

React実習

09. Reactとログイン認証

株式会社ジードライブ

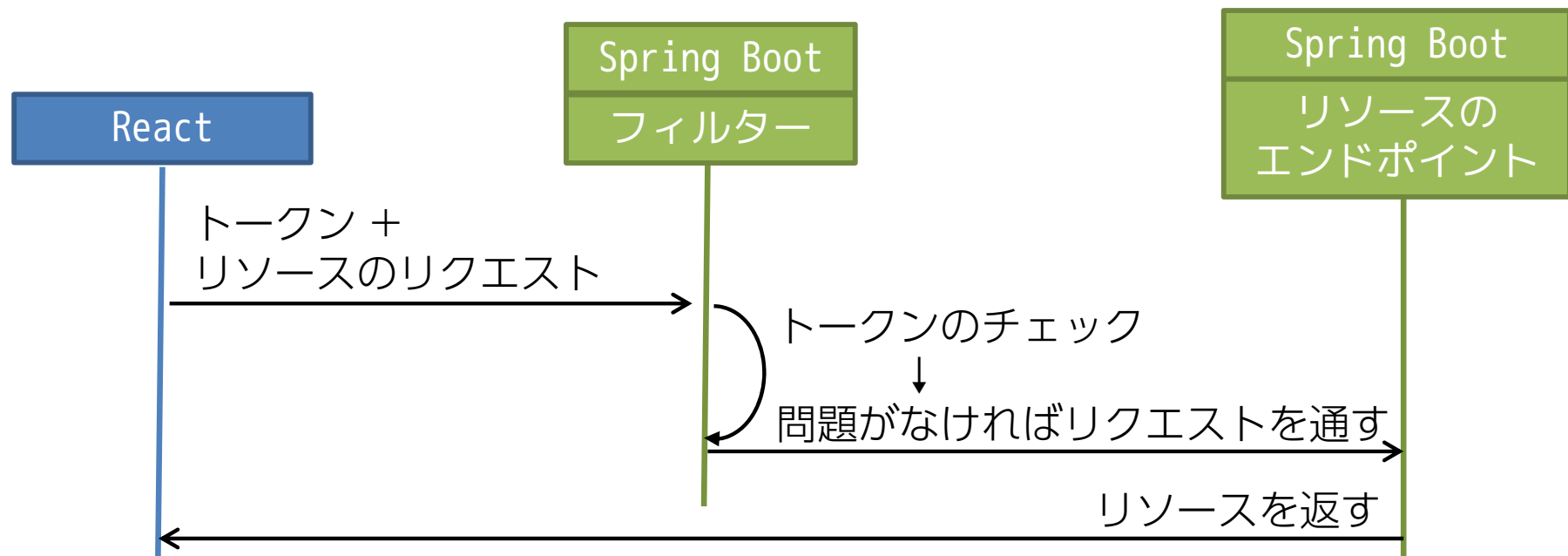
今回学ぶこと

- ReactとSpring Bootを使用したログイン認証
 - セッションやWebフィルターを使い、Web APIへのアクセスを制御する方法

