# 認証関連の機能の作成に対して取り組んだ内容

このシステムでは、セキュリティを確保するためにユーザーの認証機能や、使用できる機能の権限の管理機能を設けております。

この資料の目的は、安全性の高いシステムにするうえで工夫したことをプレゼンすることです。
主にJWT認証の内容に関して記述します。

目次



なお、このシステムではセキュリティの管理で基本的な以下の要件は満たしております。

・パスワード文字列のハッシュ化（BCrypt関数を用いています）

・ユーザーの権限の管理（ユーザーの持っている権限に応じて、アクセスできる機能を制限しています）

・CSRF対策（JWTトークンを用いていれば自ずと有効になります）

## 採用したユーザーの認証方法

選定したのは、**JWTトークンによる認証**となります。

理由としては以下のようになります。

### このアプリケーションの完全なステートレスを確保できる

この認証方法は、作成したトークンを署名鍵で検証し、トークン内の認証情報を信頼してそのまま使用します。
そのため、サーバー側にセッションIDといったユーザーごとの情報を保持しておく必要がなく、ステートレスなシステムを作れます。

### アプリケーションサーバーのスケールアウトが容易

先述の完全なステートレスであるという特徴から、複数のサーバー同士でユーザー情報を事前に共有しておく必要がありません。そのため、サーバー間でデータを共有するようなデータベースなどを用意する必要がなくサーバーの増加が容易なので、アクセス数に応じた柔軟なリソース増加が可能です。

### サーバー側のメモリの負担を軽くできる

ステートレスであるという事は、セッションIDなどをサーバー側のメモリに保持しておく必要がないため、メモリの負担を軽くでき、他の処理にリソースを割くことが可能になります。

## JWTトークンのセキュリティの確保

対策した内容としては、以下のようになります。

### 公開鍵を用いた、ブルートフォース攻撃への対策

使用できる署名鍵には、共通鍵も使用が可能でした。ですが、共通鍵の場合暗号強度が少ない場合があり、
ブルートフォース攻撃でトークンを改ざんできてしまう恐れがある為です。

### JWTトークン検証時に用いる暗号鍵の厳密な指定

JWTトークンの改ざんによる攻撃の手法としては、「トークン内の使用署名鍵の欄を空白または違う種類の鍵にして、検証を不正に免れる」という物がありますが、検証に用いる鍵を厳密に指定する事で対策してます。

### 検証に用いる暗号鍵は、絶対にサーバー内に保存してあるものを用いる

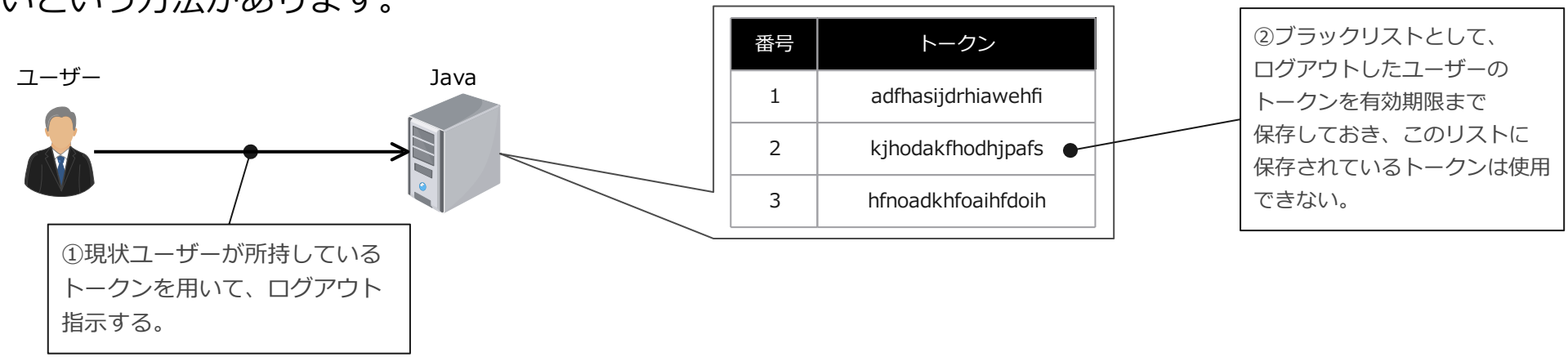
JWTには、「トークン自体に使用する暗号鍵を入れて、それで検証を行う」といった機能が存在します。
ですが、これには「トークン内の暗号鍵が不正に書き換えられて、それで検証をしてしまい、不正に合格する」といった危険性があります。それを回避するために、絶対にサーバー内に保存してある物を使用します。

### トークンで指定されたファイルパスの暗号鍵は、絶対に使用しない

JWTには「ファイルパスをトークンで指定して、そのファイルパスの鍵を使用する」といった機能が存在します。
ですが、これには「ファイルパスが書き換えられてしまい、ディレクトリトラバーサル攻撃をされる」といった危険性があります。それを回避するために、トークンからの指定のファイルパスの暗号鍵は使用せず、
サーバー内であらかじめ定義してあるファイルパスの暗号鍵を使用します。

## JWTトークンの無効化

システムのログアウト時には、トークンの無効化を行い、再度同じトークンを使えないようにすることが必要です。代表的な方法としては、トークンのブラックリストを作成し、登録されているトークンは使用しないという方法があります。




ですが、この方法ではステートレス性を確保できず、JWT認証を使用するメリットがなくなってしまいます。
（無効なトークンをサーバー側に保持している時点で、ステートレスではないからです）

これの対策として、トークンの有効期限を極めて短く設定し運用する方法を採用しています。
そうすれば、ブラックリストで無効なトークンを保存するまでもなく、トークンをすぐに無効化できます。

ただし欠点として、フロント側がトークンの有効期限が切れないうちに、定期的に非同期で新しいトークンを取得し続ける必要があり、ユーザビリティが下がります。（改善予定）