POSTMANを用いたAPI開発の品質保証の向上

認証から機能、非機能テスト、そしてチーム連携

株式会社マネーフォワード
SMB開発本部 > 会計開発部 > Journal Platformグループ

菅谷 佑司 (Yuji Sugaya)

テーマとアジェンダ

- テーマ
  - POSTMANを用いたAPI開発の品質保証の向上
    - 認証から機能、非機能テスト、そしてチーム連携

- アジェンダ
  - はじめに
  - 概要
  - 問題提起
  - 各課題及び各課題の対策
  - 対策の結果 / まとめ
  - 次にトライしたいこと
  - 苦労した点 / 気をつけた方が良い点

はじめに

- POSTMANとは
  - 開発者がAPIを設計、構築、テスト、コラボレーションするためのAPIプラットフォーム
    - 認証
    - APIのテスト、テスト自動化、性能テスト
    - モックサーバー
    - ドキュメント作成、ワークスペース共有

- なぜPOSTMANを選択したのか？
  - 直感的なGUI: ボタンやフォームで操作できる使いやすいインターフェース
  - 視覚的フィードバック: リクエストの結果を分かりやすく表示
  - 豊富な学習リソース: 公式サイトや活用記事などが多く初学者でも習得しやすい
    - https://learning.postman.com/

概要

- 背景
  - プロジェクトでは**POSTMAN**の利用が**APIの動作確認**に留まっており、うまくツールを使いこなせれば**品質の向上**に貢献できると考えた

- 概要
  - プロジェクトで抱えている問題に対して、**POSTMANが提供する機能**を取り入れ、**API開発の品質向上**を目指すために実施した取り組みを紹介する

- お伝えしたい事
  - POSTMANにはAPI開発を支援する豊富な機能があり、それを活用することで**効果的にAPI開発の品質保証**を行うことができる
  - プロジェクトで利用している**ツールの理解を深め、活用**することで、その価値を**十分に発揮**することができる

問題提起

1. OAuth2.0を使用したアクセストークンの取得および更新の手順が煩雑
2. 自動のリグレッションテストが用意されておらず、毎回手動でテストが必要
3. 機能テストと性能テストで異なるツールの準備が煩雑
4. チーム内でのセキュリティテスト実施経験がなく、外部専門会社に診断を依頼する必要がある
5. API利用方法がチーム以外で共有されておらず、知識が特定のメンバーに依存している

