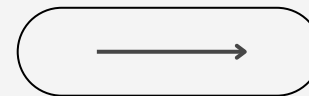


# METRIC DASHBOARD

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



FRONT END DEVELOPED BY  
Colin Berry

# TABLE OF CONTENTS

04

FIGMA DESIGN CONCEPTS

Preliminary design concepts were created using Figma to understand vision.

05

SITE NAVIGATION

The first step in the dashboard’s development was fleshing out its navigation using React Router

06

UNDERSTANDING THE BACKEND

Python, Jira API, JQL, and AWS Lambda were used to fetch and analyze data.

07

DATA VISUALIZATION USING TREMOR UI

Static, placeholder JSON data was used to create charts using Tremor UI.

08

FETCHING DATA USING AXIOS

Axios was employed on the front end to fetch data using AWS API Gateway URL.

09

FEATURE EXPANSION

Program Increment selector and historical trend visualization were added to the dashboard.

10

REDUCING COMPLEXITY

Took advantage of React’s useState() hook to re-render components rather than having pages with React Router.

11

DEMO

Complete demo of the dashboard’s current state.

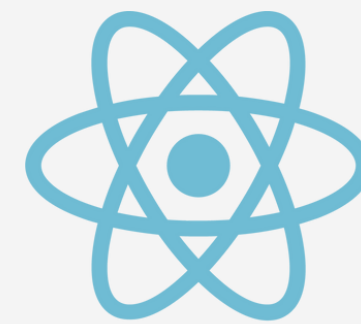
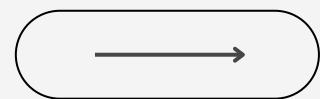


## Tools Used to Develop Front End

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

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

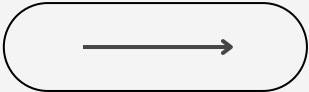
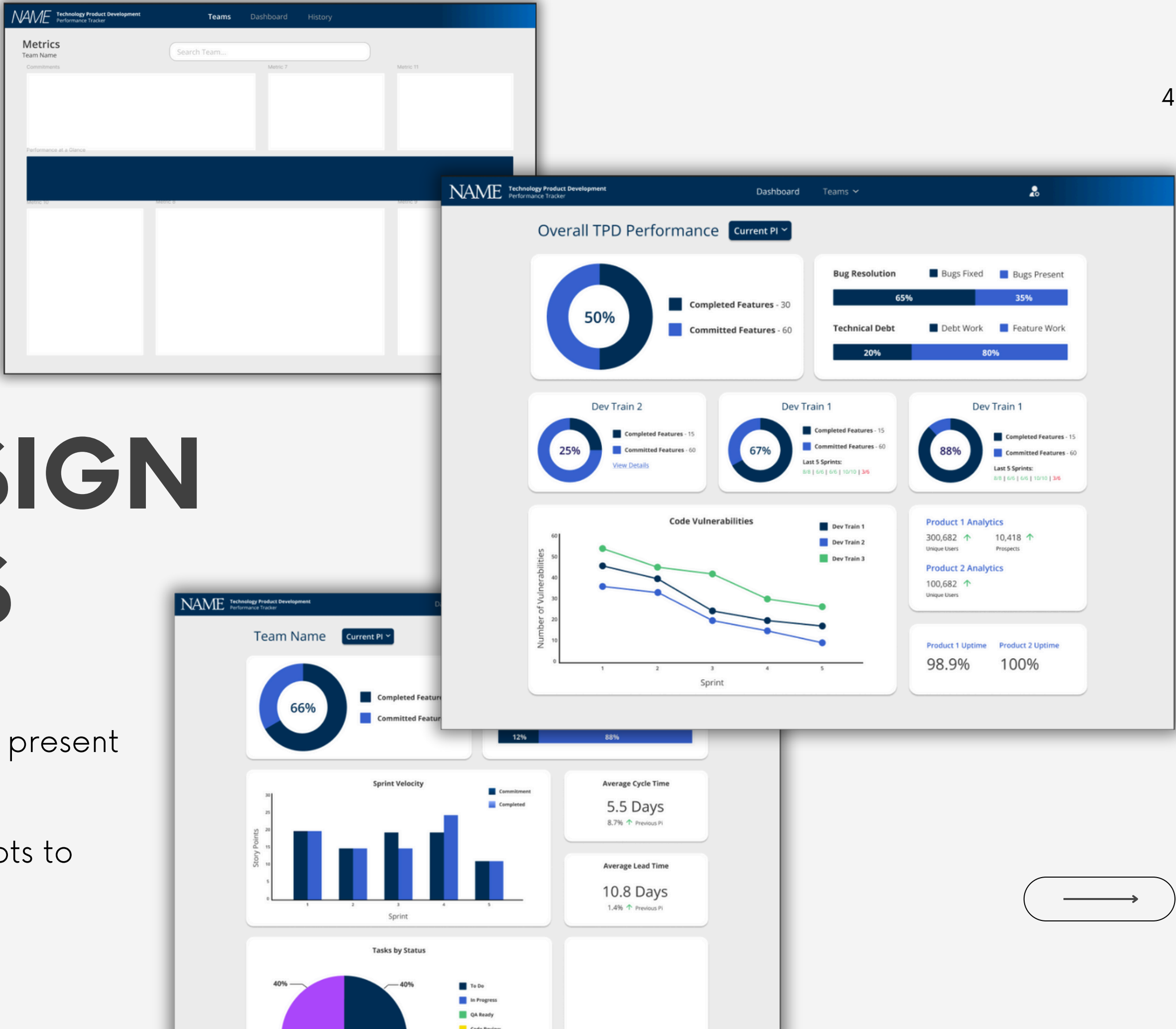


