09/28/2024

METRIC DASHBOARD

Front end development timeline during the course of a product development internship.

FRONT END DEVELOPED BY
Colin Berry

TABLE OF CONTENTS

My team and I were assigned to create a scalable platform that would track Agile metrics of an investment firm's technology development department. My objective was to fetch compiled data and display the data in a comprehensive yet easy-to-understand form.

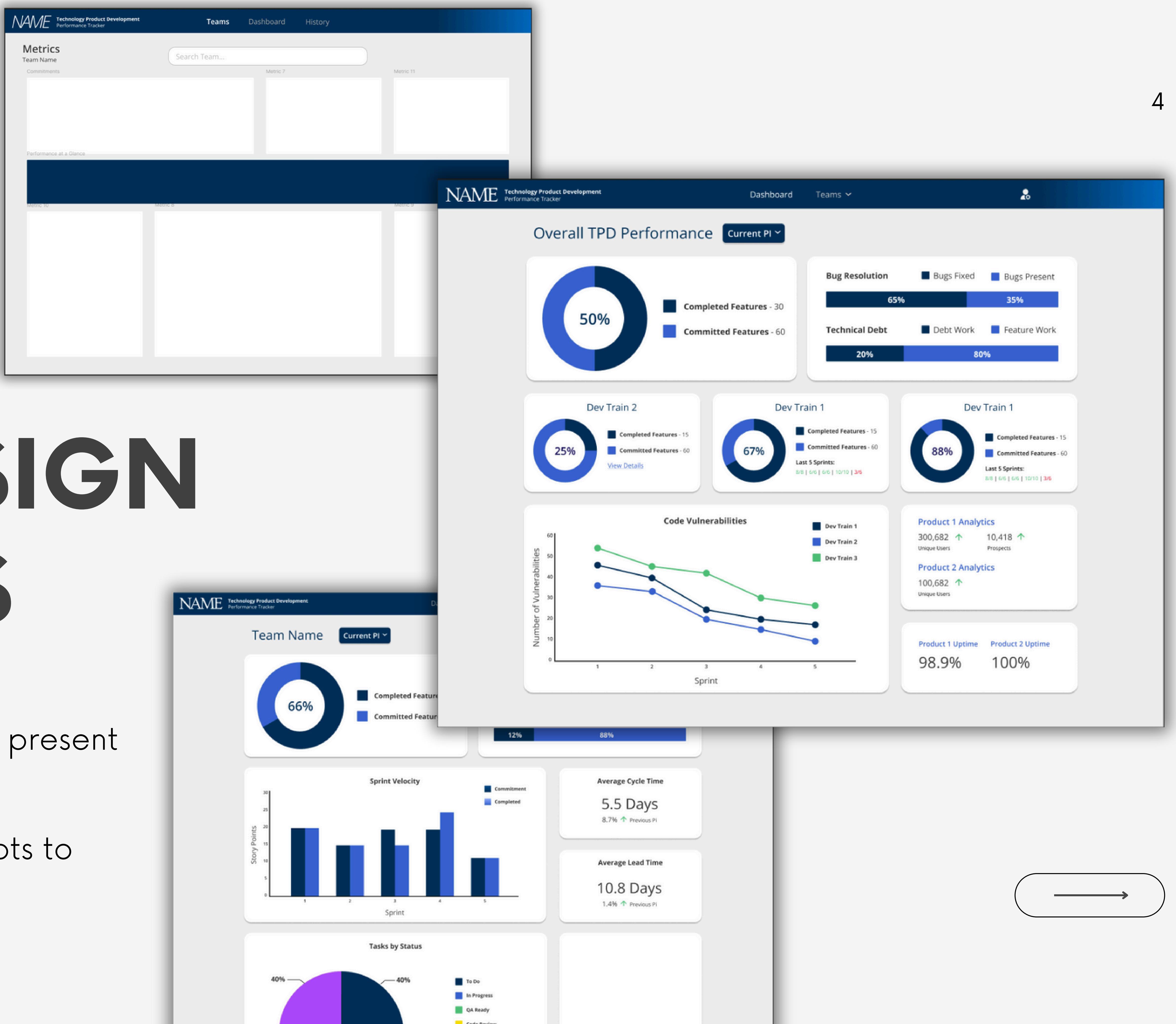
MISSION

Tools Used to Develop Front End

- Figma
- HTML
- JavaScript
- Vite
- React
- Tailwind CSS
- Axios
- Tremor UI

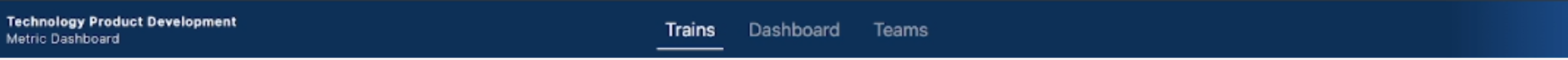
04

FIGMA DESIGN CONCETPS



I began by developing iterative dashboard designs using Figma to present to shareholders.

By presenting these design concepts to shareholders, I gained a better understanding of their vision and expectations.

