Spring Security 入門

Spring Boot アプリケーションにフォーム認証と権限管理を実装しよう

実装するアプリケーション

Spring Boot
Spring Framework
Spring Data
Spring Cloud
Spring Cloud Data Flow
Spring Security
Spring Security Kerberos
Spring Security OAuth
Spring Security SAML
Spring GraphQL
Spring Session
Spring Integration
Spring HATEOAS
Spring REST Docs
Spring Batch
Spring AMQP
Spring CredHub
Spring Flo
Spring for Apache Kafka

Spring Security 5.6.0

OVERVIEW | LEARN | SUPPORT

Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.

Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements

Features

- Comprehensive and extensible support for both Authentication and Authorization
- Protection against attacks like session fixation, clickjacking, cross site request forgery, etc
- Servlet API integration
- Optional integration with Spring Web MVC
- Much more...

認証　認可

ハッシュ
ソルト
ストレッチング

認証

- 方式：HTTP認証 / フォーム認証 / TLS認証
  - この講座ではフォーム認証（HTMLでIDとパスワードを入力する）をテーマとしています
- 機能要件
  - ログイン
  - ログアウト
- 非機能
  - 安全なパスワードの保存方法

カスタムログインページの設定方法

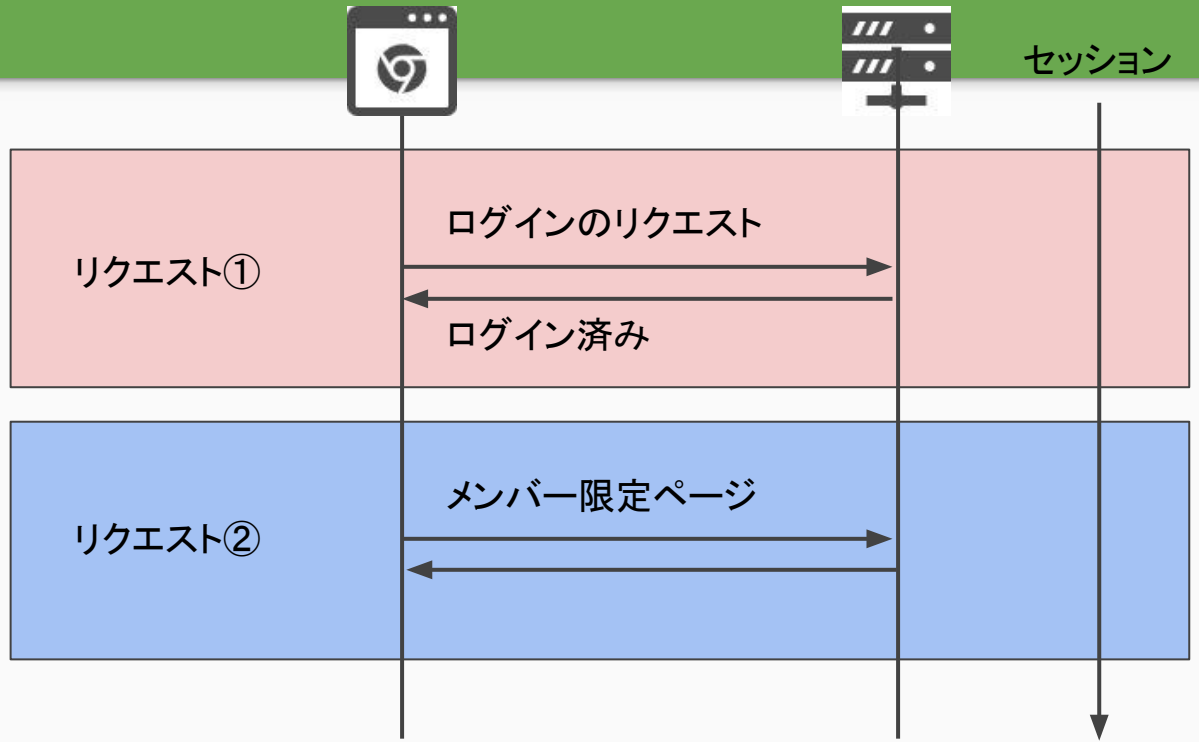
- https://docs.spring.io/spring-security/site/docs/current/guides/form-javaconfig.html#configuring-a-custom-login-page

