Usage Tab in Automation Playbooks

ATLASSIAN

Sri Sindhu Veerathu

Software Engineering Intern

19 May 25 - 11 July 25

Team- ITOps Phantom

Backend Development of Usage Tab

Why is Playbook Usage Tab Needed?

In Jira Service Management (JSM), we have a feature known as Playbooks. The insights provided by the usage tab would assist administrators in organizing the playbooks more effectively, allowing for the removal of unused playbooks and the enhancement of the most frequently used ones accordingly.

Alignment with Broader Objective & Impact

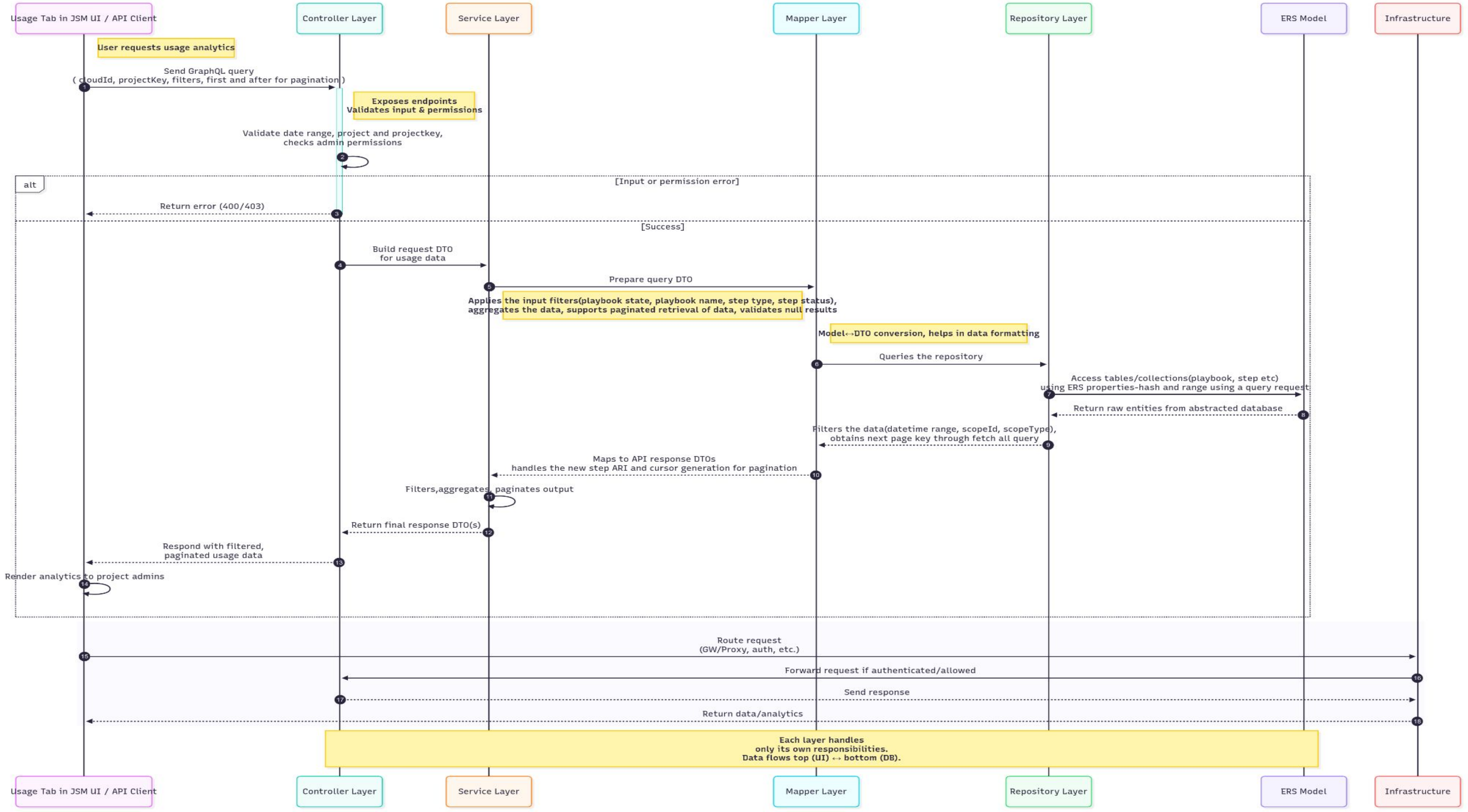
This project aligns with the broader objective of increasing the utilization of JSM Automation, as the playbook directly contributes to this goal. -