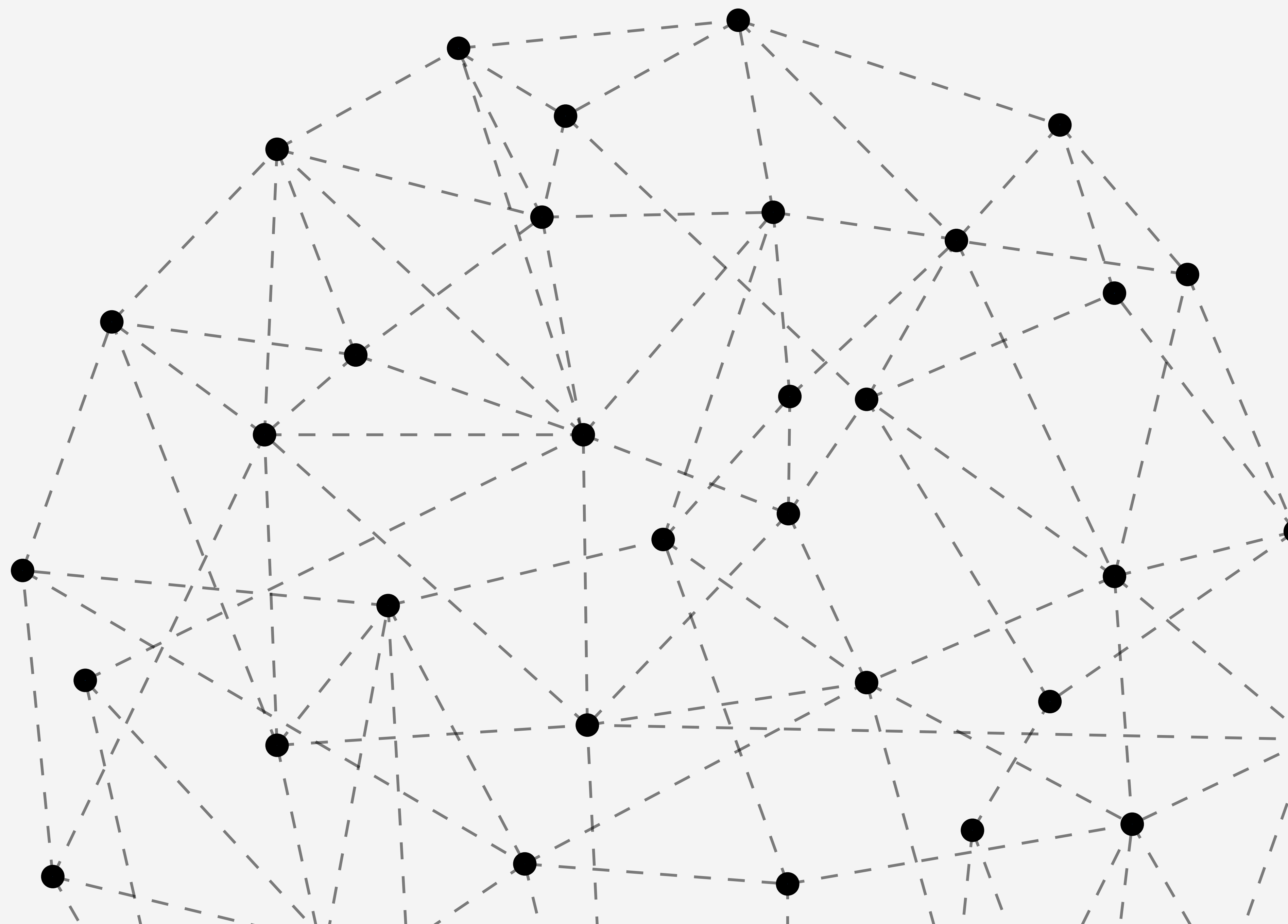


05

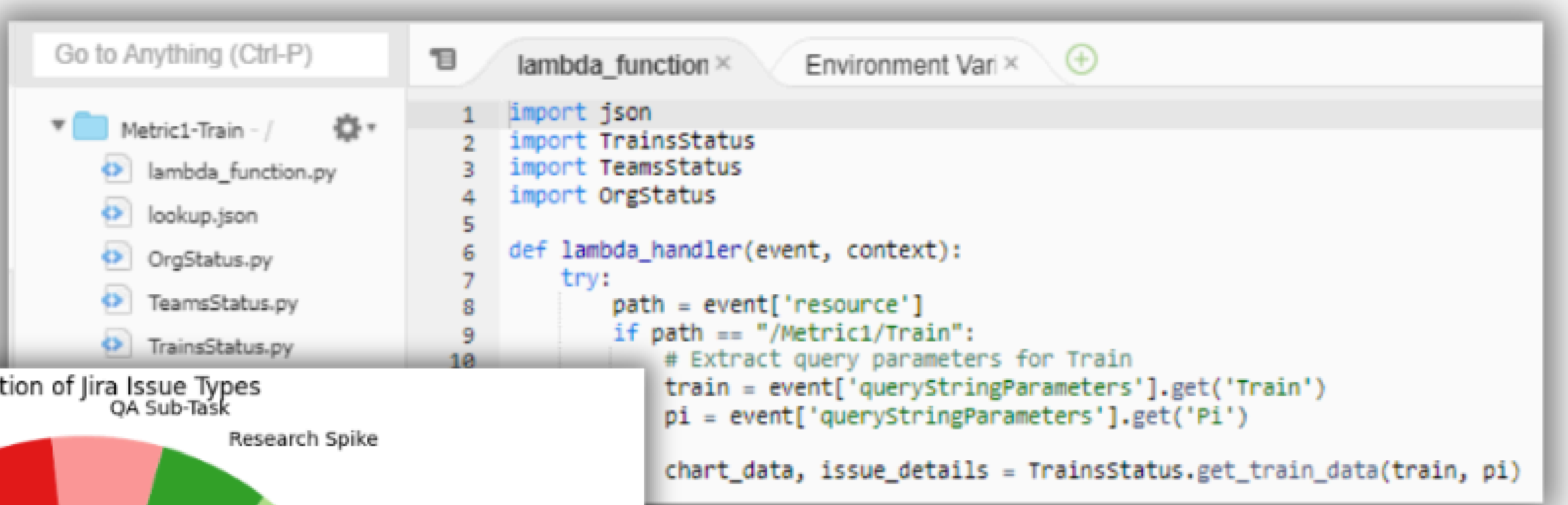
SITE NAVIGATION

Once our shareholders were enthusiastic about preliminary design concepts, I began building our dashboard's navigation bar.

This bar included a link to the main dashboard that would house org-level metrics and two drop-down menus for release train and team links. I used React Router to connect the links with their respectable metric pages.