Money Forward

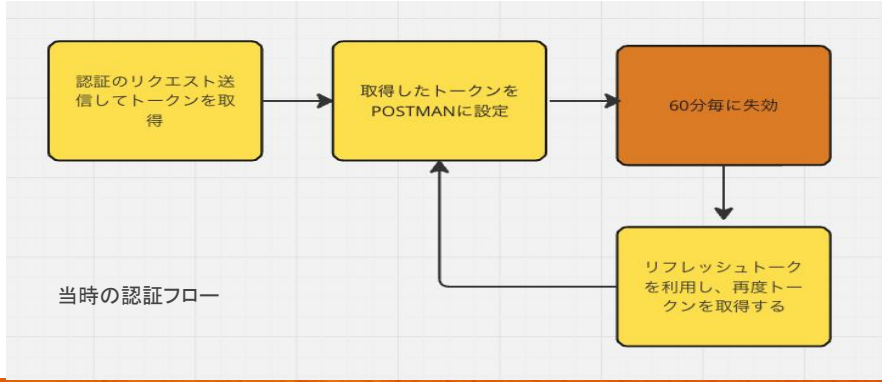
OAuth2.0を使用したアクセストークンの取得および更新の手順が煩雑

課題1

- アクセストークンの自動取得、更新が必要

- 現状
  - アクセストークン取得のプロセスが煩雑である
  - 60分ごとにトークンが失効し、毎回再取得して更新する必要がある

- 問題による影響
  - APIの利用や、テストする際の準備が煩わしい
  - API利用中にトークンが失効し、テストの継続性が損なわれる

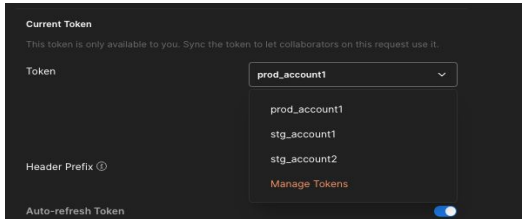


課題1の対策

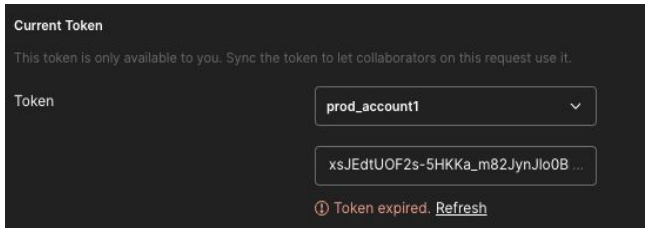
- POSTMANのOAuth2.0自動トークン・リフレッシュ機能を活用し、認証プロセスを自動化する
  - メリット
    - テスト実施中にトークンが失効せず、テストの継続性が保たれる

- 対策を実施した結果
  - 初回の認証設定した後、トークン失効時に自動で更新される
  - トークンの管理でき、複数アカウント切り替えが可能
    - dev,stg,prodなど環境ごとのアカウント
    - 設定の異なる複数のアカウント

トークンの管理



トークン失効後に自動で更新



問題2

自動のリグレッションテストが用意されておらず、毎回手動でテストが必要

課題2

- リグレッションテストの自動化が必要

- 現状
  - リグレッションテストが自動化されていない
  - 手動でのリグレッションテストに3-4時間ほどかかる

- 問題による影響
  - リグレッションテストに工数がかかりリリース速度が低下する
  - リグレッションテストを実施するたびにテスト実施のコストがかかる

課題2の対策

- POSTMANでテストスクリプトを作成し、構築したテストをCIに組み込んでテストを自動化する
  - メリット
    - テスト実施にかかる工数を大幅に短縮する。常にテストされている状態を保てる
  - デメリット
    - 認証プロセスが原因で自動化できない可能性がある
    - 1からテストスクリプトを作成するため自動化の土台を作るまでのコストがかる

- 対策を実施した結果
  - テスト実行が数分で実行可能になった

POSTMANを活用したテストの書き方の紹介

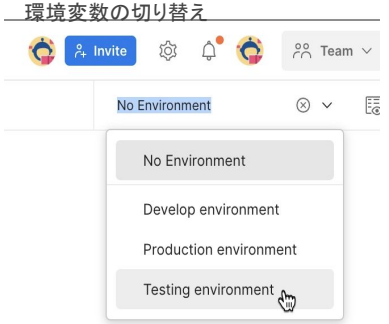
- Scriptタブにてテストを書くことができる(JavaScript)
  - 第一引数: テストケース名
  - 第二引数:assertionなど使用したテスト

```
1  pm.test("name of test case",() => {
2      pm.expect().xxx()
3  });
```

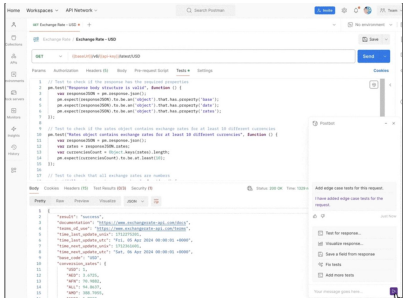
- Chai.js というassertion libraryを使用している
  - 英語の構文みたいに直感的で書きやすい
    - to.be.an(“array”) ,to.be.true , to.be.eql() , to.have()など
    - https://www.chaijs.com/api/assert/

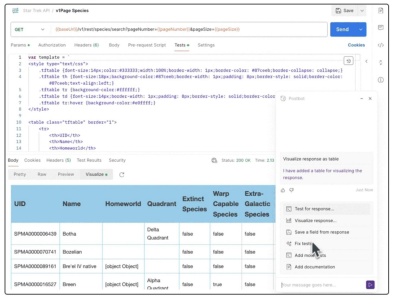
POSTMANの便利機能紹介

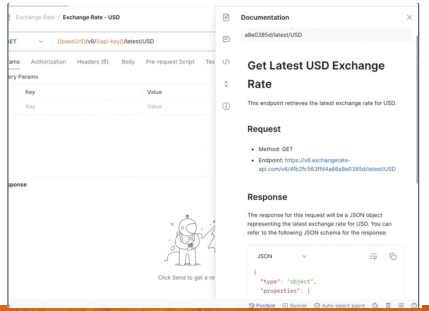
- 環境変数
  - 共通の変数を環境ごと切り替えることで、リクエストやテストスクリプトを再利用することができる
    - URL
    - コレクション内で使用している各変数