Web APIへのアクセス制御

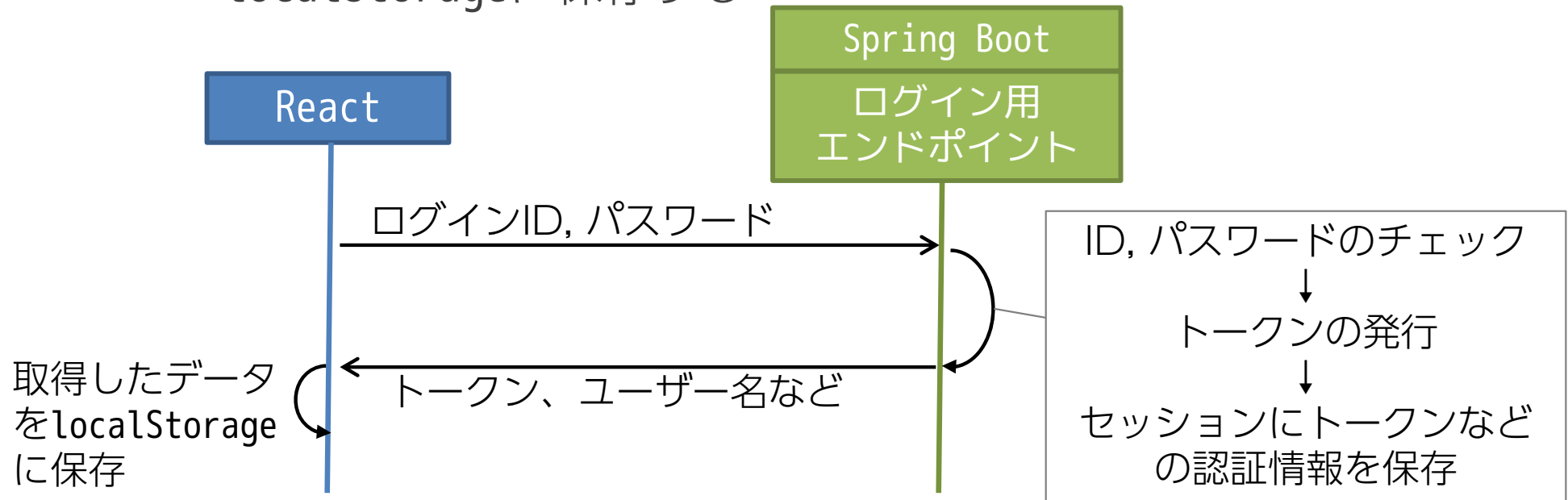
- Webフィルターを使用することで、Web APIへのアクセス制御ができる
 - React: リクエスト時に**トークン**を送信する
 - Spring Boot: Webフィルターでトークンをチェックする

