respect for alternative approaches, and continuously reflecting on the rationale behind my coding choices, so I can become a more adaptable, considerate, and effective team member through asking questions and suggesting offline meetings that can help me understand the broader team and the best practices we follow.

Furthermore, I will continually weigh the trade-offs between overly simplistic code and maintainability, seeking feedback early in the process of development. This includes suggesting offline pair-programming sessions or quick whiteboard discussions to align on how best to handle the most complex parts of a ticket. Over time, I hope this approach will help me become a more adaptive, considerate, and strategic team member, capable of incorporating varied coding styles while respecting the underlying intent behind each colleague’s feedback.