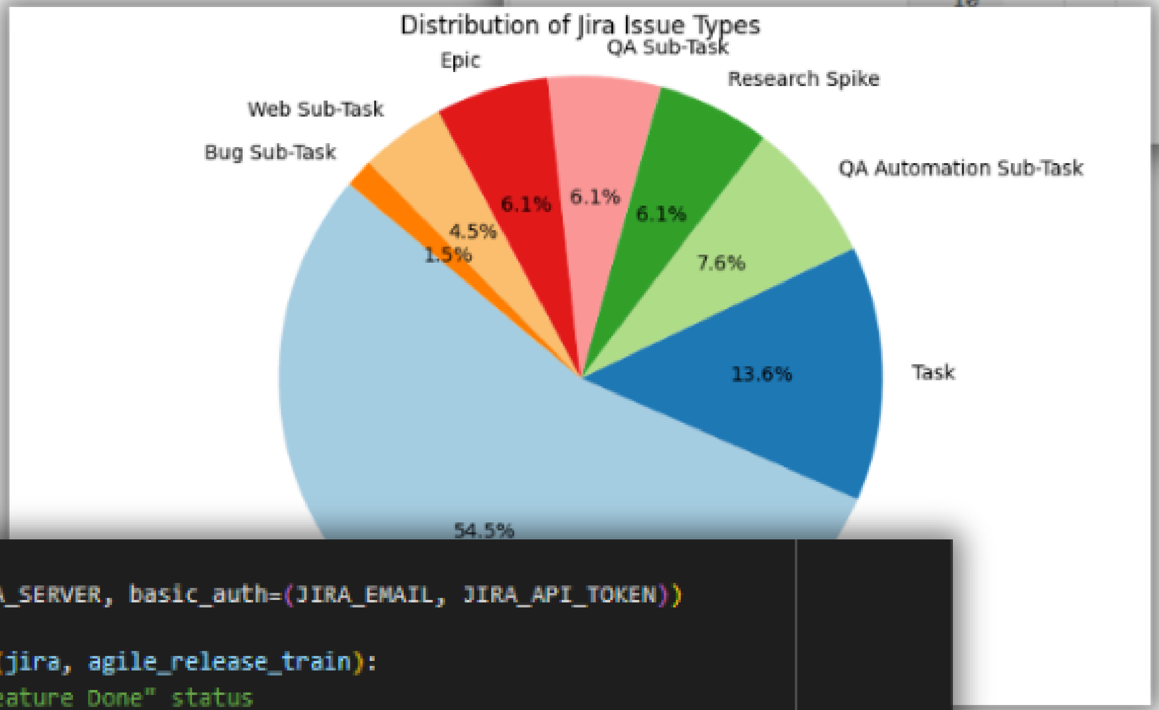
06

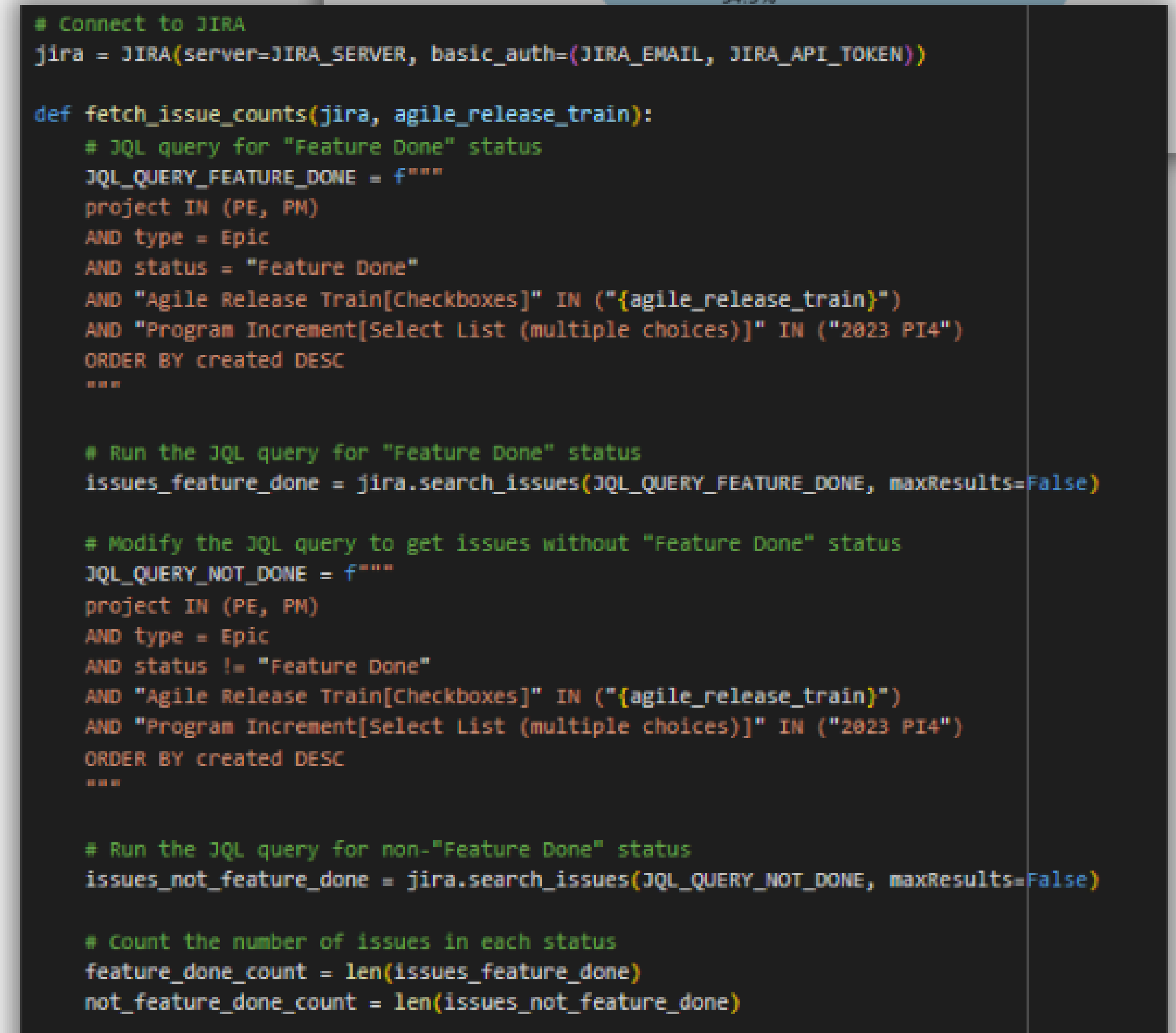


```
import json
import TrainsStatus
import TeamsStatus
import OrgStatus

def lambda_handler(event, context):
    try:
        path = event['resource']
        if path == "/Metric1/Train":
            # Extract query parameters for Train
            train = event['queryStringParameters'].get('Train')
            pi = event['queryStringParameters'].get('Pi')

            chart_data, issue_details = TrainsStatus.get_train_data(train, pi)
```





```
# Connect to JIRA
jira = JIRA(server=JIRA_SERVER, basic_auth=(JIRA_EMAIL, JIRA_API_TOKEN))

def fetch_issue_counts(jira, agile_release_train):
    # JQL query for "Feature Done" status
    JQL_QUERY_FEATURE_DONE = f"""
    project IN (PE, PM)
    AND type = Epic
    AND status = "Feature Done"
    AND "Agile Release Train[Checkboxes]" IN ("{agile_release_train}")
    AND "Program Increment[Select List (multiple choices)]" IN ("2023 PI4")
    ORDER BY created DESC
    """

    # Run the JQL query for "Feature Done" status
    issues_feature_done = jira.search_issues(JQL_QUERY_FEATURE_DONE, maxResults=False)

    # Modify the JQL query to get issues without "Feature Done" status
    JQL_QUERY_NOT_DONE = f"""
    project IN (PE, PM)
    AND type = Epic
    AND status != "Feature Done"
    AND "Agile Release Train[Checkboxes]" IN ("{agile_release_train}")
    AND "Program Increment[Select List (multiple choices)]" IN ("2023 PI4")
    ORDER BY created DESC
    """

    # Run the JQL query for non-"Feature Done" status
    issues_not_feature_done = jira.search_issues(JQL_QUERY_NOT_DONE, maxResults=False)

    # Count the number of issues in each status
    feature_done_count = len(issues_feature_done)
    not_feature_done_count = len(issues_not_feature_done)
```

UNDERSTANDING THE BACKEND

The backend consisted of several Python AWS Lambdas that would fetch the department's Jira ticket data with Jira's API using JQL.

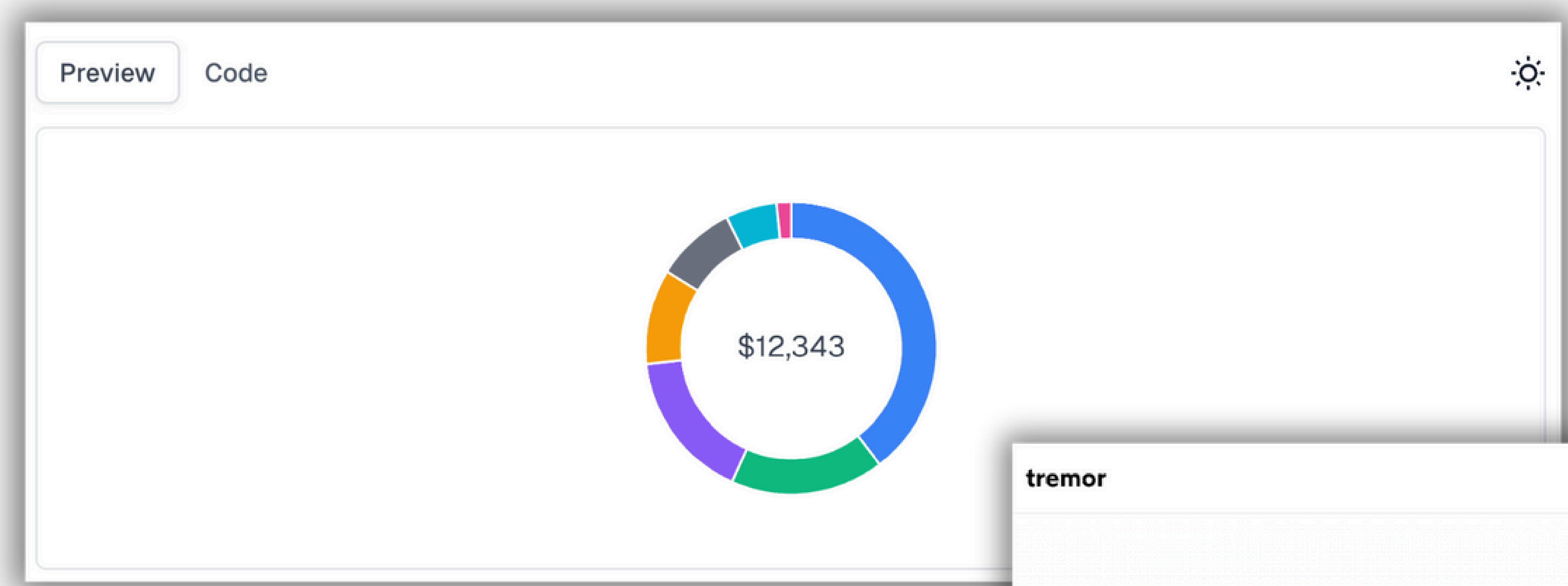
The primary metric gathered focused on feature status specified by the team's name (including the combination of many teams) and program increment. Once this data was translated into JSON, an AWS API Gateway URL could be used to fetch it on the front end.

