Teammate disagree

Situation:

While working on Intentwise, we frequently ran queries to extract aggregated insights from data stored in PostgreSQL. The queries involved joining multiple tables with millions of records, causing slow performance. Since we only queried the data once per day from some tables, I proposed using a Materialized View to precompute the results and store them efficiently, avoiding expensive joins every time.

Task:

The challenge was to optimize the daily reporting queries while ensuring the data remained consistent and available for analysis. My teammate disagreed, arguing that materialized views required manual refreshes and might not be worth the effort.

Action:

- I analyzed query execution plans using EXPLAIN ANALYZE, showing that the join operations were the main bottleneck.
- I implemented a proof of concept (PoC), where I:
  - Created a Materialized View that stored precomputed results.
  - Scheduled a nightly refresh using pg_cron to update the view automatically.
  - Compared execution times between querying the materialized view vs. running joins on raw tables.
- The PoC results showed a 70% improvement in query execution speed without affecting the accuracy of the reports.
- I presented these findings to the team, explaining that while a manual refresh was required, automating it would make it seamless.

Reaction:

Initially, my teammate was hesitant, concerned about stale data. However, after seeing the significant performance gains and understanding that data consistency wasn't an issue due to our daily refresh schedule, our team agreed to use this approach in production.

Outcome:

- We implemented the Materialized View, which reduced the query execution time from several minutes to seconds.
- The pg_cron job ensured that data was always up to date without manual intervention.
- The optimized approach improved dashboard responsiveness, enhancing the overall user experience.

- The success of this solution encouraged the team to adopt similar precomputed query strategies for other performance-critical reports.

Helped others

During my internship at Ernst & Young, one of my teammates was struggling with the Django framework while working on an Exploratory Data Analysis (EDA) application. They were having difficulty setting up Django models, handling database migrations, and implementing API endpoints for data retrieval. Since the project had a tight deadline, their challenges were delaying progress.

Task:

I wanted to help my teammate quickly understand Django ORM, database interactions, and API development so they could contribute effectively to the project.

Action:

1. Set Up a Pair Programming Session – I worked with them to explain the Django project structure, helping them understand models, views, and serializers.
2. Guided Through Django ORM – I demonstrated how to create models, run migrations, and efficiently query the database using Django ORM instead of raw SQL.
3. Helped Debug API Issues – They were facing errors in API responses due to incorrect serializer configurations. I helped them fix serializers and test endpoints using Postman.
4. Created a Knowledge Document – I compiled best practices and common troubleshooting steps in Django, which helped not only them but also other interns.

Outcome:

- My teammate quickly grasped Django fundamentals and successfully built API endpoints for the project.
- The EDA application's backend was completed on time, ensuring smooth integration with the frontend.
- They later helped another intern with Django-related issues, further strengthening team collaboration.

Mistake

Situation:

During my internship at Beehyv, I was working on a React project where I had to build a dashboard that displayed data updates. While implementing it, I made a mistake in handling page updates, which caused the page to refresh unnecessarily every time new data was fetched.

Task:

The goal was to update the dashboard smoothly without making the user experience frustrating by constantly refreshing the entire page.

Action:

- I initially placed the data-fetching logic in the wrong part of the code, which caused the page to reload every time the data was updated.
- My teammates noticed the issue, and after debugging, I realized that the data should be fetched only once when the page loads rather than on every update.
- I fixed the issue by adjusting where and when the data was fetched, ensuring updates happened without affecting the rest of the page.
- I also tested the changes to make sure the dashboard worked smoothly without unnecessary reloads.

Outcome:

- The issue was resolved, and the dashboard updated efficiently without refreshing the entire page.
- I learned the importance of carefully handling updates in a React project and making sure my code didn’t create unexpected behaviors.
- This experience helped me become more detail-oriented and better at debugging issues before they affect the user experience.Initially, I didn’t notice the issue myself. While testing the dashboard, one of my teammates pointed out that the page was refreshing more frequently than expected. This prompted me to investigate, and upon debugging, I realized that I had placed the data-fetching logic in the wrong part of the code, causing unnecessary reloads. Once I identified the mistake, I quickly worked on a fix by adjusting where and when the data was fetched, ensuring that updates happened smoothly without refreshing the entire page. This experience taught me the importance of careful testing and reviewing my work to catch such issues early

TRUST

Situation:

During my internship at Beehyv, I was working on a React-based dashboard for a client project. Since I was relatively new to React at the time, my teammates were unsure whether I could handle building key components on my own. They initially assigned me smaller tasks, but I wanted to prove that I could take on more responsibility.

