最高です！リポジトリの“いま”の構成（Kong=DB-LESS、フロントは `/app`、BFF は `/api/*`、Keycloak/Redis あり）に合わせて、README.md を丸っと書き直しました。 **DB-LESSを標準**にしつつ、**将来 DB モードへ切替えるときの注意点**と **deck/手動(curl) の手順**も残しています。

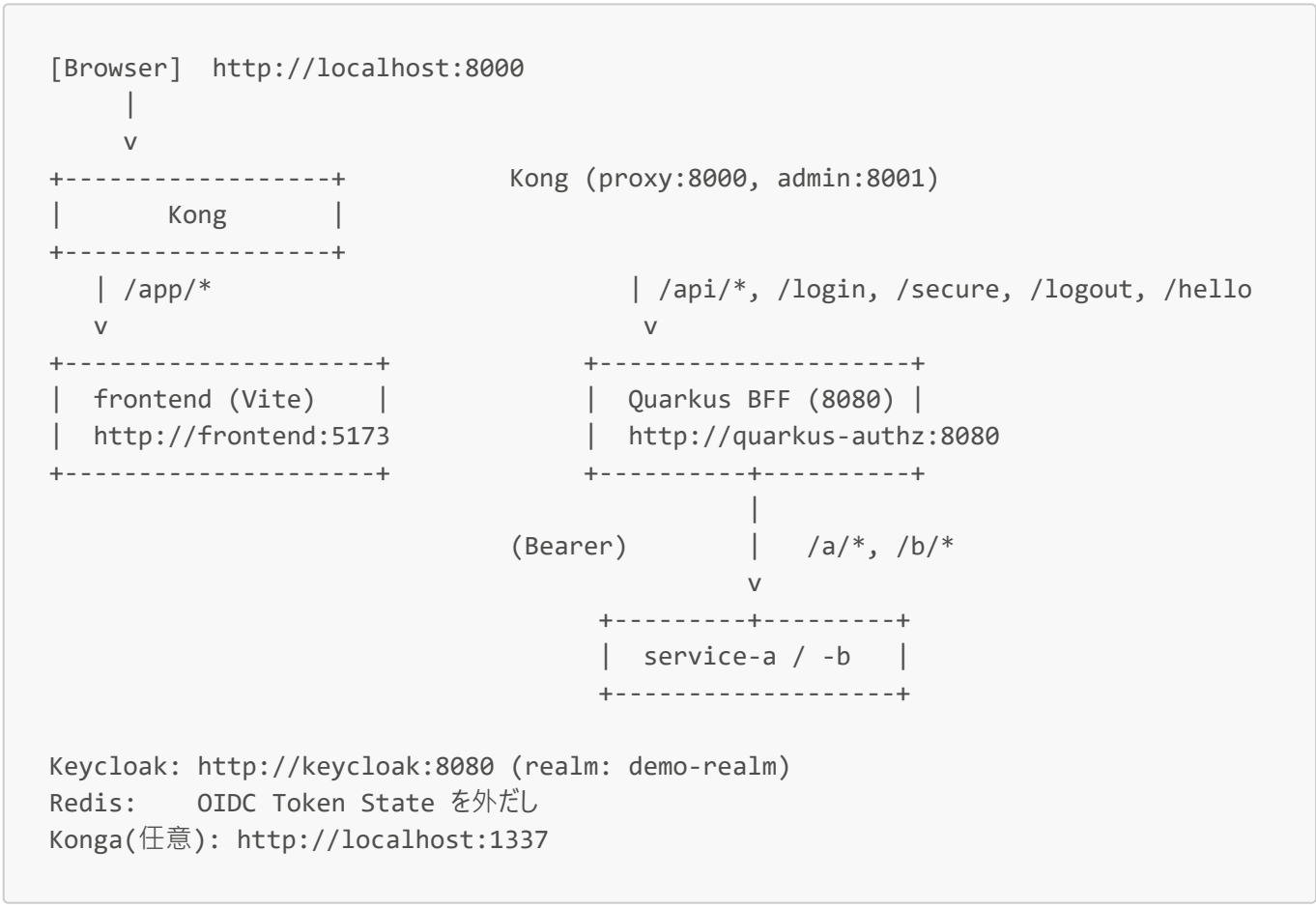
多段認証・多段認可デモ

Quarkus (BFF/Aggregator) × Kong × Keycloak × Redis × React(Vite)