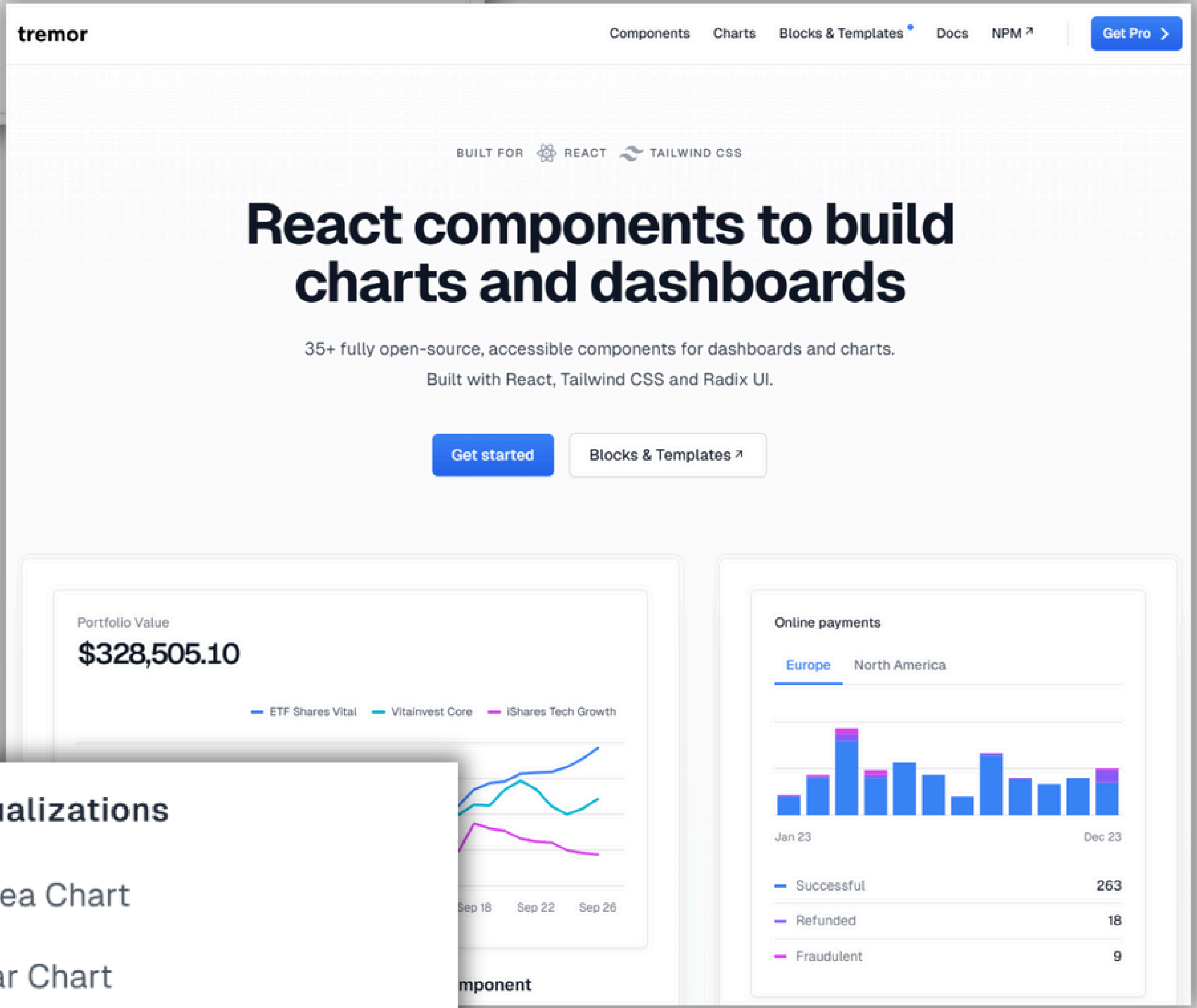
07

DATA VISUALIZATION USING TREMOR UI

For data visualization, I used Tremor UI, a React library that makes building interactive dashboards simple, clear, and informative. Its customizable charts present complex data in a clean, responsive layout, providing insights at a glance.

This intuitive and transparent approach to data visualization was a key requirement from our stakeholders.