セッションとは？

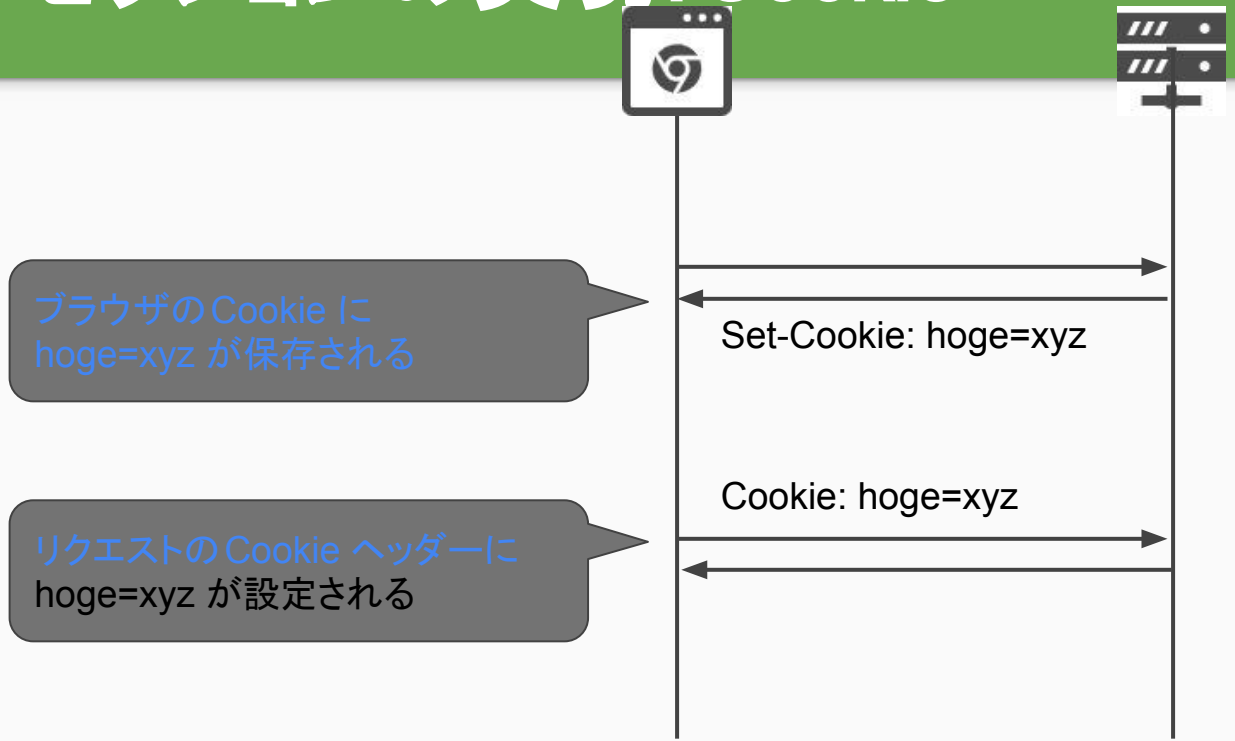
Webアプリの概観：セッション（リクエストの集まり）

HTTPはステートレス
= 状態を扱う仕組みがない
