Backlog

Wirtschaftsprojekt «px: PEAX Command Line Client»

Patrick Bucher

20.10.2019

#	User Story	Status	Story Points
1	Konfiguration sämtlicher Umgebungen	umgesetzt	1
2	Erweiterung der CI-Pipeline	umgesetzt	5
3	Login mit Zwei-Faktor-Authentifizierung	umgesetzt	3
4	Sichere Verwahrung der Tokens	eingeplant für Sprint 1	5
5	Handhabung mehrerer Umgebungen	eingeplant für Sprint 1	3
6	Generische `GET`-Schnittstelle	eingeplant für Sprint 1	3
	Verbesserung der Hilfe-Funktion	vorgesehen für Sprint 2	
	Automatische Aktualisierung von Tokens	vorgesehen für Sprint 2	
	Login für Agent API	vorgesehen für Sprint 2	
	Einliefern von Dokumenten per Delivey API	offen	
	Einliefern von Dokumenten per User API	offen	
	Einlieferung von Verzeichnissen mit Dokumenten	offen	
	Auflisten von Dokumenten mit Suche/Filterung	offen	
	Generische `POST`-Schnittstelle	offen	
	Generische `PUT`-Schnittstelle	offen	
	Generische `PATCH`-Schnittstelle	offen	
	Ausführung von Befehlen für mehrere Umgebungen	offen	
	Fortschrittsanzeige bei längeren Vorgängen	offen	
	Ausgabe von Tokens	offen	
	Inspektion von Tokens	offen	

Sprints

Sprint 1

Eingeplant: sechs Stories, 20 Story Points

Bisher abgearbeitet: 8 Story Points (in 10 Stunden)

User Stories

Die User Stories haben drei Grössen: S (small: 1 Story Point), M (medium: 3 Story Points) und L (large: 5 Story Points). Diese Grössen dienen zur relativen Einschätzung der Story-Grössen zueinander, und sollen nicht in eine Stundenplanung heruntergerechnet werden. Vielmehr sollen sie dazu dienen, übergrosse Stories als solche zu erkennen, um diese herunterbrechen zu können. Am Ende eines jeden Sprints soll eingeschätzt werden, wie viele Story Points ungefähr machbar sind.

1: Konfiguration sämtlicher Umgebungen

Als Entwickler möchte ich sämtliche relevanten PEAX-Umgebungen vorkonfiguriert haben, damit diese dem Nutzer zur Verfügung gestellt werden können.

Akzeptanzkriterien:

1. Es sollen die Umgebungen `dev`, `test`, `devpatch`, `testpatch`, `stage`, `prod`, `perf` und `prototype` zur Verfügung stehen.
2. Für jede Umgebung muss eine erreichbare URL zum jeweiligen Identity Provider und zu den relevanten APIs (User API, Admin API, Agent API) automatisch generiert werden können.

Testprotokoll

1. automatisiert: `env_test.go`
2. manuell: Login auf verschiedenen Umgebungen (nicht alle, perf/prototype sind heruntergefahren, prod benötigt 2FA) erfolgreich
    1. login vor Registrierung: 401
    2. login nach Registrierung/vor Verifizierung: 403
    3. login nach Verifizierung: 200

Notizen

- table-driven test design (gopl p. 306)
- Ausnahme: `prod`-Umgebung (via PEAX-Subdomains, issuer bei OAuth-Tokens relevant, siehe Mobile App)
- Aufwand: ca. 1 Stunde

2: Erweiterung der CI-Pipeline

Als Entwickler möchte ich `px` in Skripts einbauen können, welche anschliessend in der CI-Pipeline berücksichtigt, d.h. ausgeführt werden.

Akzeptanzkriterien:

1. Der Ausführungsschritt `script` soll nur dann ausgeführt werden, wenn die vorherigen Schritte `build` und `test` erfolgreich waren.
2. Um der Pipeline ein weiteres Skript hinzufügen zu können, soll die neu erstellte Skriptdatei nur an einer einzigen Stelle (Konfigurationsdatei) hinzugefügt werden müssen.