tremor

Components Charts Blocks & Templates Docs NPM ↗ Get Pro >

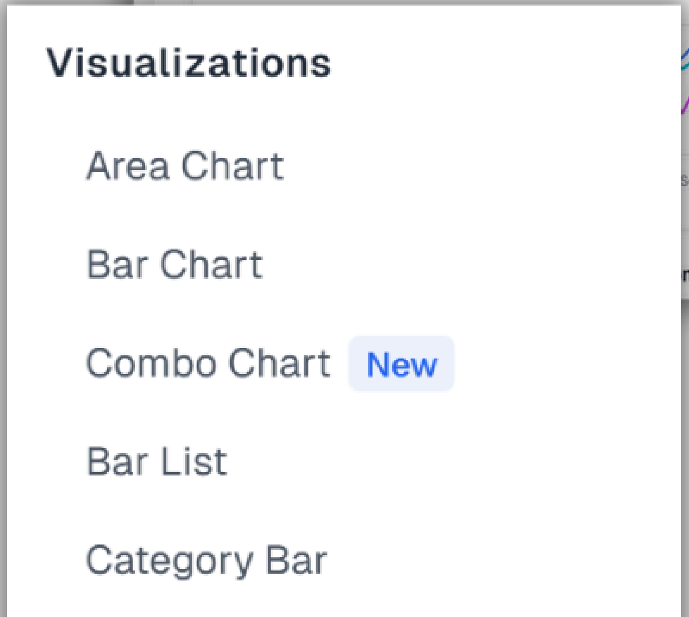
BUILT FOR REACT TAILWIND CSS

React components to build charts and dashboards

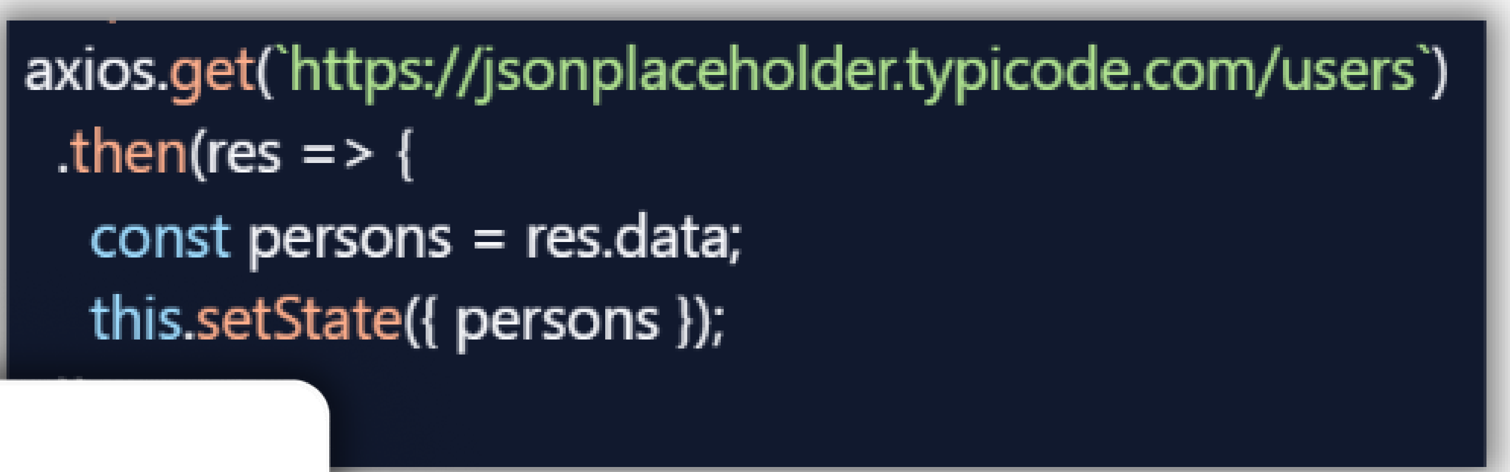
35+ fully open-source, accessible components for dashboards and charts. Built with React, Tailwind CSS and Radix UI.

Get started | Blocks & Templates ↗

Successful	263
Refunded	18
Fraudulent	9

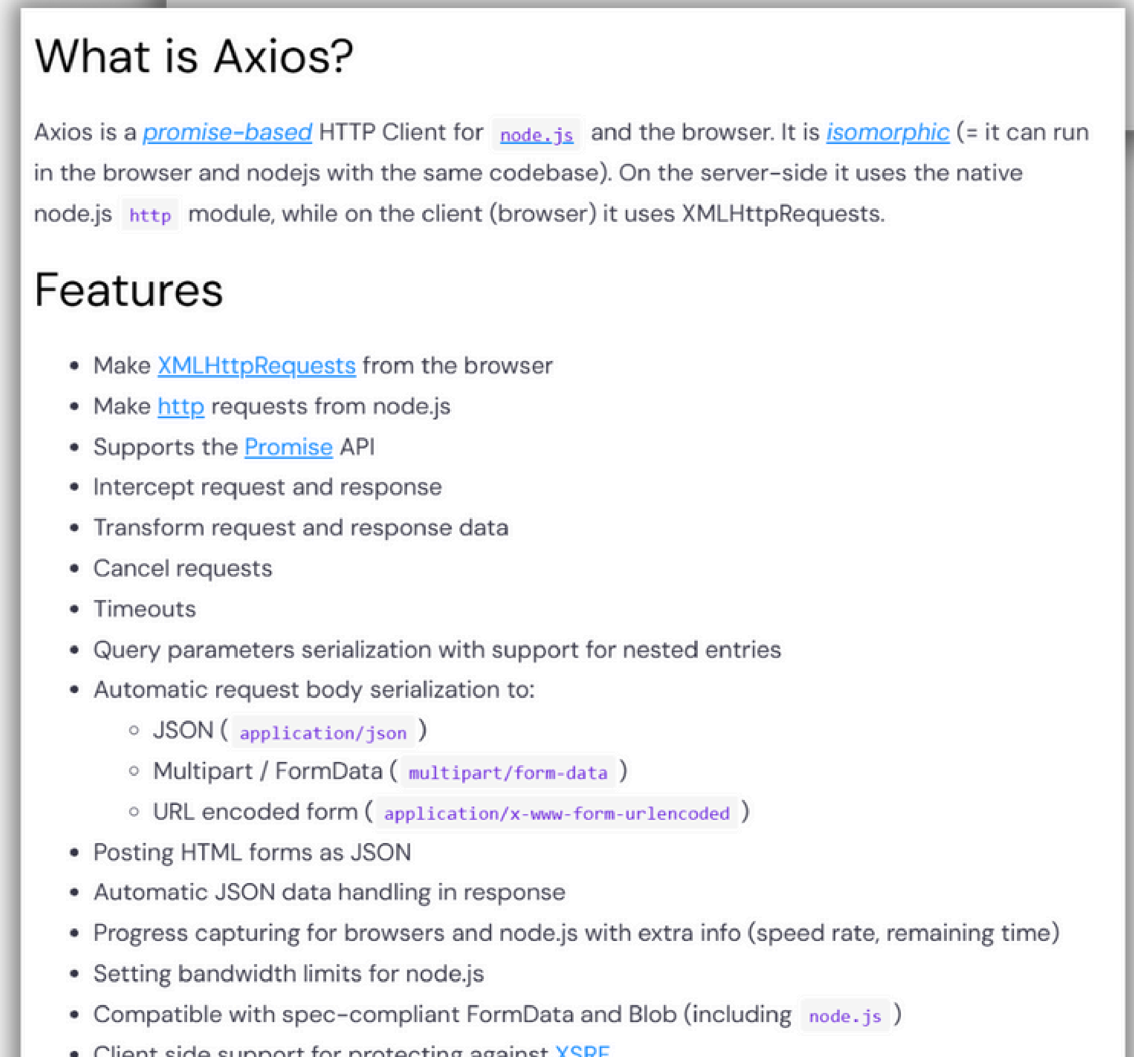


08



```
axios.get(`https://jsonplaceholder.typicode.com/users`)
  .then(res => {
    const persons = res.data;
    this.setState({ persons });
```

AXIOS



What is Axios?

Axios is a *promise-based* HTTP Client for `node.js` and the browser. It is *isomorphic* (= it can run in the browser and nodejs with the same codebase). On the server-side it uses the native node.js `http` module, while on the client (browser) it uses XMLHttpRequests.

Features

- Make XMLHttpRequests from the browser
- Make http requests from node.js
- Supports the Promise API
- Intercept request and response
- Transform request and response data
- Cancel requests
- Timeouts
- Query parameters serialization with support for nested entries
- Automatic request body serialization to:
  - JSON (`application/json`)
  - Multipart / FormData (`multipart/form-data`)
  - URL encoded form (`application/x-www-form-urlencoded`)
- Posting HTML forms as JSON
- Automatic JSON data handling in response
- Progress capturing for browsers and node.js with extra info (speed rate, remaining time)
- Setting bandwidth limits for node.js
- Compatible with spec-compliant FormData and Blob (including `node.js`)
- Client side support for protecting against XSRF

FETCHING DATA USING AXIOS

Once the backend API was available, I used Axios, a promise-based HTTP client, to handle API requests. Its simple syntax and built-in error handling made it easy to fetch and manage data efficiently.