Increase the Percentage of Premium & Enterprise tenants (with more than 50 JSM agents) using Orchestration from 24% → 50% [0.7 = 46%]On track - 0.8

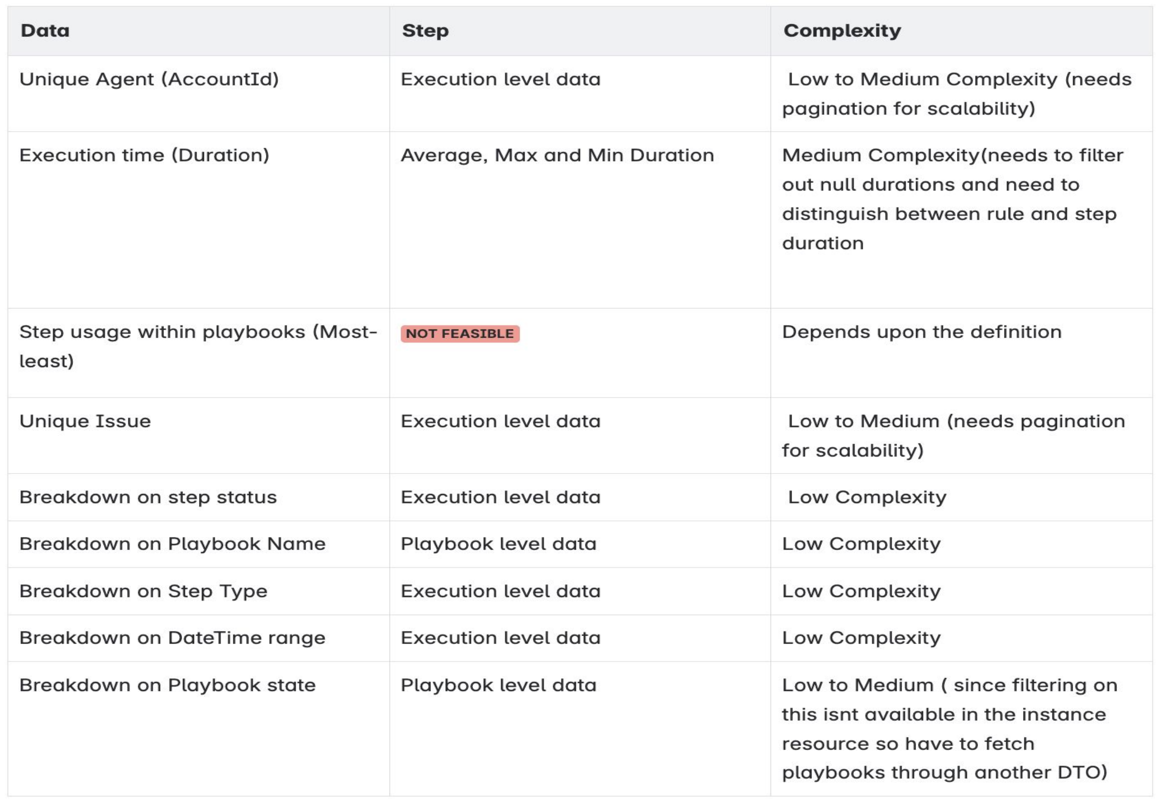
MY DELIVERABLES

- ✓ Document the design of Graphql queries for the Usage Tab.
- ✓ Deliver backend to handle Usage Tab in the Playbook Admin View.
- ✓ Implement certain filters in the APIs.
- ✓ Create integration and unit tests to ensure feature work as intended.
- ✓ Collaborate with frontend and product team for visualisation on the UI.