一時的な認証・認可に必要な手形



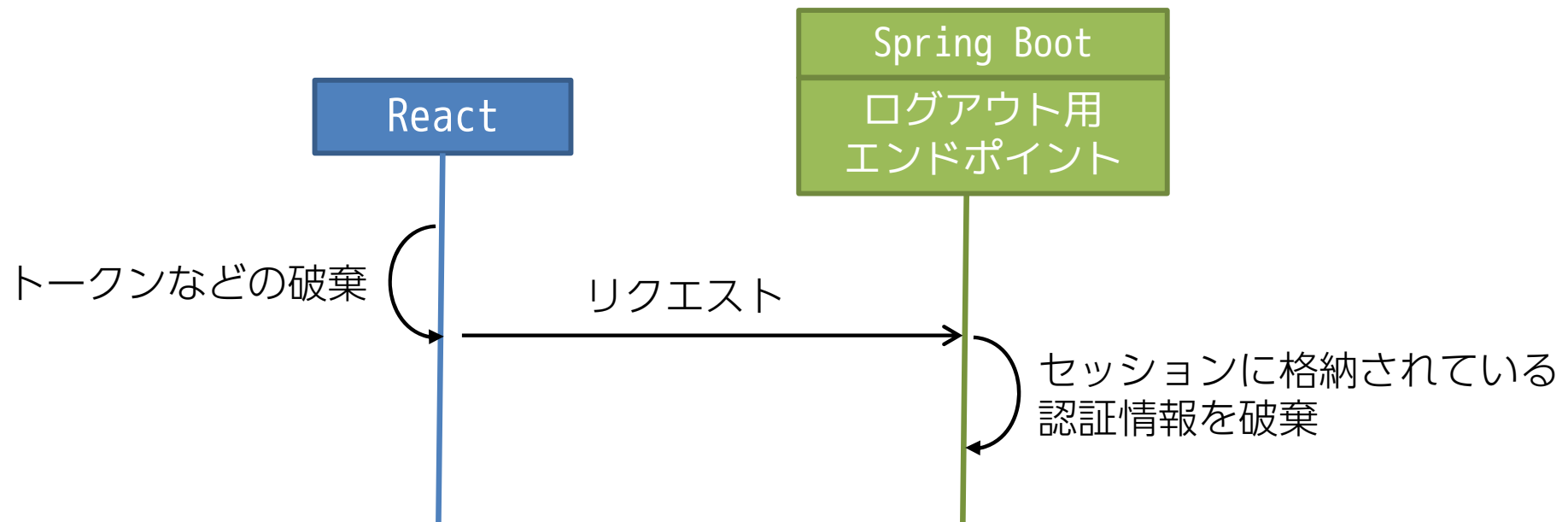
ログイン処理

- ReactからSpring Bootで用意したログイン用のエンドポイントに、認証情報を送信する
 - 認証情報に問題がなければ、トークンを発行し、必要な情報とともにセッションに保存する
 - トークンやユーザー名などReact側で必要なデータを返す
⇒ localStorageに保存する



ログアウト処理

- localStorageに保存しているトークンなどの認証情報を破棄しつつ、ログアウト用のエンドポイントに、リクエストを送信する
 - Spring Boot側でもセッションに格納している認証情報を破棄する必要がある



Spring Boot側の実装例

ログイン処理

- ログインIDやパスワードをチェックし、トークンを発行する

```
@PostMapping("/api/login")
public Map<String, String> login(
    @RequestBody @Valid User user, Errors errors) {
    Map<String, String> response = new HashMap<>();

    User authUser =
        service.getAuthUser(user.getLoginId(), user.getLoginPass());

    /* 不正なログインID・パスワードの場合: */
    if(authUser == null) {
        response.put("status", "error");
        return response;
    }
    ...
}
```

Spring

ログイン処理

前頁の続き

```
/* 不正なログインID・パスワードの場合: */
```

```
...
```

```
/* 正しいログインID・パスワードの場合: */
```

```
//ユーザー情報を、トークンをキーとしてセッションに保存
```

```
String token = generateToken();
```

```
session.setAttribute(token, authUser);
```

```
// React側で必要な情報(トークン、ユーザー名等)を返す
```

```
response.put("status", "ok");
```

```
response.put("token", token);
```

```
response.put("userName", authUser.getName());
```

```
return response;
```

```
}
```

Spring

トークンの生成

- Webアプリケーションでは、JWT(JSON Web Token)という形式のトークンが広く利用されている
 - JWTではトークン自体に、その有効期限やユーザー名など様々な情報を含めることができるが、実装の難易度は高い
- UUIDを使った簡易的なトークン生成の実装例
 - UUIDは、時刻やハードウェアなどの情報を組み合わせて生成されるため、衝突する可能性が極めて低い(一意となる識別子として利用可能)

```
private String generateToken() {  
    return UUID.randomUUID().toString();  
}
```

Spring

ログアウト処理

- 認証情報をセッションから削除する

```
@GetMapping("/api/logout")
public String logout() {
    session.invalidate();
    return "logout done";
}
```

Spring

フィルターの処理

- トークンの確認をする

Filterインターフェース
を実装するよりも便利

```
public class AuthFilter extends OncePerRequestFilter {  
  
    @Override  
    protected void doFilterInternal(HttpServletRequest request,  
                                    HttpServletResponse response, FilterChain filterChain)  
        throws ServletException, IOException {  
  
        // プリフライト・リクエストについては認証処理を無視する  
        if (request.getMethod().equals("OPTIONS")) {  
            filterChain.doFilter(request, response);  
            return;  
        }  
        ...  
    }  
}
```