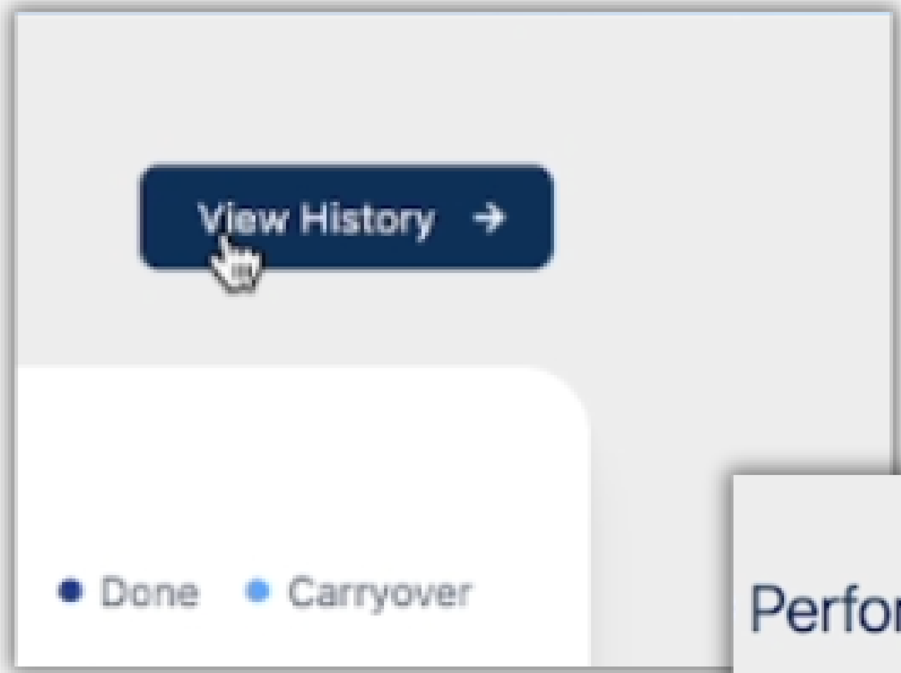
This streamlined integration allowed for reliable communication between the frontend and backend, ensuring a smooth user experience.

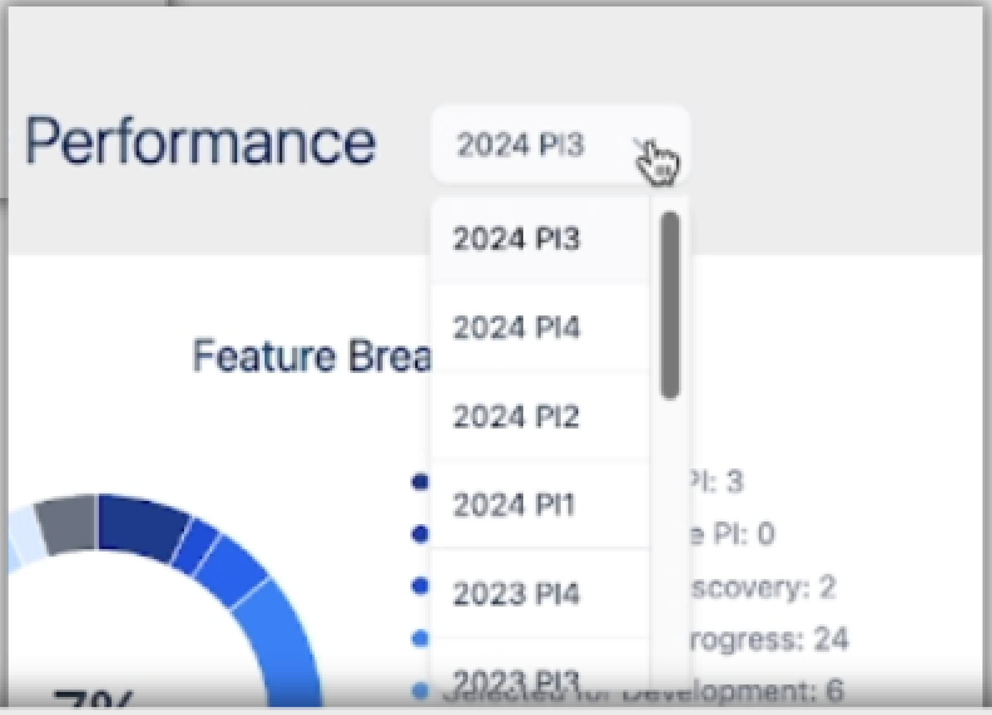
09

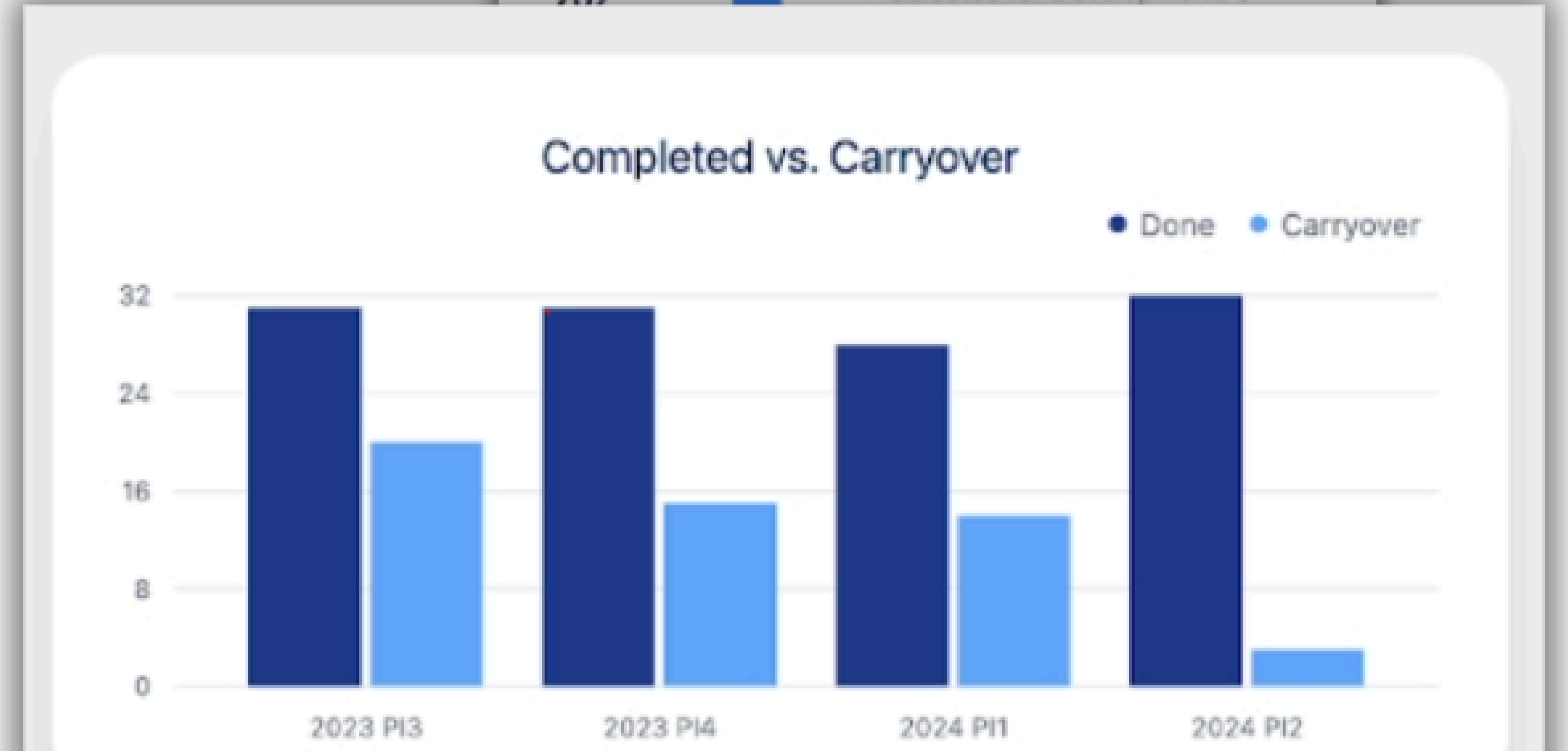
FEATURE EXPANSION

Once the backend and front end were connected, I implemented key stakeholder-requested features, including a history tab, program increment selection, and various metric charts.