Task:

I needed to gain my team's trust by demonstrating that I could deliver quality work on time and contribute effectively to the project.

Action:

- I proactively learned React concepts, studied best practices, and quickly familiarized myself with the project structure.
- I took the initiative to build a complex component for real-time data display, ensuring it functioned smoothly without affecting performance.
- When I faced challenges, I asked the right questions and collaborated with teammates instead of waiting for them to fix things for me.
- To ensure quality, I tested my component thoroughly, catching and fixing potential issues before submitting my work.

Outcome:

- The component worked flawlessly, and my teammates were impressed with how quickly I delivered a functional solution.
- After this, they started assigning me more critical tasks, trusting that I could handle them independently.
- This experience taught me that initiative, collaboration, and reliability are key to earning trust in a team environment.

5. Tell me about a time when you couldn't meet your deadline?

Situation:

While working at **Beehyv**, I was developing an **API to parse Excel and CSV files** and store the extracted data into a **database**. Initially, I implemented a straightforward approach to **process files sequentially**. However, during the **testing phase**, I encountered a **performance issue when handling large files**—the API took too long to process and sometimes even failed due to memory constraints.

Since this issue surfaced late in the development cycle, it impacted the timeline, and I **was unable to meet the original deadline**.

Task:

I needed to **refactor the API** to efficiently handle **large files** while ensuring that the parsing process was **optimized and scalable**. The challenge was to fix the issue **without compromising data integrity or performance**.

Action:

- I analyzed the bottleneck and identified that **processing the entire file at once** was causing high **memory usage and slow performance**.
- To resolve this, I **refactored the code to process files in chunks** rather than loading the entire file into memory at once.
- I implemented **parallel processing** using multithreading to **speed up parsing while reducing memory consumption**.
- After implementing these optimizations, I **ran extensive tests with large datasets** to validate the improved performance.

Outcome:

- The **optimized API handled large files smoothly**, reducing **processing time by over 50%** while preventing memory issues.
- Although I **missed the initial deadline**, I communicated proactively with the team and provided **a revised timeline** for delivering a robust and scalable solution.
- This experience taught me the importance of **early performance testing** and the need to anticipate scalability challenges when designing APIs.

Situation:

In the Intentwise Tool, the data engineering team manually checked report statuses every day to determine which reports were enabled or disabled for each organization. Based on this, they manually added or removed ETL packages and schedules in Xplenty, which was time-consuming, prone to human error, and inefficient.

Task:

I wanted to eliminate the manual effort, reduce errors, and improve efficiency by automating the entire process.

Action:

To solve this, I designed and developed an automation pipeline that:
Dynamically checked report statuses for each organization.
Automatically created, updated, or removed ETL packages in Xplenty, ensuring only the required reports were processed.
Implemented automated scheduling, so data transfers ran smoothly without manual intervention.
Integrated real-time Slack notifications, keeping the team informed about changes.
Handled edge cases, such as missing reports or API failures, to prevent data inconsistencies.

Result:

Reduced manual work by 80%, saving engineers several hours daily.
Eliminated human errors, ensuring only valid reports were processed.
Improved data accuracy and reliability for analytics.
Enabled engineers to focus on more strategic work instead of repetitive tasks.

Negative feedback

Situation:

During my **internship at Beehyv**, I was developing **React components** for a web application. While I ensured the UI was functioning correctly during manual testing, I **did not write automated test cases** using **Cypress** to validate the behavior of the components.

During a **code review**, my manager pointed out the lack of test coverage and emphasized the importance of **writing end-to-end tests to catch regressions early**.

Task:

I needed to **address the feedback** by writing **Cypress test cases** to ensure that all critical user interactions were covered and the components worked as expected.