ユーザー → **Kong** → **Quarkus(BFF)** で OIDC ログイン（セッションは **Redis**） → BFF は **ユーザーのアクセストークンを下流へ転送** → **Service A/B** がそれぞれ認可判定 → 結果をマージして返す構成です。

- フロント：`/app/*`（Vite Dev Server）
- BFF/API：`/api/*`（Quarkus）
- OIDC コールバック：`/login`（BFFで受けて `/app/` に戻す）
- セッション：Redis（Token State Manager）

アーキテクチャ



主要エンドポイント

- フロント：`http://localhost:8000/app/`

- API(BFF)：`http://localhost:8000/api/*`
  - ログイン：`/api/login` →（Kong/BFF）→ `/secure` →（Keycloak）→ `/login` → `/app/`
  - ログアウト：`/api/logout`
  - 認証ユーザ：`/api/me`
  - マッシュアップ：`/api/mashup`
- Keycloak：`http://localhost:8080/`（realm: `demo-realm`）
- Konga（任意）：`http://localhost:1337/`

前提

- JDK 17、Maven、Docker / Docker Compose、Node.js (18+)
- `jq`（Kong の確認で便利）
- deck（DB モードで使うと便利）：https://github.com/kong/deck/releases

セットアップ & 起動（DB-LESS：標準）

> リポジトリは **DB-LESS モード**で起動しやすい `docker-compose.yaml` と **宣言的設定** `kong/kong.yml` を同梱しています。

```
git clone https://github.com/h-i500/multi-authentication.git
cd multi-authentication

# アプリをビルド
cd quarkus-authz && mvn clean package && cd ..
cd service-a     && mvn clean package && cd ..
cd service-b     && mvn clean package && cd ..
cd frontend      && npm ci            && cd ..

# 全体を起動
docker compose up -d --build
```

> ☑ 起動後：`http://localhost:8000/app/` にアクセス → 「ログイン」ボタン → Keycloak ログイン → `/app/` に戻る → 「/api/me」「/api/mashup」ボタンで BFF 経由の API を確認できます。 **権限不足**の場合、フロントは **「権限エラー：この操作を行う権限がありません。」** を表示します。

Kong（DB-LESS）宣言的設定

Kong は DB-LESS で、`kong/kong.yml` が読み込まれます（compose で `KONG_DATABASE=off` / `KONG_DECLARATIVE_CONFIG` を指定）。

`kong/kong.yml`（抜粋・実態はリポジトリ内を使用）

```yaml
_format_version: "3.0"
_transform: true

services:
  # === Quarkus BFF (API) ===
  - name: api-svc
    host: quarkus-authz
    port: 8080
    protocol: http
    routes:
      - name: api-route
        paths: ["/api"]
        strip_path: true
        preserve_host: true
        path_handling: v0

      - name: login-route
        paths: ["/login"]
        strip_path: false
        preserve_host: true
        path_handling: v0

      - name: logout-route
        paths: ["/logout"]
        strip_path: false
        preserve_host: true
        path_handling: v0

      - name: hello-route
        paths: ["/hello"]
        strip_path: false
        preserve_host: true
        path_handling: v0

      - name: secure-route
        paths: ["/secure"]
        strip_path: false
        preserve_host: true
        path_handling: v0

  # === Vite Dev Server (Frontend) ===
  - name: frontend-dev
    host: frontend
    port: 5173
    protocol: http
    routes:
      - name: frontend-dev-route
        # 301/ループ回避のため "/app" と "/app/" の両方
        paths: ["/app", "/app/"]
        strip_path: false
        preserve_host: true
        path_handling: v1
```

> ⚠ **ありがちなミス**：`KONG_DECLARATIVE_CONFIG` に **ディレクトリ**をマウントすると「`…/kong.yml: Is a directory`」で起動失敗します。**ファイル**をマウントしてください。

（参考）Kong を **手動(curl)** で作る場合（DB/DB-LESS 共通）

Admin API：`http://localhost:8001`

1) Services

```
# === Quarkus BFF (API) ===
curl -sS -X PUT http://localhost:8001/services/api-svc \
  -d url=http://quarkus-authz:8080

# === Vite Dev (Frontend) ===
curl -sS -X PUT http://localhost:8001/services/frontend-dev \
  -d url=http://frontend:5173
```