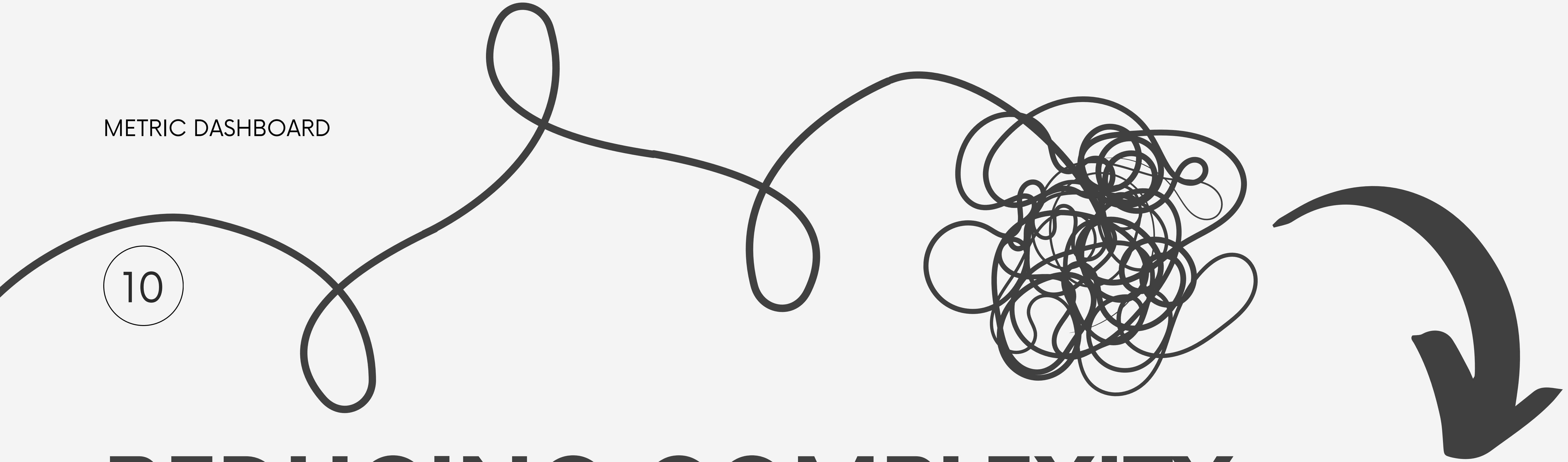
These additions were pivotal in metric transparency, allowing users to view specific periods or understand trends over time.