- AIアシスタント機能(Postbot)
  - テストケース、レスポンスの可視化、ドキュメントの自動生成をサポート。
  - 第一引数のテストケース名を入力した段階でテストケースの提案







NEWMANを活用したテスト実行

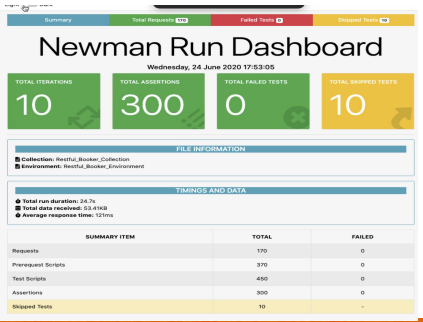
- NEWMANとは
  - POSTMANのテストを実行するコマンドラインツール
  - newman run “対象のコレクション” -e “対象の環境変数” で実行可能

- メリット
  - POSTMANアプリに依存せずターミナルからテストの実行が可能
  - テスト実施後に自動でテストレポートの出力が可能
- デメリット
  - 自動化した認証設定情報が反映されない

テストの実行結果

	executed	failed
iterations	1	0
requests	35	0
test-scripts	35	0
prerequest-scripts	0	0
assertions	86	0
total run duration: 6.8s		
total data received: 23.43KB (approx)		
average response time: 174ms [min: 92ms, max: 2.1s, s.d.: 339ms]		

テストレポート



Money Forward

機能テストと性能テストで異なるツールの準備が煩雑

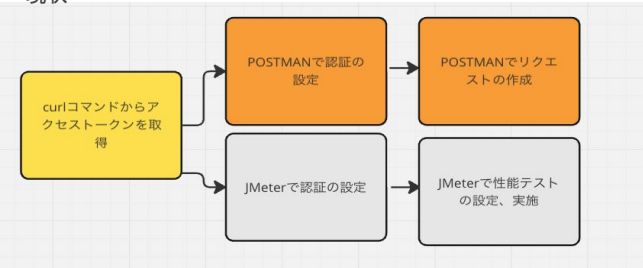
課題3

- 認証や性能テストのスクリプトを一度に準備し、再利用可能な形で管理する必要がある

- 現状
  - POSTMANで行った設定と同様の設定をJMeterでも繰り返す必要がある（例: トークン、リクエストパラメータ）

- 問題による影響
  - ツールごとに設定を再作成する手間がかかり、効率が悪化する

現状



実現したこと

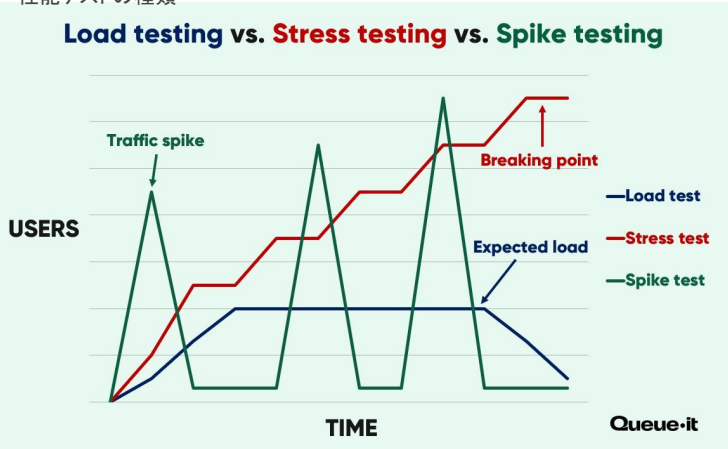
課題3の対策

- 認証設定と作成済みのリクエストを組み合わせてPOSTMAN内で性能テストを実施する
  - メリット
    - 認証設定や既存のリクエストを再利用できる
    - 機能テストと性能テストでツールの統一が可能となり、効率が向上する
  - デメリット
    - 無料プランでは仮想ユーザー数に制限があり、設計したテストの一部が実施できない可能性がある
- 対策を実施した結果
  - 認証情報やリクエストなど再利用し、性能テストの実施を可能にした