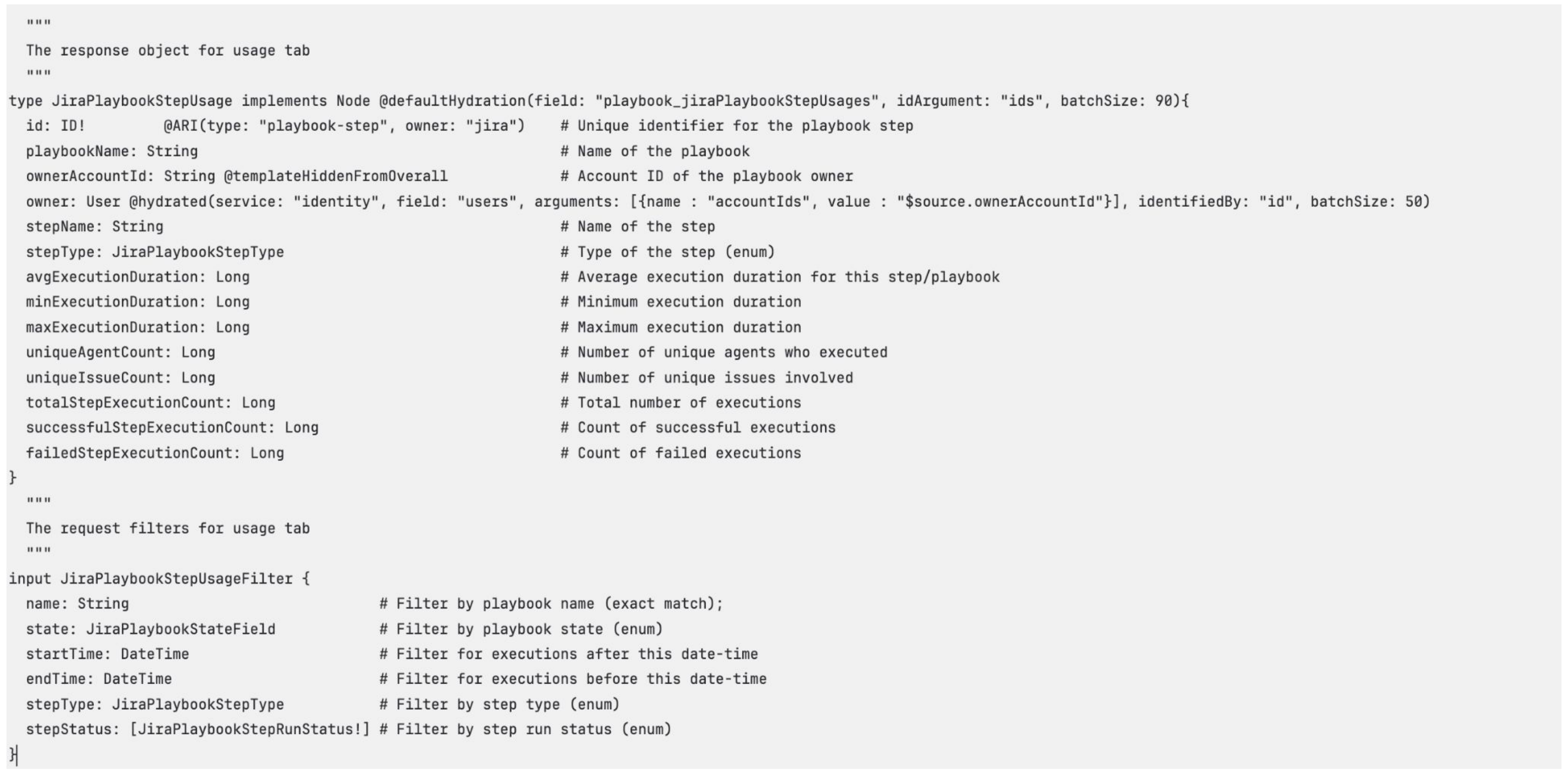
SEQUENCE DIAGRAM : PLAYBOOK USAGE BACKEND FLOW


Usage Tab in JSM UI / API Client
Controller Layer
Service Layer
Mapper Layer
Repository Layer
ERS Model
Infrastructure
User requests usage analytics
1 Send GraphQL query (cloudId, projectKey, filters, first and after for pagination)
Exposes endpoints Validates input & permissions
2 Validate date range, project and projectkey, checks admin permissions
alt
[Input or permission error]
3 Return error (400/403)
[Success]
4 Build request DTO for usage data
5 Prepare query DTO
Applies the input filters(playbook state, playbook name, step type, step status), aggregates the data, supports paginated retrieval of data, validates null results
Model↔DTO conversion, helps in data formatting
6 Queries the repository
7 Access tables/collections(playbook, step etc) using ERS properties-hash and range using a query request
8 Return raw entities from abstracted database
9 Filters the data(datetime range, scopeId, scopeType), obtains next page key through fetch all query
10 Maps to API response DTOs handles the new step ARI and cursor generation for pagination
11 Filters,aggregates, paginates output
12 Return final response DTO(s)
13 Respond with filtered, paginated usage data
14 Render analytics to project admins
15 Route request (GW/Proxy, auth, etc.)
16 Forward request if authenticated/allowed
17 Send response
18 Return data/analytics
Each layer handles only its own responsibilities. Data flows top (UI) ↔ bottom (DB).

IMPLEMENTATION



Data	Step	Complexity
Unique Agent (AccountId)	Execution level data	Low to Medium Complexity (needs pagination for scalability)
Execution time (Duration)	Average, Max and Min Duration	Medium Complexity(needs to filter out null durations and need to distinguish between rule and step duration
Step usage within playbooks (Most-least)	NOT FEASIBLE	Depends upon the definition
Unique Issue	Execution level data	Low to Medium (needs pagination for scalability)
Breakdown on step status	Execution level data	Low Complexity
Breakdown on Playbook Name	Playbook level data	Low Complexity
Breakdown on Step Type	Execution level data	Low Complexity
Breakdown on DateTime range	Execution level data	Low Complexity
Breakdown on Playbook state	Playbook level data	Low to Medium (since filtering on this isnt available in the instance resource so have to fetch playbooks through another DTO)