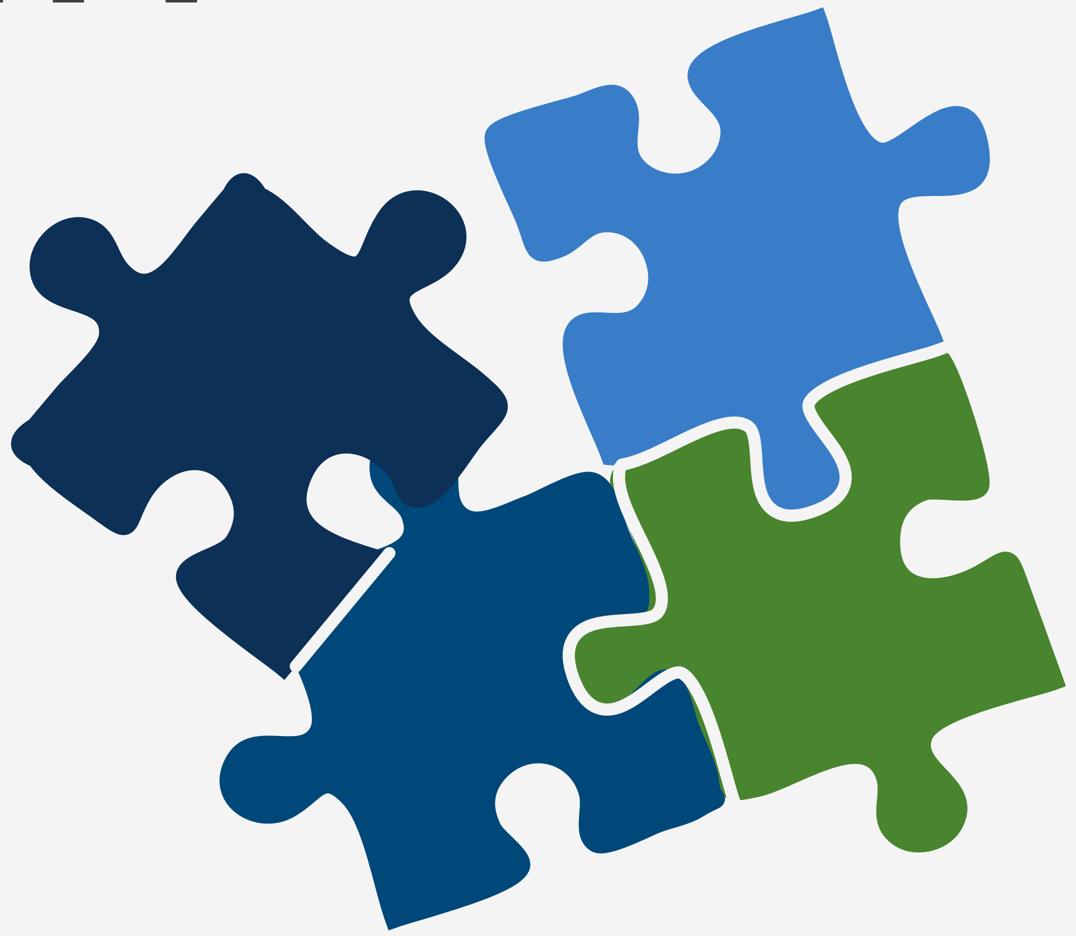
10



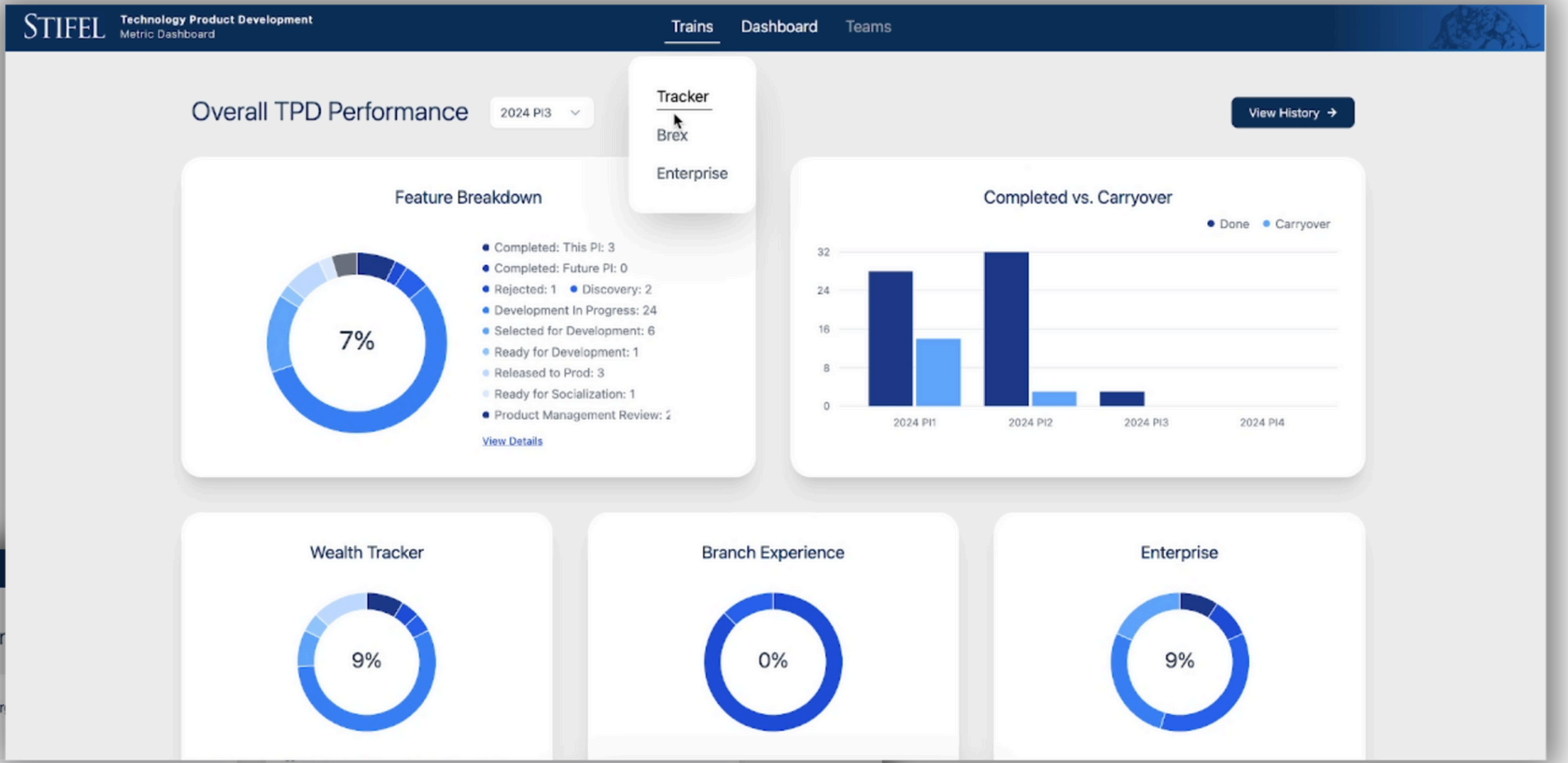
REDUCING COMPLEXITY

I simplified the front end by removing React Router and using useState to control which content displays on a single page instead of managing multiple routes.

This reduced code complexity, improved performance, and created a smoother user experience while maintaining all key features.



DEMO



Overall TPD Performance 2024 PI3

Feature Breakdown

- Completed: This PI: 3
- Completed: Future PI: 0
- Rejected: 1 ● Discovery: 2
- Development In Progress: 24
- Selected for Development: 6
- Ready for Development: 1
- Released to Prod: 3
- Ready for Socialization: 1
- Product Management Review: 2

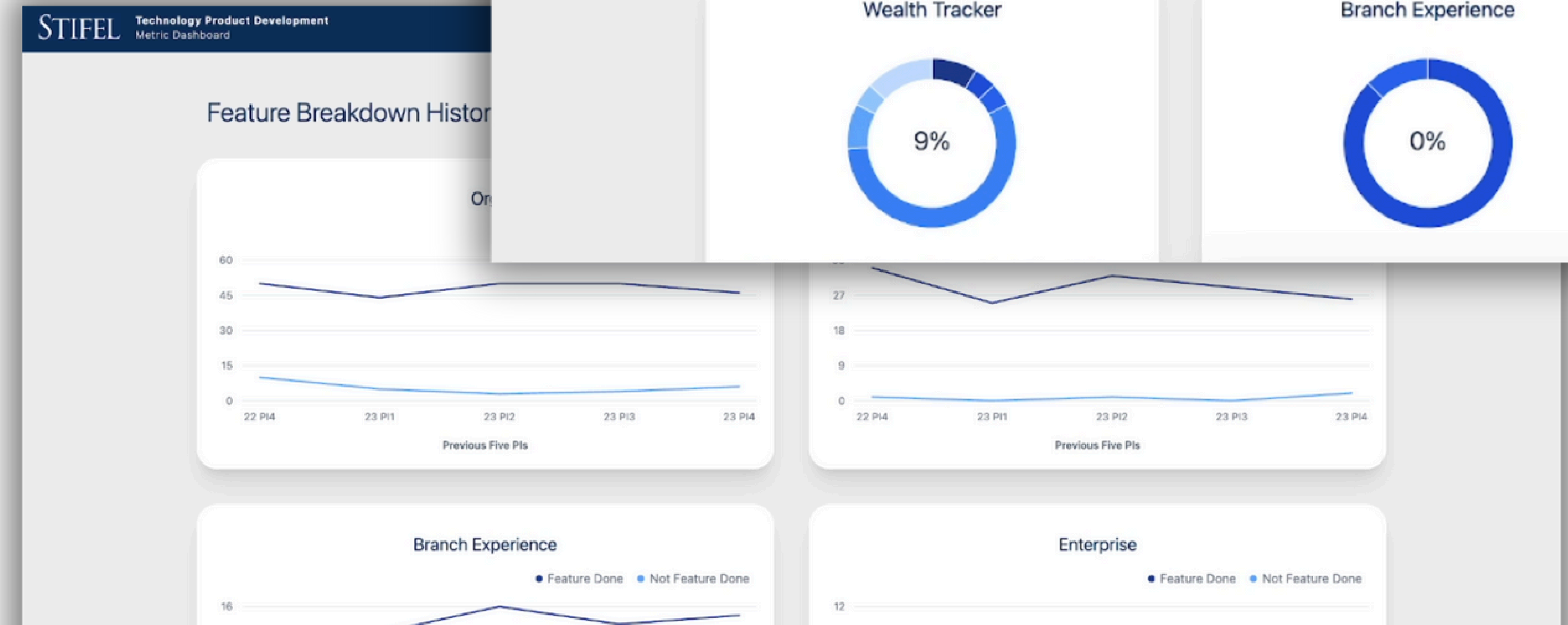
View Details

Completed vs. Carryover

Wealth Tracker

Branch Experience

Enterprise



Feature Breakdown Histor

Branch Experience

Enterprise

HTTPS://YOUTU.BE/5ZQTUSA4D_I