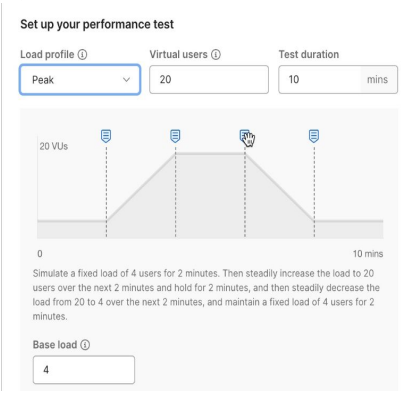
POSTMANの性能テストについて

- Options
  - 仮想ユーザー数
    - 無料プランは100名まで設定可能
- Duration
  - 1-60分
- Load profile
  - 以下の負荷の設定が可能
    - Fixed,Ramp up,Spike,Peak
- Metrics
  - スループット (req/sec)
  - エラー率 (%)
  - レイテンシー (ms)

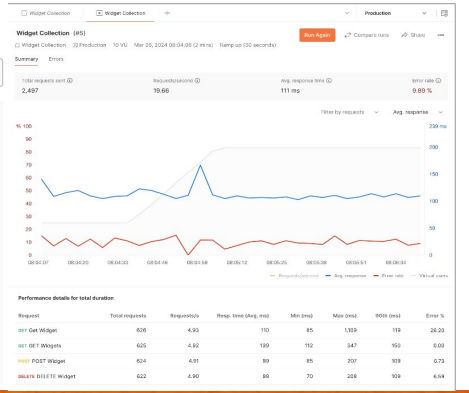
性能テストの種類



負荷の設定



テストレポート



問題4

チーム内でのセキュリティテスト実施経験がなく、外部専門会社に診断を依頼する必要がある

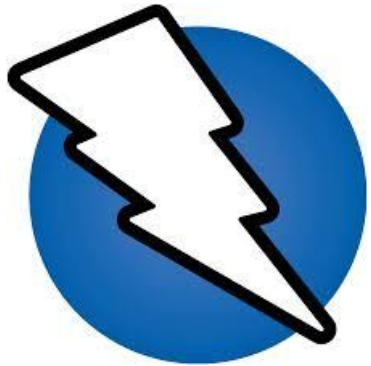
課題4

- セキュリティテストのスムーズな実施が必要
- 現状
  - 新規APIをリリースする前にセキュリティ診断を外部の専門会社に依頼している
- 問題による影響
  - 診断依頼のコミュニケーションコストが高く、スケジュール通りに実施できない可能性がある

課題4の対策

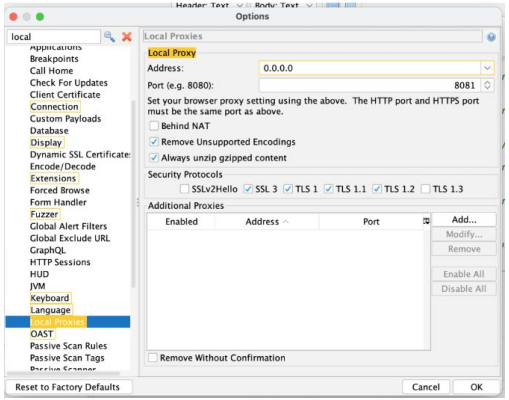
- 脆弱性診断ツールであるOWASP ZAPとPOSTMANを組み合わせて脆弱性診断を実施する
  - メリット
    - 外部環境に依存せず、社内でセキュリティテストが可能。
  - デメリット
    - 専門会社のセキュリティ診断と同等のテストが担保できるとは限らないため、社内のセキュリティ部門を巻き込み相談する必要がある
- OWASP ZAPとは？
  - オープンソースのWeb脆弱性診断ツール