Action:

- Instead of feeling discouraged, I took the feedback as an **opportunity to learn** about **Cypress testing for React applications**.
- I wrote **automated tests** to verify user interactions, such as **form submissions, button clicks, and API response handling**.
- I ensured the tests covered **edge cases**, such as incorrect user inputs and failed API requests.
- After implementing the tests, I ran them **in CI/CD pipelines** to catch issues before deployment.

Outcome:

My manager appreciated my effort in **quickly improving test coverage**.
The Cypress tests **helped catch bugs early**, reducing the risk of breaking changes in future updates.
This experience reinforced the **importance of automated testing in frontend development**, making me more disciplined in writing tests..

Time when you went above and beyond your job responsibilities.

Situation:

While I was working as a software developer, some interns were struggling with both Django and Spring, particularly in setting up models, handling database migrations, building REST APIs, and managing dependency injections. Since both frameworks were essential for different parts of the project, their difficulties were slowing down overall progress.

Task:

Although mentoring was not part of my core responsibilities, I wanted to help the interns get comfortable with Django and Spring so they could contribute more effectively.

Action:

- I conducted pair programming sessions, guiding them through:
  Django: Models, views, serializers, ORM queries, and database migrations.
  Spring: Dependency injection, REST API development, and integrating with PostgreSQL.
- Helped them debug API issues and explained best practices for performance optimization.
- Created reference guides on handling common errors in both frameworks so they could troubleshoot independently.
- While working on my own tasks, I made sure to allocate time to assist them when they were stuck.

Outcome:

The interns became more confident in Django and Spring, completing tasks with minimal supervision.
API performance improved, and they started handling migrations and dependency management efficiently.
My manager appreciated my initiative, and this experience strengthened my mentorship and leadership skills.

Situation:

During my **internship at Beehyv**, I was working on a **Django-based API** to parse **Excel and CSV files** and store the extracted data into a **database**. The goal was to process large files as efficiently as possible, so I initially designed the API to **load the entire file into memory and process it at once**.

When the project was **75% complete**, I ran tests with **larger datasets** and realized that the approach was **inefficient and prone to memory crashes**. The original goal of **processing the entire file in one go** was flawed, as it couldn't handle real-world data sizes.

Task:

I had to **rethink the approach** and implement a more **scalable solution** that could process large files **without performance issues**.

Action:

- Instead of **loading the entire file at once**, I refactored the API to **process files in chunks**, significantly reducing memory usage.
- I implemented **parallel processing**, allowing multiple chunks to be processed simultaneously for faster execution.
- To validate the new approach, I conducted **performance tests** and ensured data consistency remained intact.
- Communicated the change to the team and **adjusted the project timeline** accordingly.

Outcome:

The **refactored API successfully handled large files** without crashing.
Processing time was reduced by **over 60%**, making the system more scalable.
The project was slightly delayed due to the change, but the final solution was far **more robust and efficient**.

Situation:

If a **new employee** is struggling with technical difficulties, it can impact their **confidence, productivity, and ability to contribute effectively** to the team. I would approach the situation in a **structured and supportive manner** to help them overcome challenges and get up to speed.

Task:

My goal would be to identify the root cause of their difficulty, provide the necessary guidance, and empower them to solve problems independently moving forward.

Action:

1. **Understand the Problem:**
    - I would first listen carefully to what they’re struggling with, whether it's a tool setup issue, debugging a problem, or understanding a technical concept.
    - If needed, I would ask clarifying questions to pinpoint the exact issue.
2. **Provide Hands-on Help:**
    - I would offer to pair program with them if they’re stuck on a coding issue.
    - If the problem is related to project setup, I would guide them through documentation and best practices.
3. **Break It Down & Share Resources:**
    - If the concept is complex (e.g., database migrations in Django or dependency management in Spring), I would break it down into simpler terms and suggest helpful **resources, tutorials, or internal documentation**.
4. **Encourage Independent Problem-Solving:**
    - I would show them debugging techniques rather than just giving them a solution, so they learn to troubleshoot independently in the future.
5. **Follow Up & Foster a Supportive Environment:**
    - I would check back later to see if they are making progress and encourage them to ask questions anytime.