04

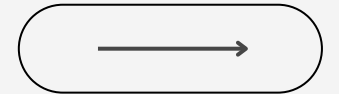
# FIGMA DESIGN CONCEPTS

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.



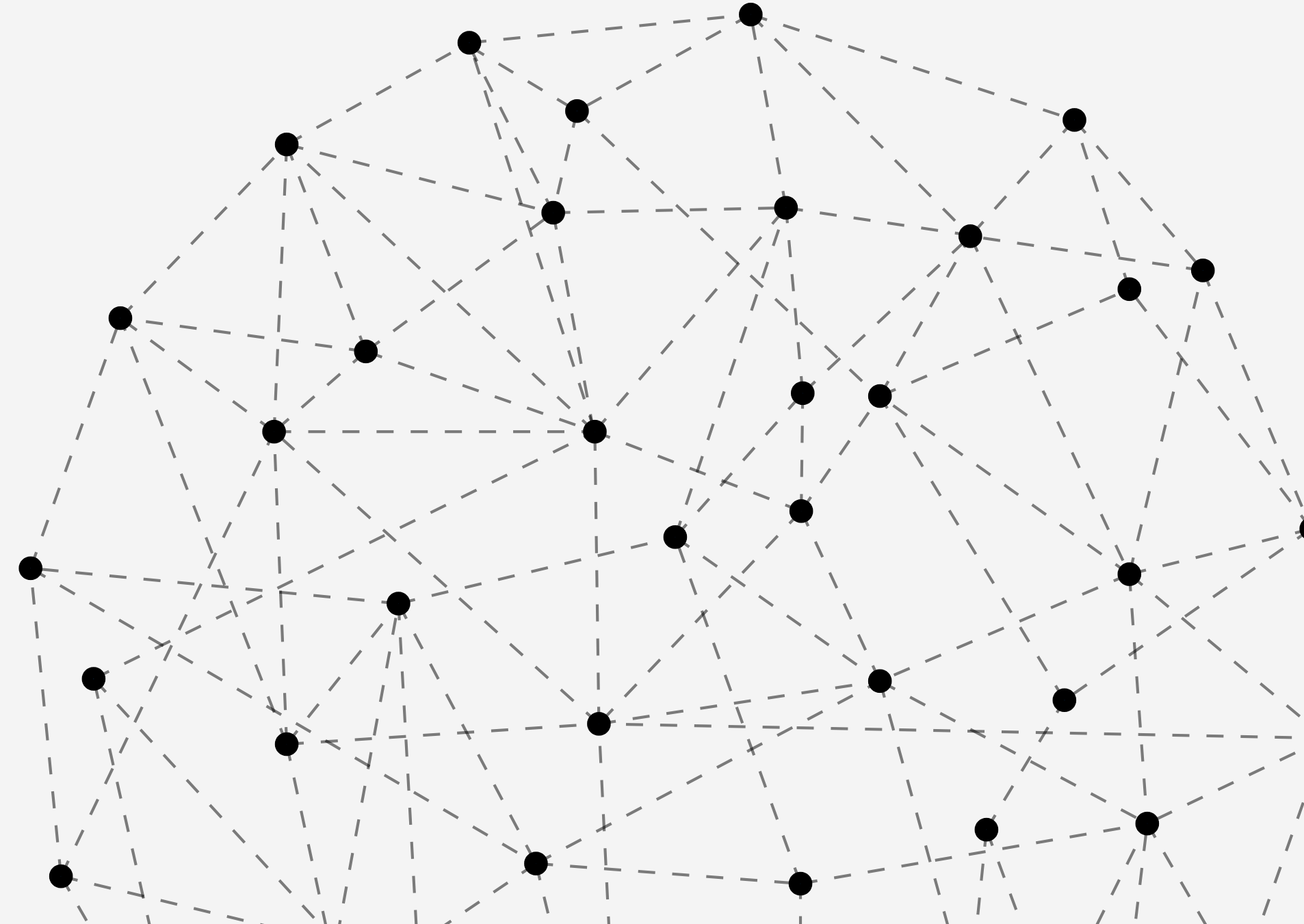
05



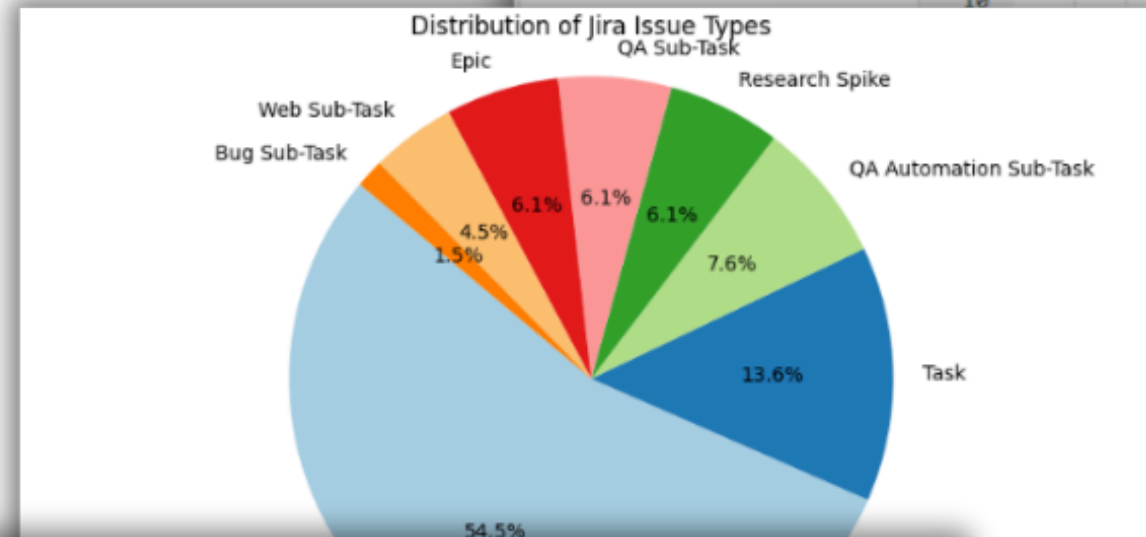
# SITE NAVIGATION

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.