= 前のリクエストの情報を
参照できない

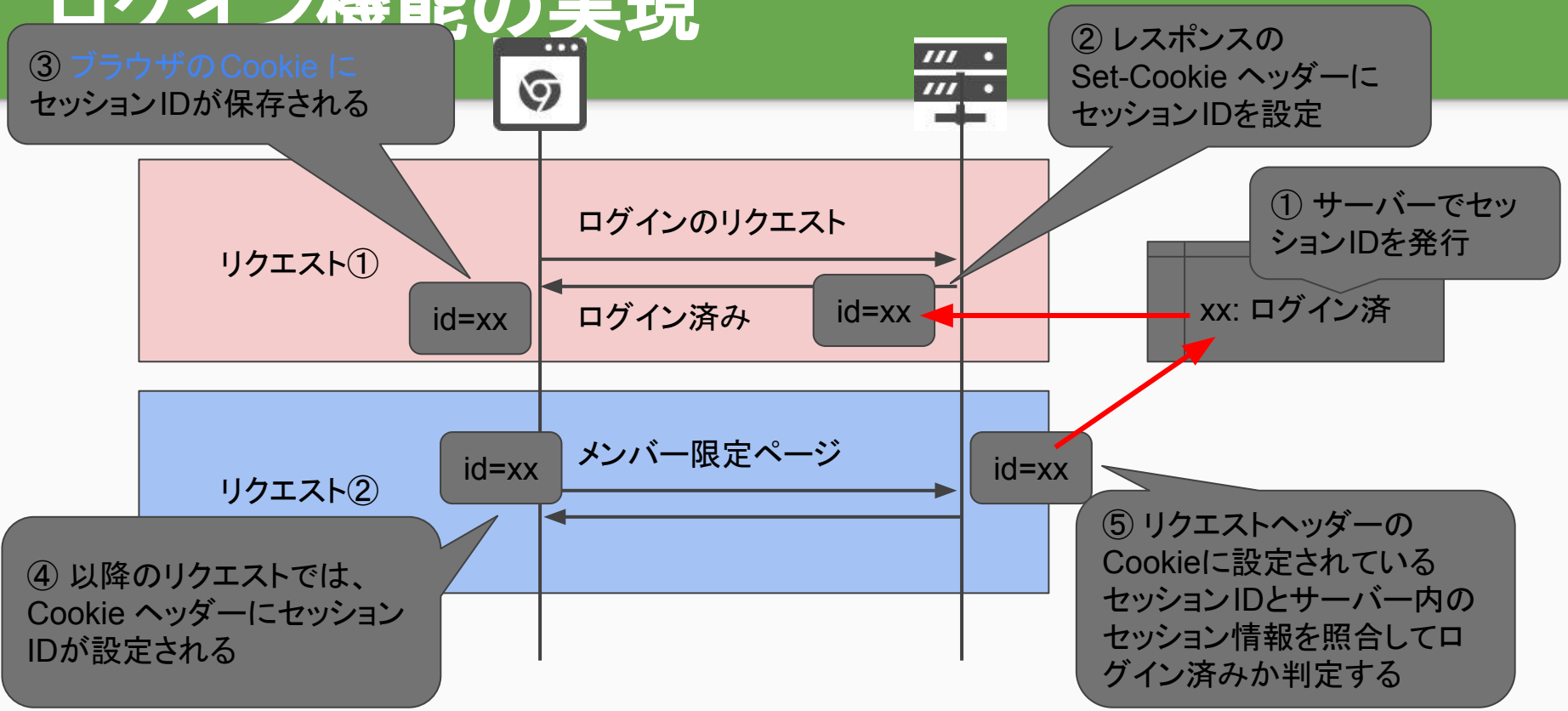
リクエスト①と②の送信元が
同じであるかは分からない
→ ログイン済みか分からない