```
  """
  The response object for usage tab
  """
type JiraPlaybookStepUsage implements Node @defaultHydration(field: "playbook_jiraPlaybookStepUsages", idArgument: "ids", batchSize: 90){
  id: ID!         @ARI(type: "playbook-step", owner: "jira")    # Unique identifier for the playbook step
  playbookName: String                                          # Name of the playbook
  ownerAccountId: String @templateHiddenFromOverall             # Account ID of the playbook owner
  owner: User @hydrated(service: "identity", field: "users", arguments: [{name : "accountIds", value : "$source.ownerAccountId"}], identifiedBy: "id", batchSize: 50)
  stepName: String                                              # Name of the step
  stepType: JiraPlaybookStepType                                # Type of the step (enum)
  avgExecutionDuration: Long                                    # Average execution duration for this step/playbook
  minExecutionDuration: Long                                    # Minimum execution duration
  maxExecutionDuration: Long                                    # Maximum execution duration
  uniqueAgentCount: Long                                        # Number of unique agents who executed
  uniqueIssueCount: Long                                        # Number of unique issues involved
  totalStepExecutionCount: Long                                 # Total number of executions
  successfulStepExecutionCount: Long                            # Count of successful executions
  failedStepExecutionCount: Long                                # Count of failed executions
}
  """
  The request filters for usage tab
  """
input JiraPlaybookStepUsageFilter {
  name: String                          # Filter by playbook name (exact match);
  state: JiraPlaybookStateField         # Filter by playbook state (enum)
  startTime: DateTime                   # Filter for executions after this date-time
  endTime: DateTime                     # Filter for executions before this date-time
  stepType: JiraPlaybookStepType        # Filter by step type (enum)
  stepStatus: [JiraPlaybookStepRunStatus!] # Filter by step run status (enum)
}
```

1.Feasibility of Features in Usage Tab

Collaborated with frontend and product on what kind of filters would be feasible to implement in the usage tab.