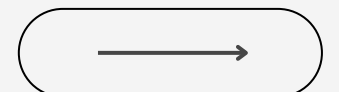
06



```
Go to Anything (Ctrl-P)  lambda_function x  Environment Vari x +
Metric1-Train - /
  lambda_function.py
  lookup.json
  OrgStatus.py
  TeamsStatus.py
  TrainsStatus.py

1 import json
2 import TrainsStatus
3 import TeamsStatus
4 import OrgStatus
5
6 def lambda_handler(event, context):
7     try:
8         path = event['resource']
9         if path == "/Metric1/Train":
10             # Extract query parameters for Train
11             train = event['queryStringParameters'].get('Train')
12             pi = event['queryStringParameters'].get('Pi')
13
14             chart_data, issue_details = TrainsStatus.get_train_data(train, pi)
```

# 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.

```
# 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)

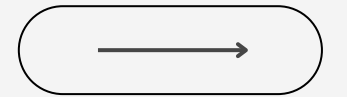
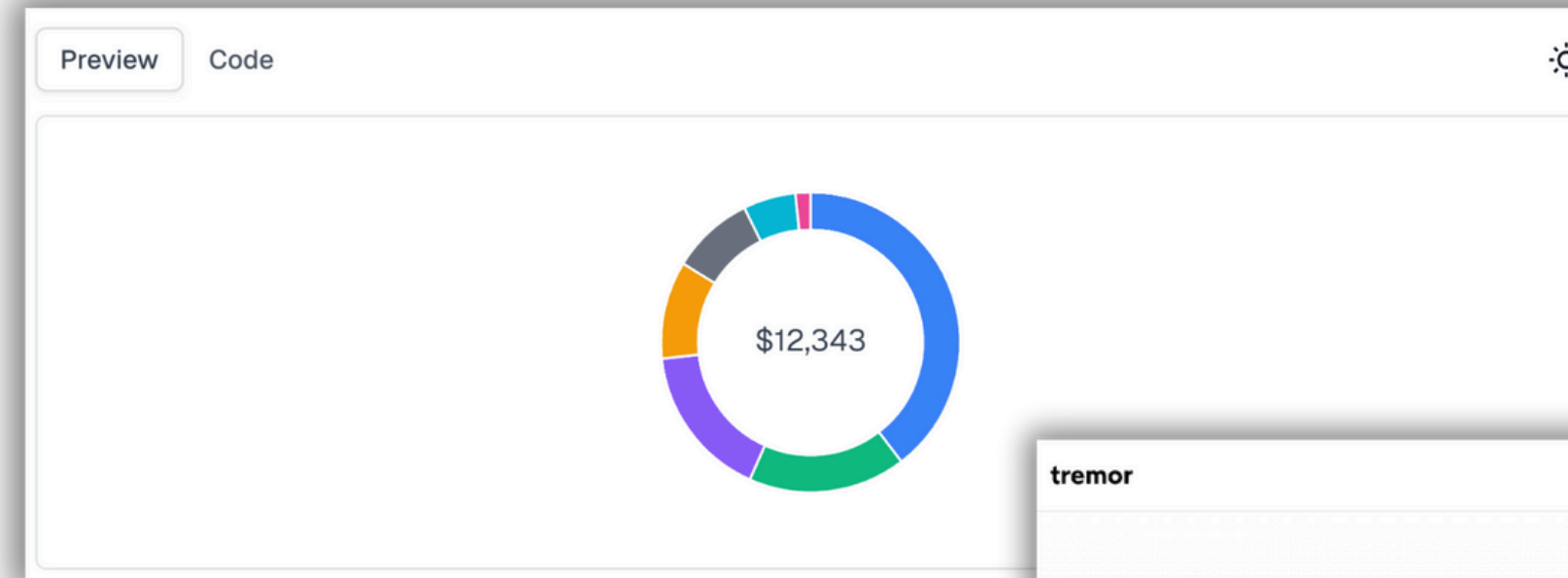
    # Count the number of issues in each status
    feature_done_count = len(issues_feature_done)
    not_feature_done_count = len(issues_not_feature_done)
```