Outcome:

The new employee would gain confidence in handling technical challenges.
They would feel more included and supported, making their onboarding process smoother.
This would improve team collaboration, ensuring everyone can contribute effectively.

Situation:

While working on the **Intentwise Tool** at Beehyv, I was tasked with developing an **API to parse large Excel and CSV files** and store the extracted data in a **database**. Initially, my approach worked fine for **small files**, but when testing with **large datasets**, the API **crashed due to high memory usage and slow processing times**.

Task:

I needed to **redesign the API** to efficiently handle **large file processing** while ensuring **scalability and reliability**.

Action:

- I **analyzed the bottleneck** and realized that **loading the entire file into memory** was causing the crashes.
- To fix this, I **refactored the API** to **process files in chunks** instead of reading them all at once.
- I **implemented parallel processing**, allowing multiple chunks to be processed simultaneously, significantly improving speed.
- I **added error handling** to detect and log corrupt or incomplete data, preventing system failures.
- Finally, I ran **extensive performance tests** to ensure the solution worked under real-world conditions.

Outcome:

Processing speed improved by 60%, and the API handled large files smoothly without crashing.
The new design made the system **scalable and efficient** for future data growth.
This experience strengthened my **problem-solving skills** and taught me the importance of **optimizing for scalability early in the development cycle**.

Situation:

During my **internship at Beehyv**, I was working on a **React-based dashboard** that displayed **real-time data updates** from an API. While implementing the feature, I faced an issue where the **page kept refreshing unnecessarily**, leading to performance issues and a poor user experience.

Despite trying different debugging methods, I couldn't figure out why the component was **rerendering excessively**.

Task:

I needed to **seek help from an experienced teammate** to understand the root cause and implement the correct fix.

Action:

- I approached a **senior developer on my team**, explained the issue, and shared my debugging attempts.
- They **reviewed my code** and pointed out that I had mistakenly placed the **data-fetching logic inside the component's render method**, which was causing continuous re-renders.
- They suggested moving the API call to **useEffect() with an empty dependency array (`[]`)**, ensuring the data was fetched only once when the component mounted.
- After making the change, I **tested the component thoroughly** to confirm that it was updating correctly **without excessive rerenders**.

Outcome:

The **page refresh issue was resolved**, improving performance and user experience.
I **gained a better understanding of React's lifecycle methods**, especially **useEffect() and state management**.
This experience reinforced the importance of **seeking help when needed**, as a fresh perspective can often lead to a quick resolution.

Situation:

During my **internship at Beehyv**, I was assigned to develop a **React-based dashboard** that displayed **real-time data updates**. However, I **did not have hands-on experience with React** before this project. With a **tight deadline**, I had to quickly **learn React and implement the dashboard** while ensuring everything functioned smoothly.

Task:

I needed to **understand React basics, build the required components, and integrate everything properly** within the short timeframe.

Action:

- I **spent extra time learning React** through documentation and tutorials to get up to speed.
- I **built the dashboard step by step**, making sure it displayed data correctly and responded efficiently to user interactions.
- When I faced challenges, I **sought help from teammates** and used debugging tools to fix issues quickly.
- I tested everything thoroughly to ensure the dashboard was **fast and responsive** before submission.

Outcome:

I **completed the project on time**, even though I was learning React along the way.
The dashboard worked smoothly and met all **business requirements**.
This experience **helped me gain confidence in React** and reinforced my ability to **learn new technologies quickly under pressure**.

Situation:

During my **internship at Beehyv**, I was assigned to develop a **React-based dashboard** to display real-time data. Since I **did not have prior hands-on experience with React**, I initially struggled with understanding **component structure, state management, and API integration**.

This lack of experience **slowed down my progress**, and I fell behind the initial timeline.

Task:

Despite the slow start, I needed to **learn React quickly, complete the dashboard on time, and ensure it met all functional requirements**.

Action:

- I dedicated extra time to **learning React fundamentals**, studying documentation, and following tutorials.
- I **broke the project into smaller tasks**, focusing first on the UI, then handling API integration, and finally optimizing the dashboard.
- When I faced challenges, I **reached out to teammates for guidance** and used debugging tools to resolve issues efficiently.
- I tested everything thoroughly to ensure the dashboard was **smooth, responsive, and met all business needs**.