2.GraphQL Schema of Usage Tab

Collaborated with frontend regarding what aggregations and response variables should be present in the response object and in what format, deciding which would offer the best insights to the user.

3.Code implementation

Implemented the required features in close collaboration with the team, ensuring code is clean, maintainable, and follows established standards. All changes were peer-reviewed before merging.

4.Unit and integration tests

Added comprehensive unit and integration tests to validate core functionality and interactions. Tests are simple, reliable, and follow the Arrange-Act-Assert pattern for clarity and maintainability

LEARNINGS

01

Aggregations in Memory

- Due to the limitations of the ERS properties, I had to find the best suitable index for fetching the required data from the abstracted database and perform all the aggregation operations in memory.

02

Need for a new ARI

- The present ARI's were not suitable for the proper identification of a log in the usage tab, so we implemented a new Playbook Step ARI which would uniquely identify and reference a specific step within a playbook in Jira - satisfying usage tab requirements and future needs too.

03

Code Quality, Documentation & Testing

- Understood the importance of readability of code and the amount of work that goes into writing and documenting a maintainable piece of code along with comprehensive testing.

04

Cross Team Collaboration

- Working closely with my mentor, buddy , interns and other teams helped me realize the importance of aligning requirements, resolve blockers. Maintaining this iterative feedback loop would be one of my biggest learnings - it helped in resolving issues and progress at a steady pace.

OBSTACLES INTO OPPORTUNITIES

01

Debugging Locally

- Faced issues in running and debugging tests locally
- **Overcame** this by delving deeper into confluence pages and slack threads and added a Byte-Buddy agent to the VM options etc to fix this issue

02

AGG PR for GraphQL Schema

- Faced hydration and conflict issues while raising the schema pr through AGG pipeline
- **Overcame** this by collaborating with other interns and team members to figure out the correct procedure.

03

Integration and Staging Testing

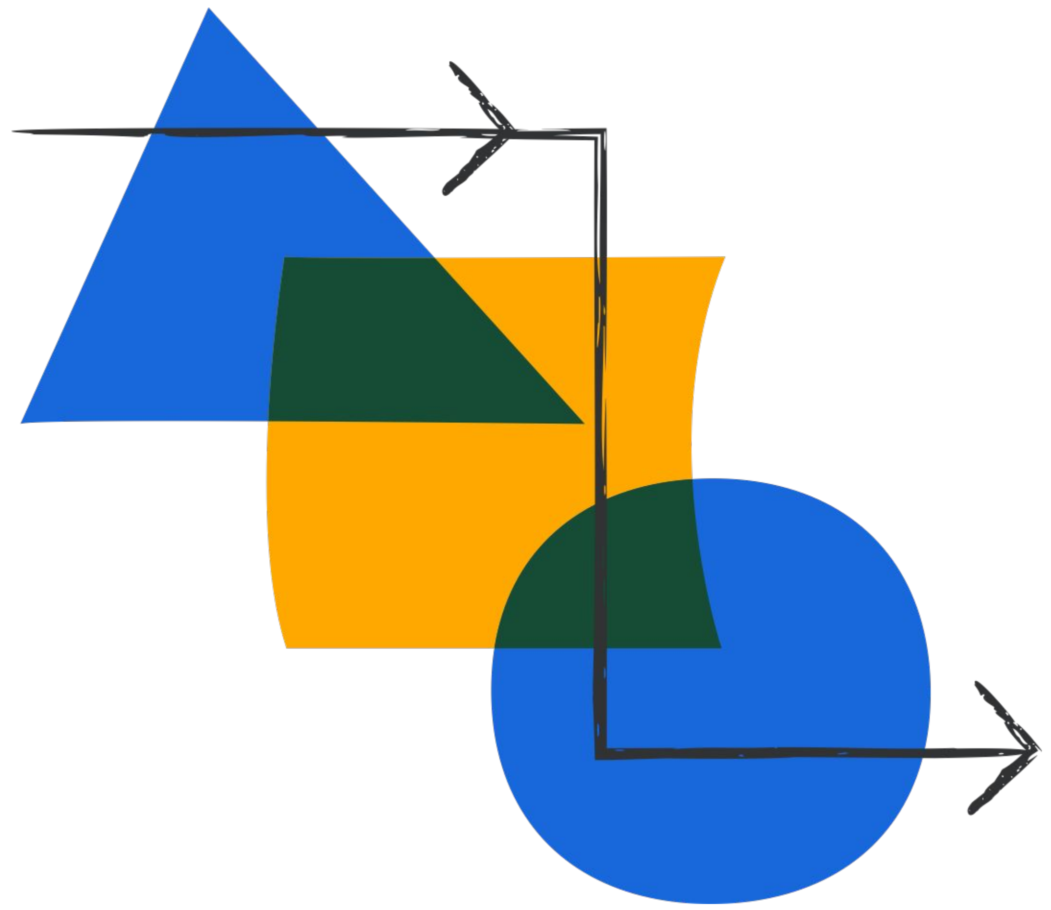
- Faced errors while building integration tests and also some errors while staging
- **Overcame** this by being more comprehensive while debugging and this in turn made me much more confident and knowledgeable about my code.