2) Routes（BFF 側）

```
# /api → BFF。/api は剥がす
curl -sS -X PUT http://localhost:8001/routes/api-route \
  -d service.name=api-svc -d paths[]=/api -d strip_path=true \
  -d preserve_host=true -d path_handling=v0

# OIDC コールバックはそのまま渡す
for p in login logout hello secure; do
  curl -sS -X PUT http://localhost:8001/routes/$p-route \
    -d service.name=api-svc -d paths[]=/$p \
    -d strip_path=false -d preserve_host=true -d path_handling=v0
done
```

3) Routes（フロント側）

```
# /app と /app/ の両方を受ける。strip_path=false & v1
curl -sS -X PUT http://localhost:8001/routes/frontend-dev-route \
  -d service.name=frontend-dev \
  -d paths[]=/app -d paths[]=/app/ \
  -d strip_path=false -d preserve_host=true -d path_handling=v1
```

4) 確認

```
# ルート一覧
curl -s http://localhost:8001/routes \
  | jq '.data[] | {name, paths, strip_path, preserve_host, path_handling, service:
```

```
  .service.id}'

# サービス一覧
curl -s http://localhost:8001/services \
  | jq '.data[] | {name, host, port, path}'
```

（将来用）Kong **DB モード**での注意点

DB モードに切替えるときは：

- `KONG_DATABASE=postgres` を指定
- `KONG_DECLARATIVE_CONFIG` は **外す**
- 初回のみ **マイグレーション**が必要

```
# （DBモードのみ初回）
docker compose run --rm kong kong migrations bootstrap
```

宣言的設定を DB に反映するときは **deck** を使用（ファイル：`kong/kong-deck.yaml`。select-tag で対象限定）。

```
# 差分確認
deck gateway diff kong/kong-deck.yaml --select-tag stack-multi-authn
# 反映
deck gateway sync kong/kong-deck.yaml --select-tag stack-multi-authn --yes
```

> ☑ **DB と DB-LESS を同時に設定しない** `KONG_DATABASE=postgres` と `KONG_DATABASE=off`／`KONG_DECLARATIVE_CONFIG` を同居させないでください。

Keycloak（realm: `demo-realm`）

`docker-compose.yaml` で `./realms/demo-realm.json` を **自動インポート**します。 同 JSON にはユーザ例が含まれます：

- `testuser`（パスワード `password`、A/B に `read`,`user` 付与）
- `dummyuser`（パスワード `password`、`dummy`,`user` 付与）→ **A/B の `read` が無い**ため権限エラー確認に使えます

> もし `quarkus-client` を **Confidential** にする場合は、Keycloak 管理画面で クライアント種別を変更して **Secret** を発行し、BFF の `application.properties` に反映してください。

アプリのポイント

Quarkus（BFF）

- OIDC Code Flow（`/login` で受け、認証後は `/app/` に戻す）
- セッションは **Redis Token State Manager** を使用（Cookie は参照キーのみ）
- **トークン転送**：BFF はログイン中の **AccessToken** を取り出して **下流クライアントに `Authorization: Bearer …` を手動で付与**（MicroProfile Rest Client）
- **権限エラー整形**：下流が 401/403 の場合、BFF は **403** を返すようにハンドリング → フロントは「権限エラー」を表示
- `/api/me` は **roles を集約して返却**（`realm_access.roles` + `resource_access.*.roles`）

Frontend（Vite/React）

- `vite.config.ts` の `base: '/app/'`
- `/api/*` 呼び出しのエラーハンドリングで **401/403 → 権限エラー表示**

動作確認手順

1. `http://localhost:8000/app/`
2. 「ログイン」→ `testuser/password` でログイン
3. 「/api/me」→ `roles` に `read`,`user` が含まれること
4. 「/api/mashup」→ A/B の結果が返ること
5. ログアウト → `dummyuser/password` でログイン
6. 「/api/mashup」→ **権限エラー**（403）が表示されること