セッションの実現：Cookie



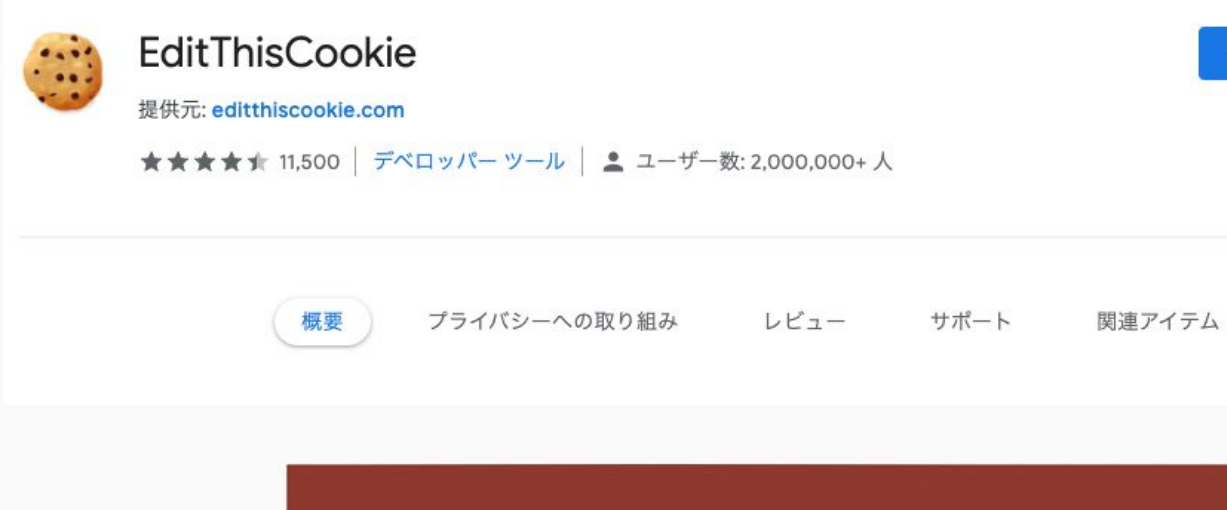
ログイン機能の実現


③ ブラウザのCookie に
セッションIDが保存される
② レスポンスの
Set-Cookie ヘッダーに
セッションIDを設定
① サーバーでセッ
ションIDを発行
ログインのリクエスト
リクエスト①
id=xx
ログイン済み
id=xx
xx: ログイン済
メンバー限定ページ
id=xx
リクエスト②
id=xx
④ 以降のリクエストでは、
Cookie ヘッダーにセッション
IDが設定される
⑤ リクエストヘッダーの
Cookieに設定されている
セッションIDとサーバー内の
セッション情報を照合してロ
グイン済みか判定する

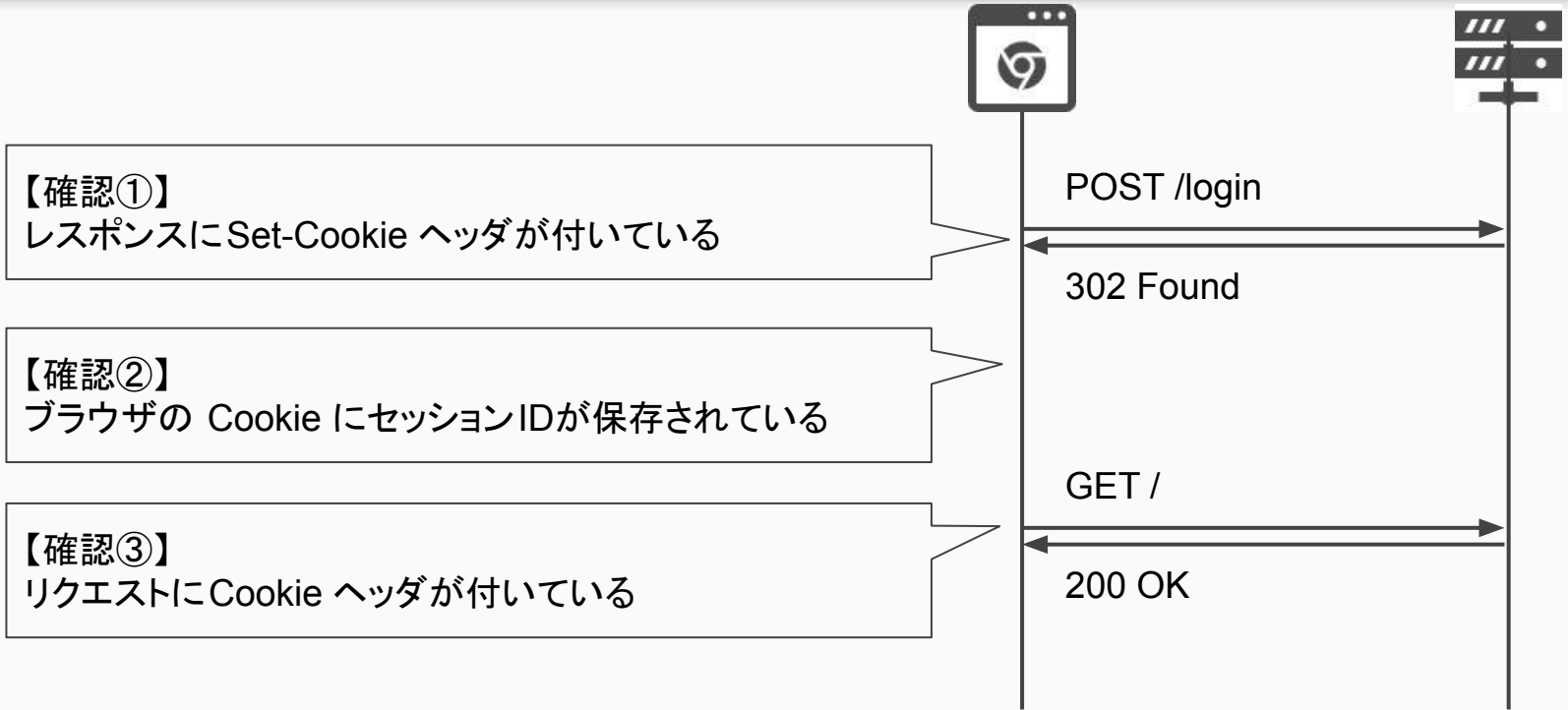
Cookieの挙動をデバッグしてみよう

EditThisCookie:
Cookie 確認のための Chrome extension

https://chrome.google.com/webstore/detail/editthiscookie/fngmhnnpilhplaeedifhccceomclgfbg