ALIGNING WITH ATLASSIAN VALUES

Documented onboarding experiences, technical challenges, and solutions in Confluence to support future interns and team members.	Proactively collaborated with mentor, buddy, and cross-functional teams, participating in standups and sprint planning to align on goals and resolve blockers.	Engaged in Intern Bingo Challenge and backend development being my passion, I ensured meaningful impact for customers and the team.	Took ownership of code quality, improved onboarding documentation, and initiated cross-team syncs for better collaboration.	Enhanced user experience through the Playbook Usage Tab, iterated on schemas and design docs, and deprecated non-value-adding code to deliver the best solutions for users.
Open company, no bullshit	Play, as a team	Be the change you seek	Don't #@!% the customer	Build with heart and balance

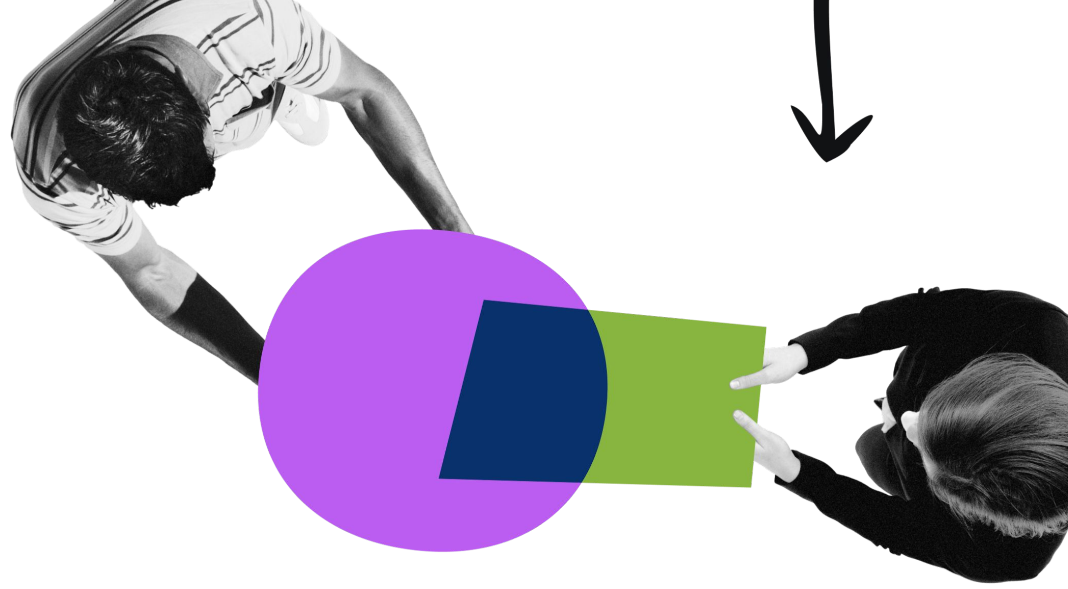
WHAT’S NEXT

- Add an Issue Type Filter to identify the most effective playbooks/steps per issue type (POC proposed).
- Display top 5 Unique Agents in the Usage Tab (per previous discussion).
- Enhance backend with endpoints for graphical usage insights (trends, most/least used playbooks, bar graphs).Aligning with the Usage Tab in Automation.