- 対策を実施した結果
  - 外部環境に依存せず、セキュリティテストの実施が可能

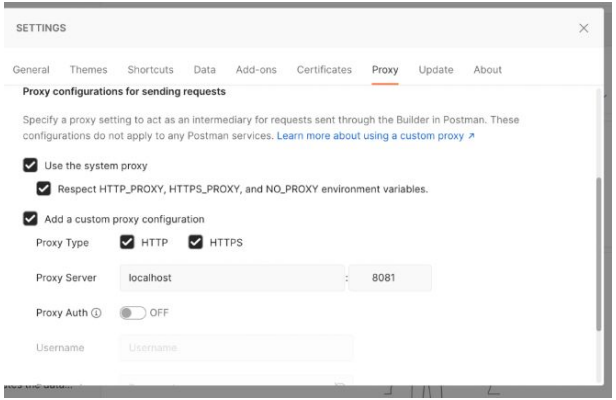


実施方法について

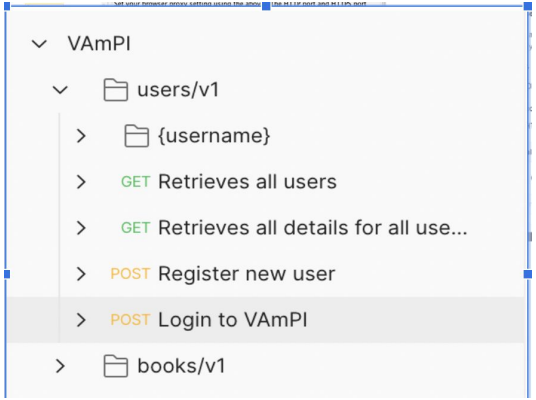
1.Proxyの設定をする(OWASP ZAP)



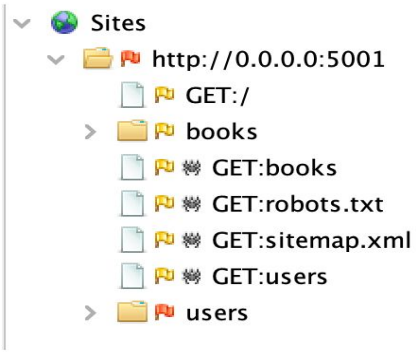
2. Proxyの設定をする(POSTMAN)



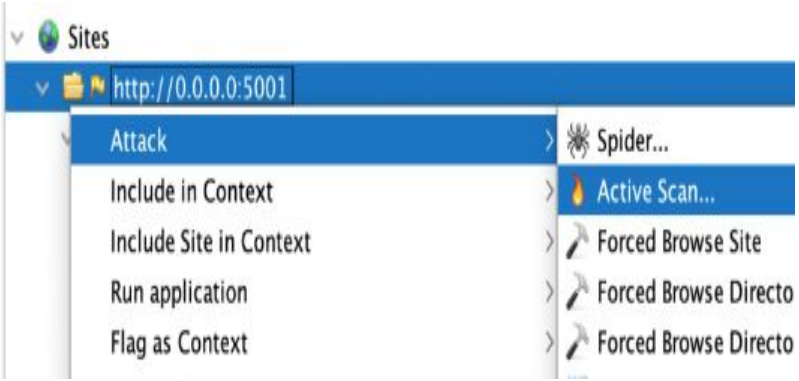
3. 診断したいエンドポイントにリクエストを送信



4.OWASP ZAPにリクエストが連携されていることを確認



5.OWASP ZAPで診断の実施をする



問題5

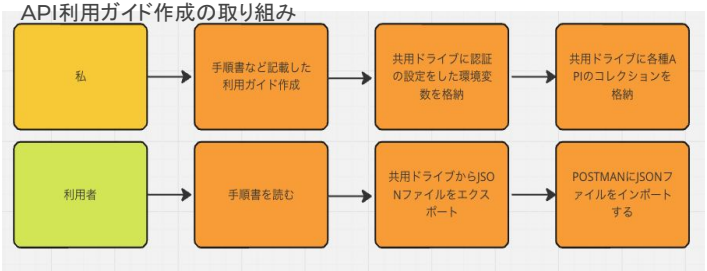
API利用方法がチーム以外で共有されておらず、
知識が特定のメンバーに依存している

課題5

- 開発チーム以外でもAPIへのアクセスと共有が容易になる環境の整備が必要
- 現状
  - APIの利用手順が社内で公開されておらず、開発メンバーのみで利用している
- 影響
  - 他チームでAPI連携が必要な場合、すぐに共有できるドキュメンがなく、その都度利用手順について説明する必要がある
  - 関連サービスで障害が発生した場合の動作確認など属人化してしまう