METRIC DASHBOARD

07

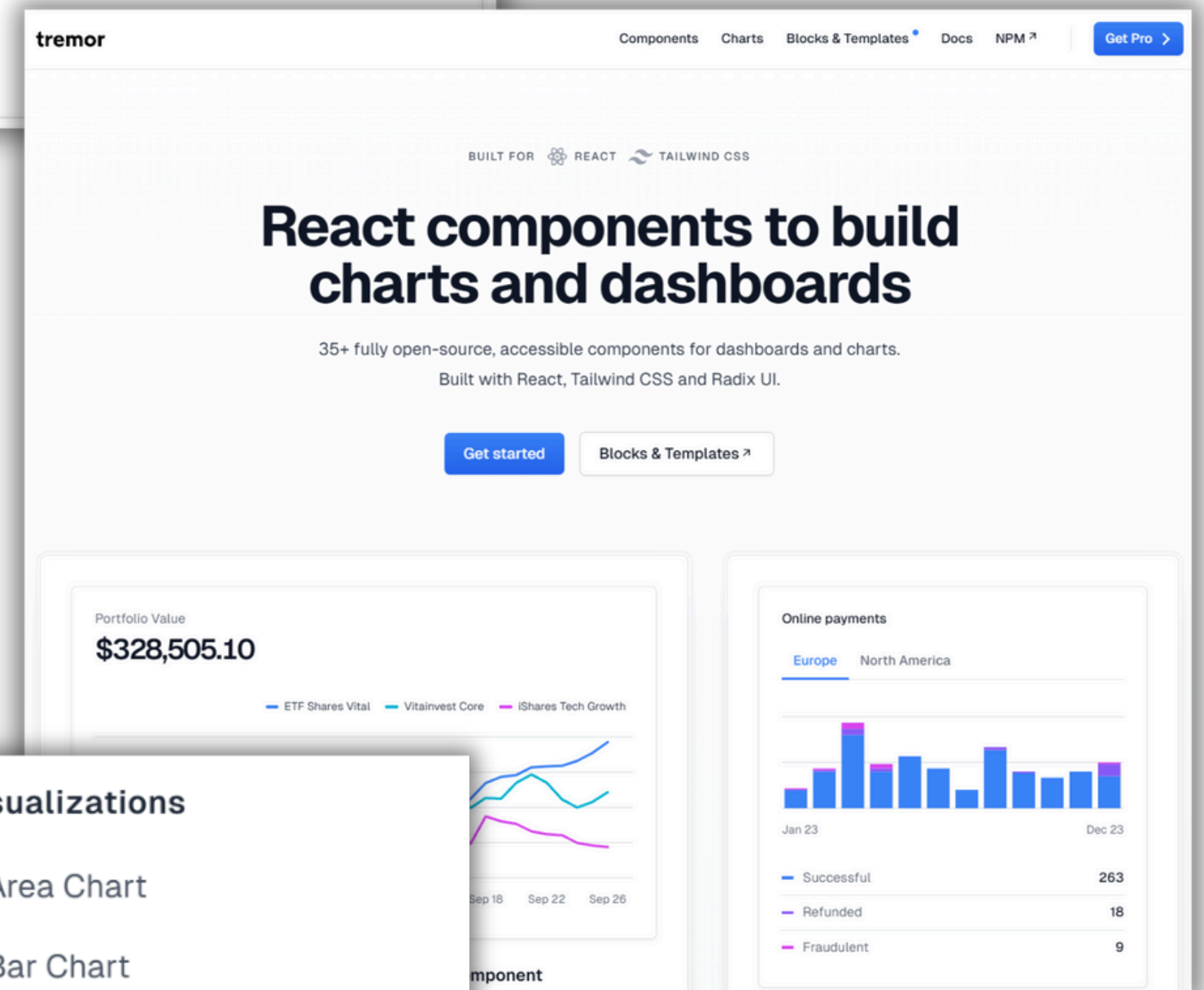
7



# 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.



### Visualizations

Area Chart

Bar Chart

Combo Chart **New**

Bar List

Category Bar

08

A X I O S

### What is Axios?

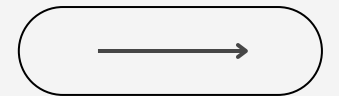
Axios is a [promise-based](#) HTTP Client for [node.js](#) and the browser. It is [isomorphic](#) (= it can run in the browser and nodejs with the same codebase). On the server-side it uses the native node.js [http](#) module, while on the client (browser) it uses XMLHttpRequests.