Testprotokoll

- zunächst Skript erstellt, das die anderen Skripts ausführt
- ein scheiterndes und ein durchlaufendes Skript erstellt und in ersterem referenziert
- Pipeline für GitLab konfiguriert, Test erfolgreich gescheitert, d.h. scheiterndes Skript hat die Pipeline wie gewünscht zum Abbruch gebracht
- scheiterndes Skript aus der Liste entfernt, Pipelien lief nun durch
- Artefakt kompilieren und dem Skript zur Verfügung stellen
- Minimaltest mit `px help` durchführen
- Login-Testfall (mit Variablen via GitLab) erstellt
- gescheiterte Versuche mit `px login` gaben Status 0 zurück, musste korrigiert werden
- Login-Skript hat schliesslich wie gewünscht funktioniert

Notizen

- viel Zeit aufgrund mangelhafter Bash-Kenntnisse verloren (iterieren über Liste von Skriptdateien)
- TODO: erneut prüfen (Begründung wegen lokaler Ausführbarkeit sinnvoll)! Akzeptanzkriterium 1 über den Haufen geworfen: das Artefakt aus dem `build`-Schritt kann nicht im `script`-Schritt verwendet werden, darum eigene Kompilierung im `script`-Schritt; dafür höhere Performance
- Aufwand schlussendlich überschätzt
- Erweiterung: soll auch lokal ausgeführt werden müssen
- dazu Verschiebung der Build-Logik von gitlab-ci zu Skript
- optionale envvars-Datei (unter .gitignore) für lokale Ausführung
- Bash-Code vereinfacht

3: Login mit Zwei-Faktor-Authentifizierung

Als Benutzer möchte ich mich per Zwei-Faktor-Authentifizierung einloggen können, damit ich `px` auch mit entsprechend konfigurierten Zugängen verwenden kann.

Akzeptanzkriterien:

1. Die Abfrage des zweiten Faktors soll interaktiv passieren.
2. Es sollen die Authentifizierungsarten SMS und OTP (One-Time Password) unterstützt werden.
3. Das Login soll bei entsprechend konfigurierten Benutzerkonti auch weiterhin ohne Zwei-Faktor-Authentifizierung funktionieren.

Testprotokoll

- beim Refactoring traten immer wieder Build-Fehler auf, die jedoch einfach zu beheben waren
- Logik-Fehler traten keine auf
- HTTP Status 400 nach erstem Versuch, lange am Fehler analysirt
- gemerkt, dass hier kein JSON unterstützt wird, sondern nur form-urlencoded (unnötiges Debugging auf Server)
- schliesslich Erfolgreich
- Login mit SMS und TOTP erfolgreich manuell getestet (automatischer Test nicht praktikabel)
- Login ohne 2FA wird weiterhin via Pipeline getestet

Notizen

- bevor die bestehende `login`-Funktion erweitert werden konnte, musste sie zunächst etwas aufgeräumt werden
- hierzu wurden verschiedene Teilaspekte (interaktive Abfrage fehlender Credentials, Erstellen des Requests und Parsen der Response) in eigene Funktionen ausgelagert
- die `script`-Pipeline zahlte sich bereits aus, zumal die `login()`-Funktion direkt in `cmd/px.go` implementiert war und darum nicht durch einen Unit Test abgedeckt war
- das Refactoring wurde schliesslich ausgedehnt; es enstanden neue Submodule `px/tokenstore` und `px/utils`
- es wurden neue Datenstrukturen erstellt, etwa für die Credentials (mit und ohne 2FA), und für den Login-Payload mit 2FA
- Code neu organisieren war nötig und sinnvoll

4: Sichere Verwahrung der Tokens

Als Benutzer möchte ich, dass beim Login geholte Tokens sicher lokal verwahrt werden, damit ein Angreifer diese nicht auslesen kann.