セッションの動作をデバッグしてみよう



セッションのセキュリティ要件

参考資料

- 「安全なWebアプリケーションの作り方」（情報処理推進機構）
  - https://www.ipa.go.jp/security/vuln/websecurity.html
  - 資料のダウンロード > 「安全なウェブサイトの作り方」 > PDF

補足：セッションIDをURLパラメーターに格納しない

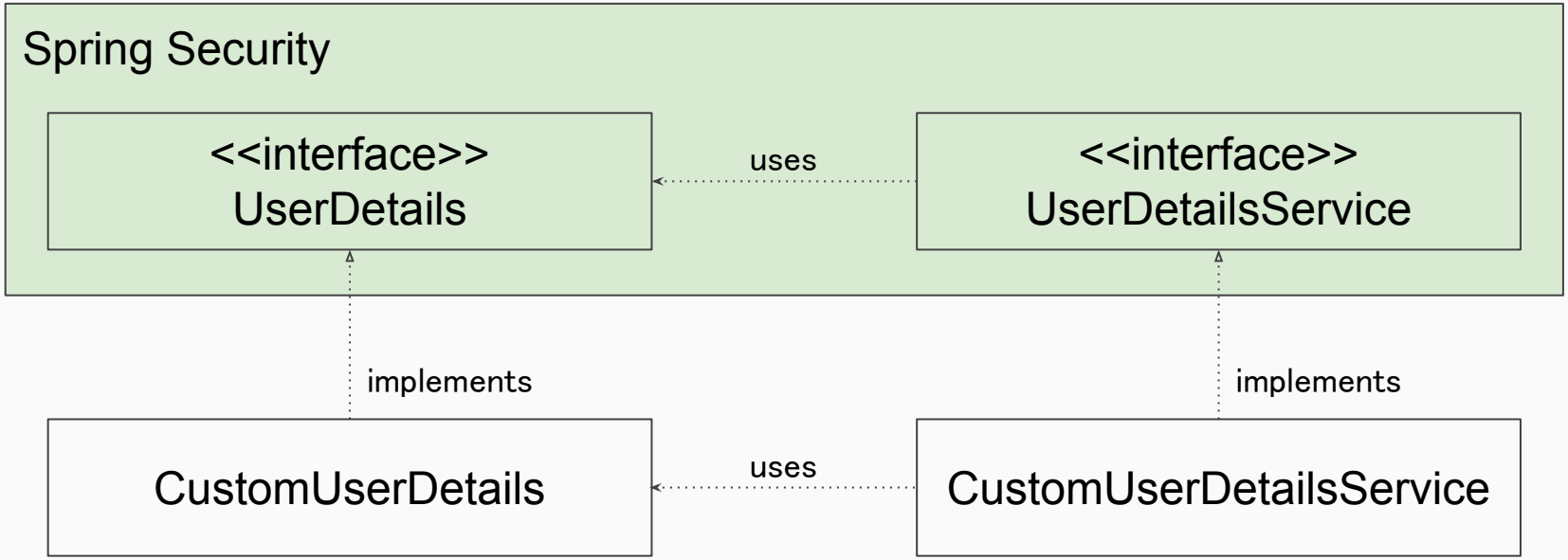
- NG：http://localhost:8080?JSESSIONID=hogehogehoge
- Cookie を使っていればOK
  - SpringはCookieを使っているので対応済み