### 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](#)

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

# 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.

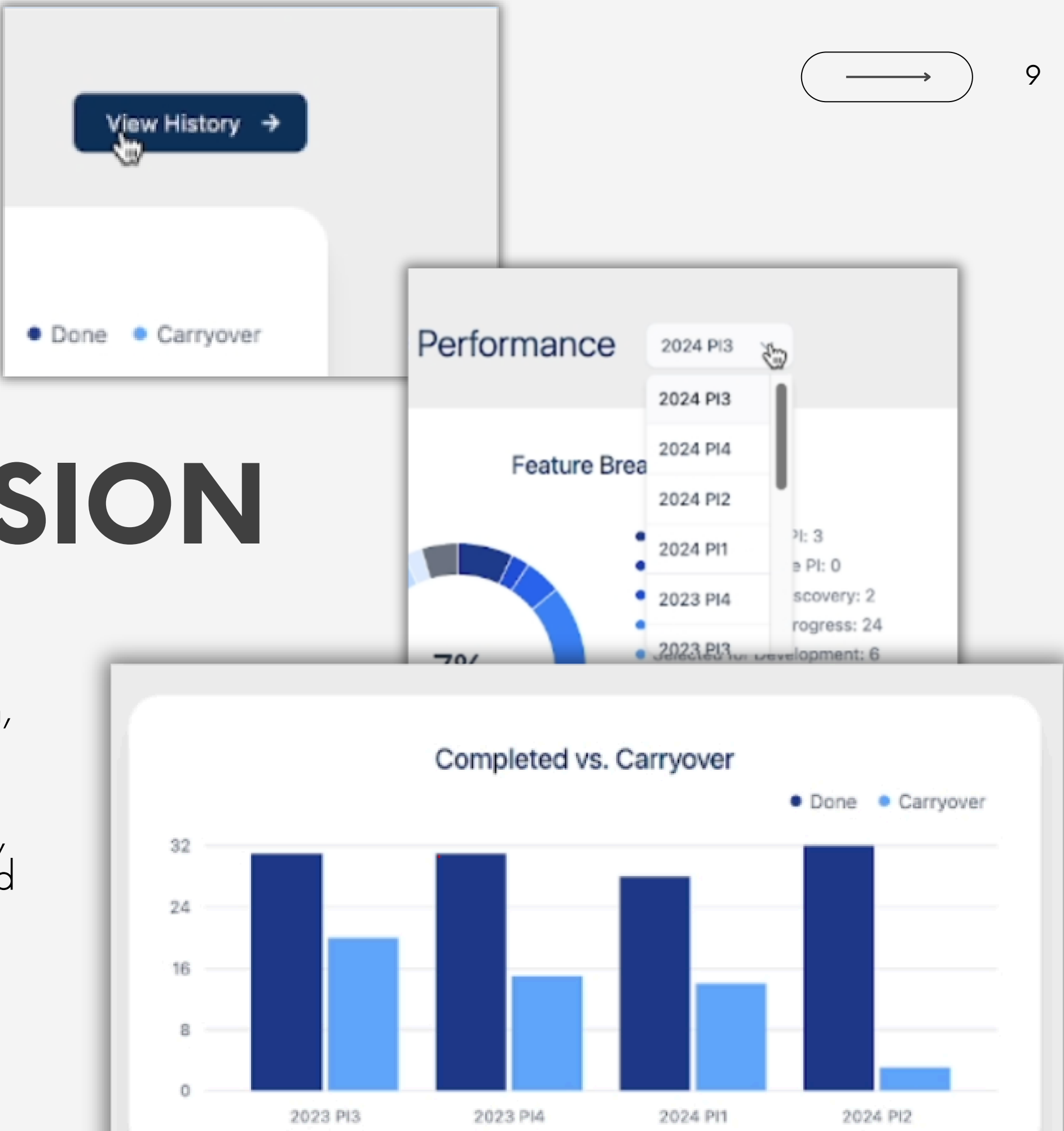


09

# 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.