Akzeptanzkriterien:

1. Die Credentials (Benutzername und Password) werden zu keinem Zeitpunkt lokal persistent abgespeichert.
2. Refresh Tokens, die von der Produktivumgebung (`prod`) geholt werden, dürfen standardmässig nicht im Klartext abgespeichert werden.
3. Access und Refresh Tokens von nicht-produktiven Umgebungen, sowie Access Tokens der Produktivumgebung, können lokal im Klartext abgespeichert werden, sofern dies der einfacheren Bedienbarkeit zuträglich ist (weniger Passwortabfragen durch den Keystore).
4. Der Benutzer soll das Standardverhalten für die jeweiligen Umgebungen (produktiv: nur sichere Verwahrung; nicht-produktiv: Verwahrung im Klartext) mit den Kommandozeilenparametern `-safe` bzw. `-unsafe` übersteuern können.
5. Die sichere Verwahrung der Tokens muss Windows, macOS und Linux funktionieren.
6. Auf Systemen ohne GUI soll zumindest die unsichere Variante der Token-Verwahrung funktionieren.

Testprotokoll

- der Unittest `env_test` wurde um das `Confidential`-Flag erweitert, wobei nur `prod` so konfiguriert ist
- Tests auf `prod` mit Linux (Tokens schreiben) funktionierte nachdem Key Store korrekt konfiguriert wurde
- der Skript-Test `ci-px-login.sh` wurde erweitert und in `ci-px-login-logout.sh` umbenannt, sodass nun auch das Logout getestet wird
- nach dem Login wird mit `jq` geprüft, ob ein Feld `access_token` für die Umgebung `test` in `~/.px-tokens` vorhanden ist
- nach dem Logout wird das Fehlen desselben geprüft
- auf Windows sind die Tokens in der Anwendung *Credential Manager* unter *Windows Credentials* zu finden
- auf macOS sind die Tokens in der Anwendung *Keychain Access* unter *login* zu finden

Notizen

- die Umgebungskonfiguration wird um ein Flag (confidential) erweitert, dass besagt, ob für die jeweilige Umgebung die Tokens per default sicher oder unsicher verwahrt werden sollen
- die Library `zalando/go-keyring` kann Schlüssel auf Linux, macOS und Windows sicher verwahren

- die Konfiguration des nativen Key Stores ist für jede Plattform anders und wird im README des Projekts dokumentiert

5: Handhabung mehrerer Umgebungen

Als Benutzer möchte ich, dass `px` meine Befehle standardmässig gegen die Umgebung ausführt, auf der ich mich zuletzt eingeloggt habe, damit ich nicht immer eine Umgebung per Kommandozeilenparameter anwählen muss.

Akzeptanzkriterien:

1. Es muss einen Unterbefehl geben, der mir die aktuelle Umgebung (d.h. die Umgebung, auf der sich der Benutzer zuletzt eingeloggt hat) anzeigt.
2. Es muss einen Unterbefehl geben, womit eine Umgebung mit bereits aktivem Logging als die Standardumgebung gesetzt werden kann.
3. Bei allen Befehlen, die gegen die API operieren, soll die Umgebung mit einem Kommandozeilenparameter `-e` bzw. `-env` spezifiziert werden können.

6: Generische `GET`-Schnittstelle

Als Benutzer möchte ich einen `get`-Befehl zur Verfügung haben, damit ich lesend auf meine Ressourcen zugreifen kann.

Akzeptanzkriterien:

1. Dem Befehl kann ein beliebiger Ressourcenpfad mitsamt Query-Parametern mitgegeben werden.
2. Die Base-URL der jeweiligen API und Umgebung wird dem Ressourcenpfad automatisch vorangestellt.
3. Im Falle eines erfolgreichen Zugriffs (`200 OK`) soll der resultierende Payload auf die Standardausgabe (`stdout`) ausgegeben werden.
4. Im Falle eines fehlerhaften Zugriffs soll der Status-Code auf die Standardfehlerausgage (`stdout`) ausgegeben werden.