課題5の対策

- POSTMANのコラボレーション機能を活用し、関係者にワークスペースの共有を行う
  - メリット
    - 事前設定不要でAPIの利用が可能になり、チーム以外のメンバーもAPIを容易に触れることができる
    - 他部署とのAPI連携やPdMなどの受け入れテストなどスムーズに実することができる
  - デメリット
    - 無料プランではワークスペース共有の人数制限があり、有料プランの加入が必須になる
- 無料プランで実現したいため、API利用ガイドを作成し、共用ドライブでデータを管理する方法に変更した



対策の結果 / まとめ1

- 計画通り進んだもの
  - 認証フローの自動化により、効率よくAPIの利用が可能になったPOSTMANで自動のリグレッションテストのスクリプトを構築した
    - 3-4時間かけていたテストが5-10分に短縮できた
  - POSTMANのトークンやリクエストを再利用し、性能テストを実施し効率性が向上した
  - 外部環境に依存せずにセキュリティテストの実施が可能になった

対策の結果 / まとめ2

- 計画通り進まなかったもの
  - POSTMANで構築したテストの自動化
    - 認証の自動化がNEWMANでサポートされていないため、CIに組み込むことができなかった
      - CIに組み込まず、手動で自動テストを実行する
      - 代替策についてはチームで検討中
  - 有料プラン加入のハードルが高く、チームコラボ機能が使えない。
    - POSTMANのコレクションエクスポートし、データを共用ドライブで管理し、各自インポートする運用に変更した
    - 設定や使い方をまとめたAPI利用ガイドの資料を作成し、部内で展開した

次にトライしたいこと

- TestRailとPOSTMANを連携したテストケースや実行結果の管理
  - TestRailとは
    - テストケースやテスト結果などテストに関係するデータ管理ツール

- 背景
  - 各プロジェクトでテストケースが分散しており、一元管理が難しい。
  - TestRailを活用することで、テストケースと実行結果を一元管理したいと考えている

- 目標
  - TestRailを使用して、POSTMANのテストケースとその実行結果を効率的に管理する。
  - テスト結果の可視化と共有を容易にし、プロジェクト全体の品質向上を目指す

苦労した点 / 気をつけた方が良い点

- 苦労した点
  - 社内でPOSTMANの活用事例などの情報が少なく、活用できそうな機能の調査や自動テストの書き方を学ぶのに時間や学習コストがかかった

- 気をつけた方が良い点
  - OWASP ZAPを用いた脆弱性診断について、専門のセキュリティ会社と同等の診断ができるとは限らないため、セキュリティテストを外部に依頼するかどうかの根拠やデータを集めた上で判断する必要がある
    - (本プロジェクトでは新規API公開時には専門のセキュリティ会社の診断を受け、その後のリリースについては基盤等の大きな変更がない限りOWASP ZAP での診断を行なっている。)
  - POSTMANの導入について、今回は既にプロジェクトで導入されていたため活用方法を計画したが、未導入の場合はプランごとの機能制限や無料プランでの代替策を事前に調査し、導入の可否を検討する必要がある

Appendix

参考文献

Web

認証関連
・https://learning.postman.com/docs/sending-requests/authorization/oauth-20/
・https://qiita.com/yokawasa/items/b4d89d983bfd23792788

機能テスト関連
・https://www.postman.com/product/postbot/
・https://learning.postman.com/docs/sending-requests/variables/managing-environments/
・https://learning.postman.com/docs/collections/using-newman-cli/command-line-integration-with-newman/
・https://qiita.com/nagix/items/4ae601117b39b05458ec

セキュリティテスト関連
・https://thetesttherapist.com/2022/02/13/api-security-testing-with-postman-and-owasp-zap/comment-page-1/

性能テスト関連
・https://learning.postman.com/docs/collections/performance-testing/performance-test-configuration/
・https://qiita.com/yokawasa/items/c172f1765026bf90d3ac

チームコラボレーション関連
・https://learning.postman.com/docs/collaborating-in-postman/working-with-your-team/collaboration-overview/
・https://speakerdeck.com/nagix/postmandetimukoraboresiyon

ありがとうございました

株式会社マネーフォワード
菅谷 佑司
sugaya.yuji@moneyforward.co.jp