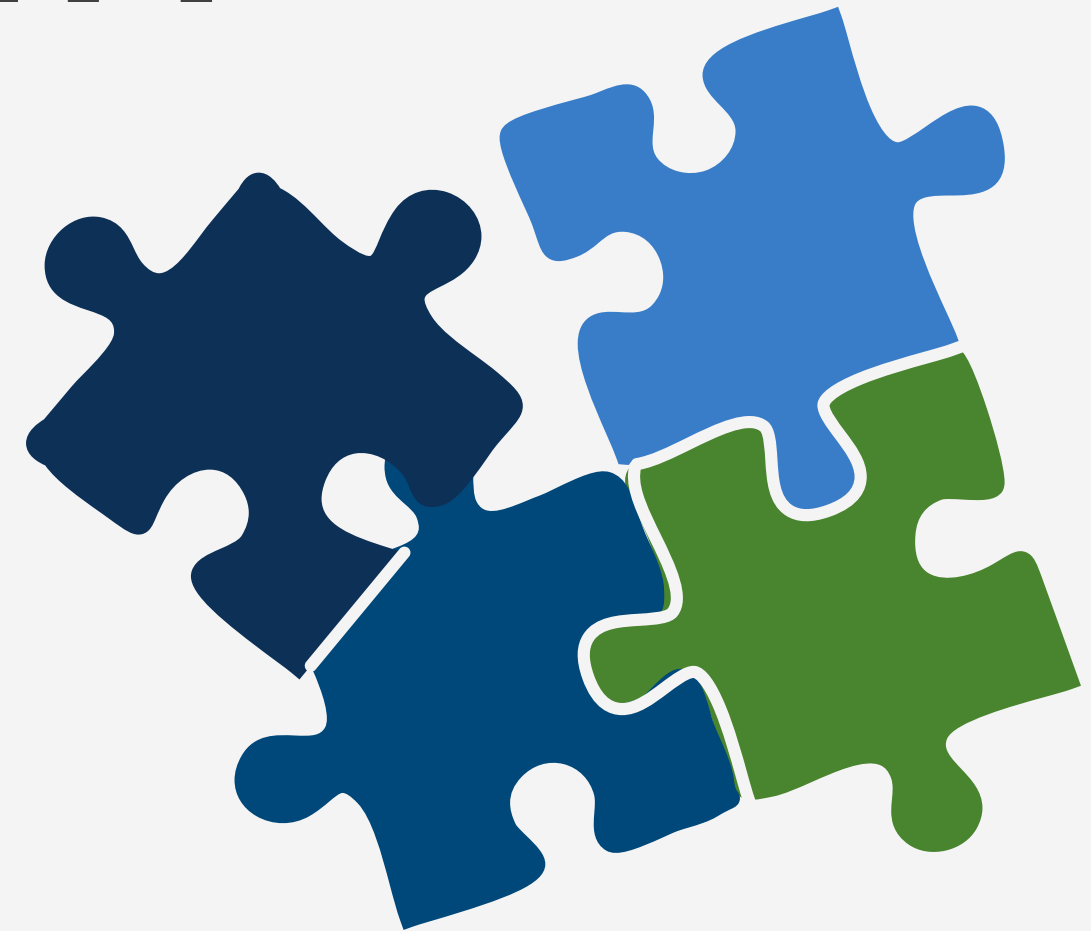


10

# REDUCING COMPLEXITY

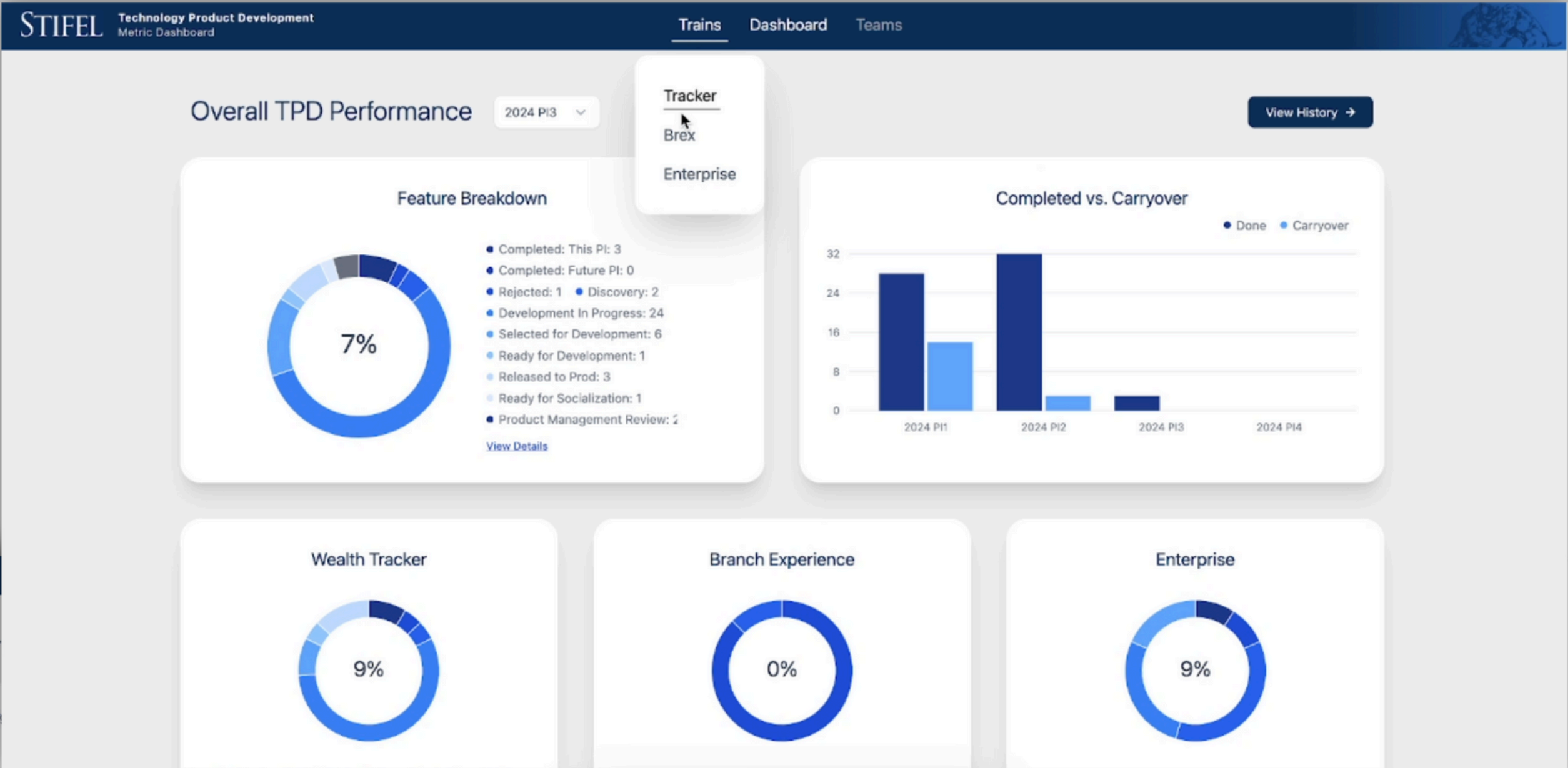
I simplified the front end by removing React Router and using useState to control which content displays on a single page instead of managing multiple routes.

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



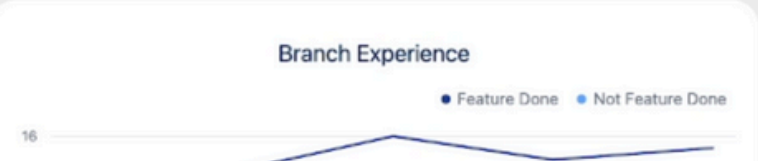
11

DEMO



STIFEL Technology Product Development Metric Dashboard

Feature Breakdown History



[HTTPS://YOUTU.BE/5ZQTUSA4D\\_I](https://youtu.be/5ZQTUSA4D_I)