REFERENCES

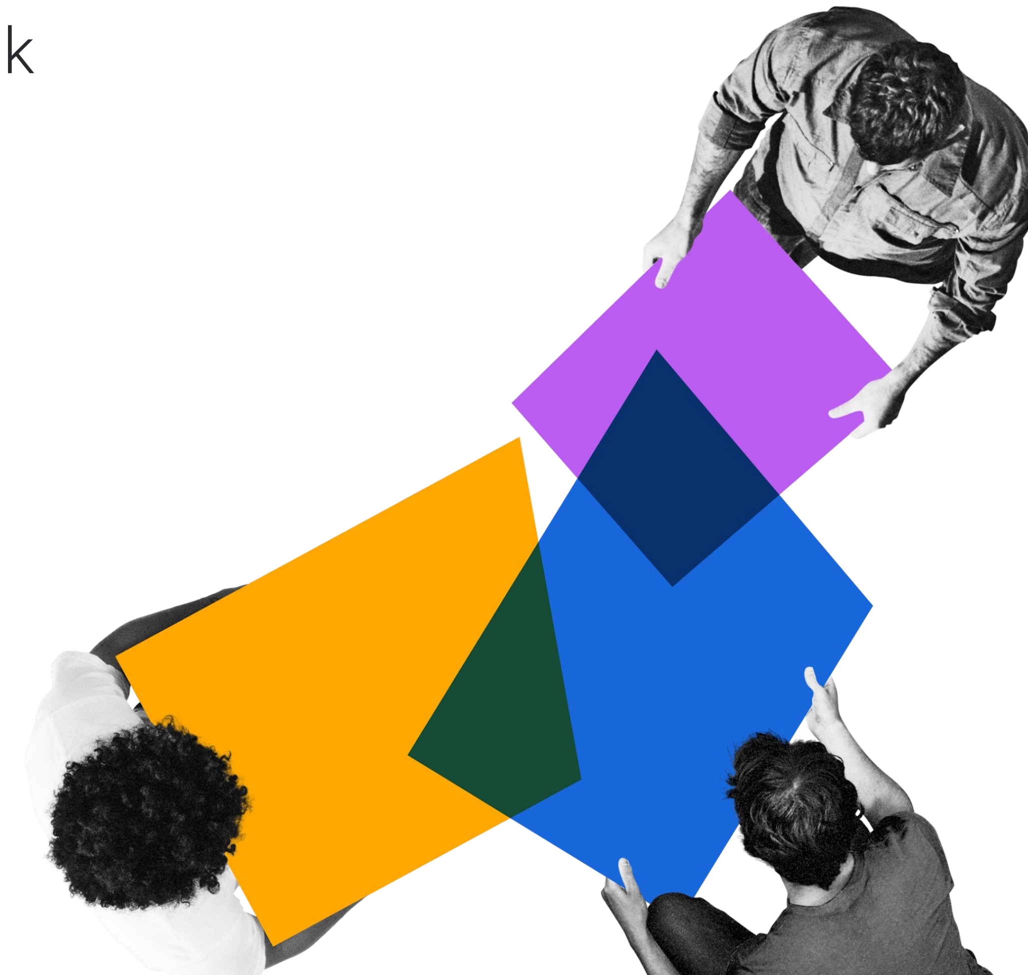
- My Internship Documentation
- Intern Buddy Diary
- Usage Tab Feasibility
- GraphQL Schema and Code Flow for Usage Tab
- New ARI Documentation



GRATITUDE & KUDOS !!

A massive shoutout to all the people I have worked and collaborated with, especially Gaurav, Harshita, Sandeep and Snithik and my team Phantom :)

These two months have been so special 🫶



ATLASSIAN

Thank you!