Outcome:

Despite the **initial struggles**, I **completed the project on time** with a fully functional dashboard.
The final dashboard **performed well and met all requirements**.
I **gained hands-on experience with React**, making future projects much easier.

Situation:

During my **internship at Beehyv**, I was tasked with developing an **API to parse Excel and CSV files** and store the extracted data into a **database**. Initially, I implemented a basic solution that **loaded the entire file into memory and processed it all at once**.

However, during **testing with large files**, the API **crashed due to memory overload**, making the approach inefficient and unsuitable for real-world usage.

Task:

I needed to **find a better way to handle large file parsing**, ensuring that the API could process massive datasets **without performance issues or crashes**.

Action:

- I analyzed the issue and realized that **reading large files all at once** was causing excessive memory usage.

- I **refactored the API** to process files in **chunks instead of loading everything at once**, significantly reducing memory consumption.
- I implemented **parallel processing**, allowing multiple chunks to be processed simultaneously, making the system more efficient.
- Thoroughly tested the optimized solution to ensure **data integrity and system stability**.

Outcome:

The **new approach improved processing speed by 60%** and eliminated memory crashes.
The API could now **handle large Excel and CSV files smoothly**, making it **scalable for future use**.
This experience reinforced the importance of **testing early, optimizing performance, and designing scalable solutions**.

During my **internship at Beehyv**, I was working on an **API to parse Excel and CSV files** and store the extracted data into a **database**. As the deadline approached, I encountered **unexpected performance issues** when testing with large files—**the API was crashing due to high memory usage**, making it unusable in real-world scenarios.

With only a **short time left before the deadline**, I felt under pressure because the current approach wasn't scalable, and I needed to **quickly pivot to a better solution**.

Task:

I needed to **fix the performance issues quickly** and ensure the API could handle **large file processing efficiently** while still meeting the deadline.

Action:

- Instead of **loading the entire file into memory**, I quickly **redesigned the approach** to **process files in chunks**, reducing memory consumption.
- To speed up execution, I **implemented parallel processing**, allowing multiple chunks to be processed simultaneously.
- I prioritized **core functionality** over additional validations to **meet the deadline** while keeping the API reliable.
- After making these optimizations, I **ran final tests** to ensure the API worked smoothly under real-world conditions.

Outcome:

The API **successfully processed large files** without crashing, making it scalable and efficient.
Despite the **last-minute pivot**, I was able to **meet the deadline** with a **working and optimized solution**.
This experience taught me how to **stay calm under pressure, adapt quickly, and focus on solutions rather than problems**.

Situation: While working on **multiple tasks** at Beehyv, I follow a **customer-focused approach** to prioritization. My first priority is always to **address any issue that is blocking the customer**, ensuring minimal disruption to their workflow. After resolving **critical blockers**, I move on to **feature enhancements** for existing functionality, and finally, I focus on **developing new features** that provide long-term value.

For example, while working on the **Data Catalog Tool**, I had three tasks at the same time:
Fixing a bug in an existing connector that was impacting a live customer.
Enhancing a feature in another connector to improve performance.
Developing a new database connector to support additional data sources.

Action Plan for Prioritization:

🔹 **Step 1: Resolve the Blocking Issue First**

- I immediately addressed the **bug in the existing connector** since it was **directly affecting a customer** and preventing them from using the tool.
- Debugged, tested, and released a fix quickly to restore normal operations.

🔹 **Step 2: Implement Feature Enhancements**

- Once the blocking issue was fixed, I focused on **enhancing an existing connector** to improve efficiency and performance.
- Since customers were already using it, this improvement **added value without disrupting workflows**.

🔹 **Step 3: Develop New Features**

- Finally, I worked on **creating a new database connector**, which had long-term benefits but **was not impacting current users**.
- Ensured proper **design and testing** to integrate it smoothly into the system.

Outcome:

The **customer-facing bug was resolved first**, preventing major disruptions.

The **feature enhancement was successfully implemented**, improving efficiency for existing users.

The **new database connector was developed** without affecting high-priority tasks.