ユーザー情報を
データベースで管理しよう

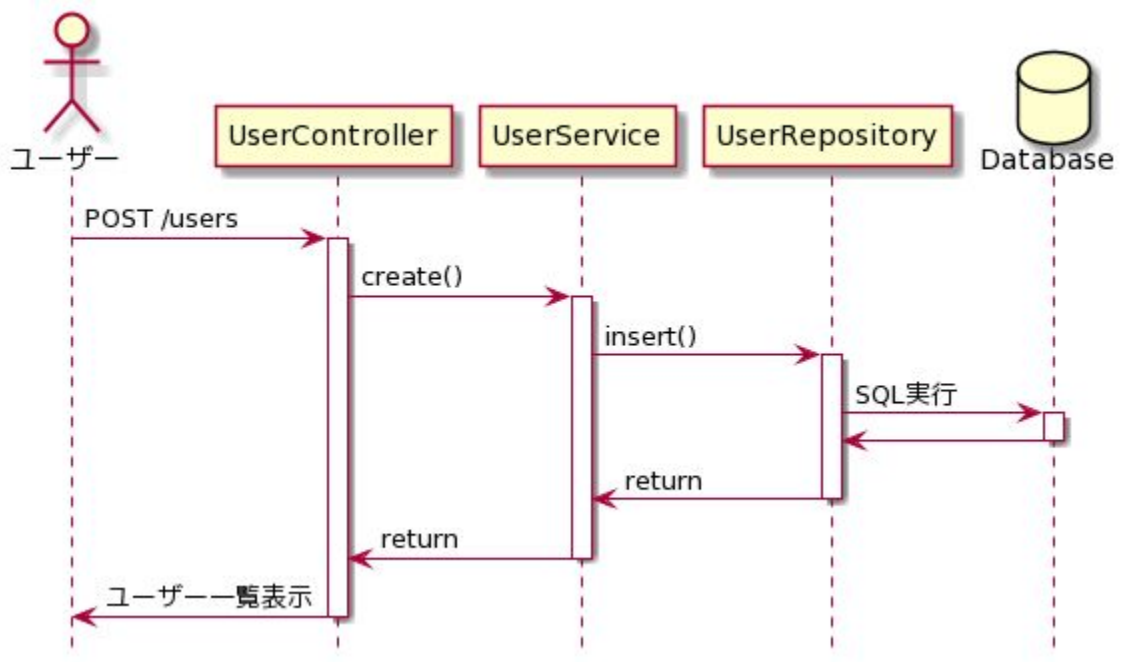
重要なインターフェース

- UserDetails
  - ユーザー名/ パスワード / 権限 などの情報を保持する
  - ref. API Doc
- UserDetailsService
  - UserDetails を取得するメソッドを持つ (loadUserByUsername)
  - ref. API Doc

登場人物の関係


Spring Security
<<interface>>
UserDetails
uses
<<interface>>
UserDetailsService
implements
implements
CustomUserDetails
uses
CustomUserDetailsService

5-2. ユーザー作成のシーケンス図



Spring Security + Thymeleaf

https://www.thymeleaf.org/doc/articles/springsecurity.html

5-11. Custom constraints

https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#validation-beanvalidation-spring-constraints

```
@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy =)
public @interface UniqueUsername {
    String message() default "";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
```

パスワードを安全に管理しよう

パスワードを取り扱う上での観点

1. 弱いパスワードを設定できないようにする
2. パスワードを安全にデータベースに保存する

弱いパスワードを設定できないようにする

- 文字数を制限する：12文字以上（+ 128文字以下）
  - ref. JPCERT/CC
  - 12文字以上 = セキュリティ観点による制限
  - 128文字以下 = システム実装観点による制限
- ユーザー作成時のバリデーションでチェックする