トラブルシューティング

- **`/app/` が 301 ループ** → フロントのルートを **`/app` と `/app/` の両方**登録、`strip_path=false`、`path_handling=v1`。
- **Kong が起動時に `…/kong.yml: Is a directory`** → `KONG_DECLARATIVE_CONFIG` に **ファイル**をマウントしているか確認（ディレクトリ不可）。
- **Keycloak 反応が遅く BFF が接続失敗** → compose の `depends_on` と **healthcheck** を設定（本リポジトリは設定済み）。
- **`/login` で 404** → `login-route` が **`strip_path=false`** で BFF に素通しされているか。
- **roles が `/api/me` に出ない** → BFF は JWT から `realm_access.roles` と `resource_access.*.roles` を集約。 Keycloak 側で対象クライアントのロール付与/Token に `aud` が載っているか確認。

参考：主要ファイル

- `docker-compose.yaml` … 全体起動（DB-LESS前提）
- `kong/kong.yml` … DB-LESS の宣言的設定（Service/Route）
- `kong/kong-deck.yaml` … **DB モード用**の deck 状態ファイル
- `quarkus-authz/src/main/java/example/api/MeResource.java` … ユーザ情報（roles 集約）

- `quarkus-authz/src/main/java/org/example/api/MashupResource.java` … 下流呼び出し & 401/403 → 403 整形
- `frontend/src/api.ts` … 401/403 を「権限エラー」にマップ
- `frontend/vite.config.ts` … `base: '/app/'`

ライセンス

本リポジトリ内のコードは学習目的のサンプルです。必要に応じてコピー/改変してご利用ください。

付録：手元で Kong の状態を確認するワンライナー

```
# Routes
curl -s http://localhost:8001/routes \
  | jq '.data[] | {name, paths, strip_path, preserve_host, path_handling, service:
.service.id}'

# Services
curl -s http://localhost:8001/services \
  | jq '.data[] | {name, host, port, path}'
```

付録：Redis に保管されたセッション情報を確認する（開発時の手引き）

BFF（Quarkus）は **Redis Token State Manager** でセッション相当のトークン状態を保存します（Cookie には参照キーのみ）。 開発時に中身や存続時間を確認したい場合は、以下の手順で Redis を操作します。

> ⚠本番では `KEYS *` や `MONITOR` は避けてください。**`SCAN` を使う**のが安全です。 このリポジトリの Compose では `redis` サービス名で起動しています。

1) Redis CLI を開く

```
# コンテナ上の redis-cli を開く
docker compose exec redis redis-cli
```

2) キーを探す（SCAN 推奨）

ログイン直後にセッション関連キーが作られます。キー名はバージョンで多少変わるため、**ワイルドカードで探す**のがコツです。

```
# まずは全体をざっと（開発専用）
SCAN 0 MATCH * COUNT 100

# よく使うパターン例（いずれかにヒットするはず）
```

```
SCAN 0 MATCH *session* COUNT 100
SCAN 0 MATCH *oidc*    COUNT 100
SCAN 0 MATCH *token*   COUNT 100
```

> ブラウザの Cookie にある **q_session** の値（localhost:8000 ドメイン）を使うと特定が早いです。
> Cookie 値の一部でマッチさせる例：SCAN 0 MATCH *<q_sessionの値の先頭数文字>* COUNT 100

3) キー内容と TTL を見る

キーの **種類** によって読み方が異なります。TYPE で判定してから GET（string）または HGETALL（hash）を使います。

```
# 例：見つけたキーを K とする
TYPE K
TTL  K          # 残存秒数（例：1200秒 ≒ 20分）

# 文字列キーの場合
GET K

# ハッシュキーの場合（フィールドと値の一覧）
HGETALL K
```

> 値は JSON かシリアライズ済みの文字列です。GET の表示をそのまま参考にし、意味が分かりにくい場合は **TTL** で寿命だけ確認するのが手堅いです。

4) 特定セッションだけ削除（開発時の再現用）

```
DEL K
```

> ログアウトや消し込みの再現をしたい時に便利です。

5) すべてのセッションを消す（開発専用）

```
FLUSHDB     # 現在のDBを全消去（開発専用）
# もしくは
FLUSHALL    # すべてのDBを全消去（要注意）
```

> Compose の redis は ./data/redis を永続化しています。完全にリセットしたいときは コンテナ停止後にそのディレクトリを削除する方法でも初期化できます。