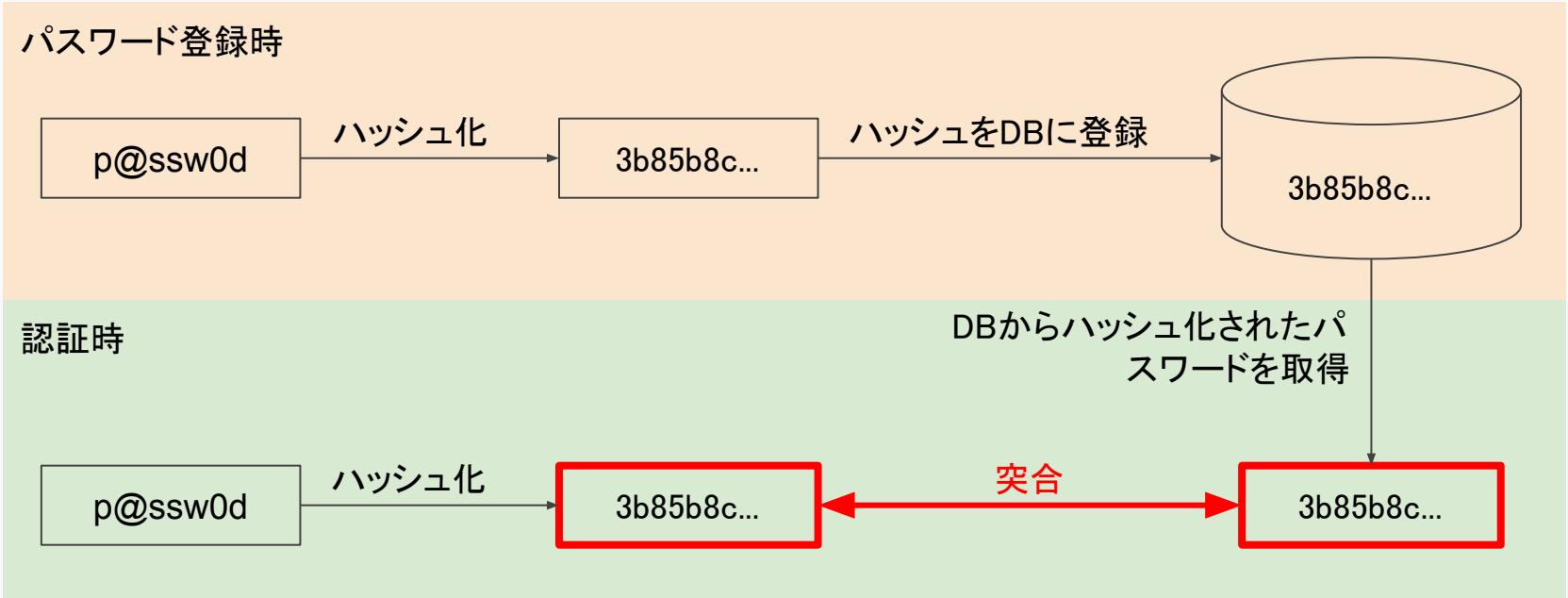
パスワードを安全に保存する

ハッシュ
ソルト
ストレッチング

ハッシュ

- ハッシュとは？
  - 与えられた文字列を別の文字列に変換すること
    - ex) p@ssw0rd → 3b85b8c…
  - 次の性質をみたす
    - ハッシュ値から元になった入力を求めることが難しい
    - 異なる入力から同じハッシュが得られることが極めて少ない
- ハッシュの目的
  - データベースの情報が盗まれても、元のパスワードが流出しない

ハッシュを使った認証


パスワード登録時
p@ssw0d
ハッシュ化
3b85b8c...
ハッシュをDBに登録
3b85b8c...
認証時
DBからハッシュ化されたパスワードを取得
p@ssw0d
ハッシュ化
3b85b8c...
突合
3b85b8c...

ソルト

- ソルトとは？
  - 元のパスワードに追加する、ある程度の長さの文字列
    - ex) 663e6fa020f9add4 + p@ssw0rd
  - ユーザーごとに異なる値にする
- ソルトの目的
  - 長いパスワードにできる（20文字以上）
  - 同じパスワードのユーザーでもハッシュ値が異なるようにする

ストレッチング

- ストレッチングとは？
  - 元のパスワードに繰り返しハッシュ計算を適用すること
    - ストレッチングなし：hash(“p@assw0rd”) → 3b85b8c…
    - ストレッチングあり：hash(hash(hash(“p@assw0rd”))) → …
- ストレッチングの目的
  - ハッシュ計算を遅くし、総当たり攻撃を防ぐ

まとめ：パスワードの要件

- 12文字以上、128文字以下
- ハッシュ + ソルト + ストレッチング

認可

認可（Authorization）とは？

- ざっくり言うと『アクセス制御』
- 詳しく言うと『認証されたユーザーに権限を与えること』
  - ex）部長以上は稟議申請を承認できる。それ以外は承認できない
  - ex）自分のコメントは閲覧・編集が可。他人のコメントは閲覧のみ可

認可の実現方法

- 特定のURLへのアクセスを限定する
  - ex) /admin/ 以下のパスには管理者アカウントのみがアクセスできる
- 特定のメソッドの実行を限定する
  - ex) userService.findAll メソッドは管理者アカウントのアクセス時のみ実行可能
- 画面上に操作する動線を置かない（抜け道あり）
  - ex) 一般ユーザーがログインしているときは、ユーザー作成ボタンを表示しない

Spring での認可の実装

- ユーザーとロールを紐付ける
  - Tom = Admin（管理者ロール）
  - Bob = User（一般ユーザーロール）
- ロールを使ってアクセス制御をする
  - ex) Admin ロールを持つユーザーのみ、ユーザー追加可能

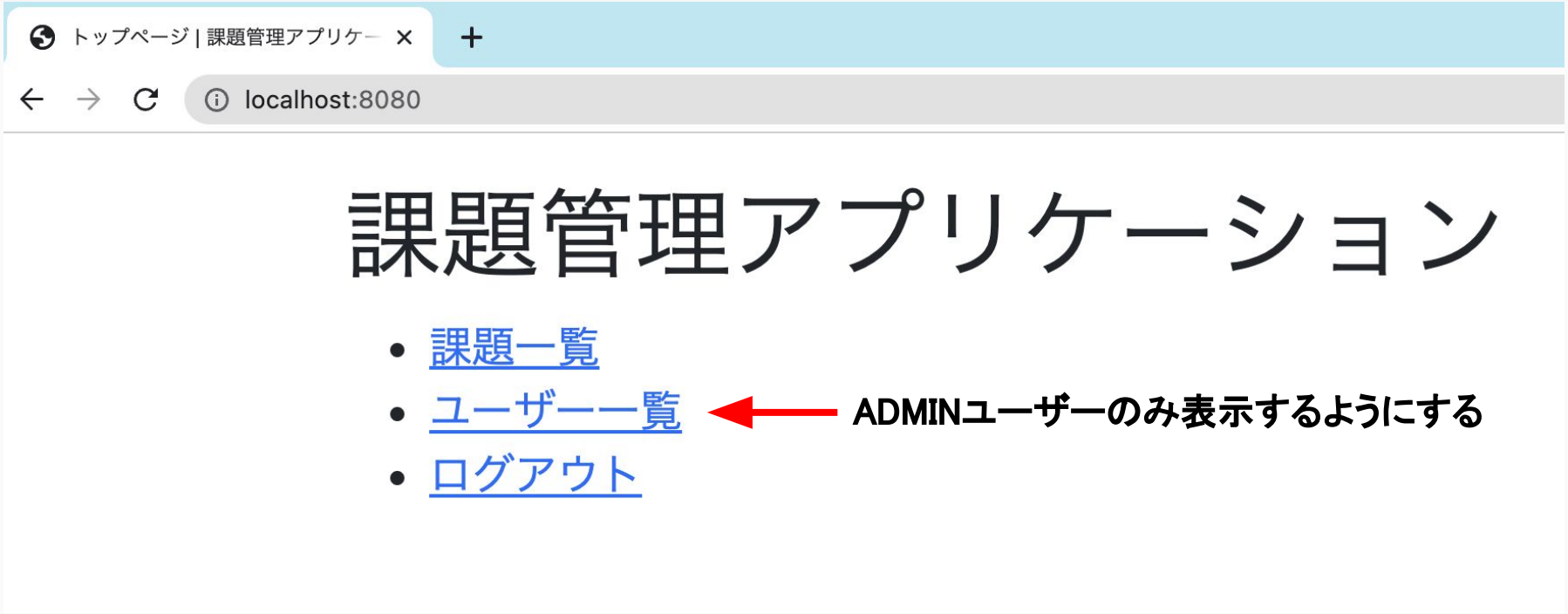
権限によって
画面の表示内容を変えよう

Thymeleaf with Spring Security

- ref. Thymeleaf + Spring Security integration basics

```
<div sec:authorize="isAuthenticated()">
  This content is only shown to authenticated users.
</div>
<div sec:authorize="hasRole('ROLE_ADMIN')">
  This content is only shown to administrators.
</div>
<div sec:authorize="hasRole('ROLE_USER')">
  This content is only shown to users.
</div>
```

実装イメージ


トップページ | 課題管理アプリケー
localhost:8080
課題管理アプリケーション
課題一覧
ユーザー一覧
ADMINユーザーのみ表示するようにする
ログアウト

xmlns の追加

- “sec” を使用するために、以下を index.html の html タグに追加
- xmlns:sec=″http://www.thymeleaf.org/thymeleaf-extras-springsecurity5″

Custom Error Page

- Spring Boot Reference Documentation > Custom Error Pages

```
src/
 +- main/
     +- java/
     |   + <source code>
     +- resources/
         +- public/
             +- error/
             |   +- 404.html
             +- <other public assets>
```