Spring

リクエストの安全確認のために、ブラウザが自動送信する

フィルターの実装

前頁の続き

トークンがAuthorizationヘッダとして送られてくることを想定

Spring

```
...
// トークンの取得
String token = request.getHeader("Authorization");
// トークンが存在しない、またはトークンが不正
if (token == null ||
    request.getSession().getAttribute(token) == null) {
    response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
    return;
}

filterChain.doFilter(request, response);
}
}
```

設定ファイルの実装

- CORS設定とフィルターの有効化を行う

```
@Configuration
public class ApplicationConfig implements WebMvcConfigurer {

    // CORS設定
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
            .allowedOrigins("http://localhost:3000")
            .allowedMethods("GET", "POST", "PUT", "DELETE")
            .allowedHeaders("Authorization", "Content-Type")
            .allowCredentials(true);
    }

    ...
}
```

Spring

ログイン認証を実装する際には必須

設定ファイルの実装

前頁の続き

Spring

```
...  
  
// 認証用フィルタの有効化  
@Bean  
FilterRegistrationBean<AuthFilter> authFilter() {  
    var bean =  
        new FilterRegistrationBean<AuthFilter>(new AuthFilter());  
    bean.addUrlPatterns("/api/items/*");  
    return bean;  
}  
}
```

練習問題

- 練習09-1
- 練習09-2
- 練習09-3

React側の実装例

ログイン処理

- ログイン成功時に、トークンなどのデータを保存する

```
const setLoginUser = useSetAtom(loginUserAtom);
const url = "http://localhost:8080/api/login";
const user = { loginId: "taro", loginPass: "pass" };
const {data} = await axios.post(url, user, {
  withCredentials: true,
  headers: { "Content-Type": "application/json" }
});

// ログイン成功時の処理
if(data.status === "ok") {
  setLoginUser({userName: data.userName,
    token: data.token});
  ...
}
```

React

ログイン処理

- セッションを利用するためwithCredentialsをtrueを設定する

```
const setLoginUser = useSetAtom(loginUserAtom);
const url = "http://localhost:8080/api/login";
const user = { loginId: "taro", loginPass: "pass" };
const {data} = await axios.post(url, user, {
  withCredentials: true,
  headers: { "Content-Type": "application/json" }
});
```

```
// ログイン成功時の処理
if(data.status === "ok"){
  ...
}
```

React

この設定によりクロスオリジン
(localhost:3000⇒localhost:8080)へのクッキー
送信(セッションの利用)が可能になる

fetch()を使う場合は、credentials: 'include' を設定

Atomの作成

- ログイン成功時に取得したトークンなどのデータはステートとしてだけでなく、localStorageにも保存をする
 - これにより、ブラウザのリフレッシュにも対応できるようになる
 - atomWithStorageを利用することで、データをステート(アトム)としてセットする際に、localStorageへの保存も行える

```
import { atomWithStorage } from "jotai/utils";  
  
export const loginUserAtom =  
  atomWithStorage("loginUser", null);
```

React

ログアウト処理

- localStorageやステートからトークンなどを削除する
- ログアウトのためのエンドポイントにリクエストを送る

```
const setLoginUser = useSetAtom(loginUserAtom);
```

React

```
// ステートとローカルストレージをクリアする  
setLoginInfo(null);  
localStorage.clear();
```

```
// ログアウト用のエンドポイントにリクエストを送る  
const url = "http://localhost:8080/api/logout";  
await axios.get(url, {withCredentials: true});
```

```
... トップページ等へリダイレクト
```

認証が必要なAPIの利用

- localStorageから取り出したトークンをヘッダに設定する
 - Axiosのインターセプターを使用することで、リクエストやレスポンスに際しての共通処理を記述できる
 - <https://axios-http.com/ja/docs/interceptors>

```
// ベースの設定
const customAxios = axios.create({
  baseURL: "http://localhost:8080/api/items",
  headers: {
    "Content-Type": "application/json"
  },
  withCredentials: true
});
```

React

… リクエスト・インターセプターを使い、ヘッダにトークンを設定する

認証が必要なAPIの利用

前頁の続き

React

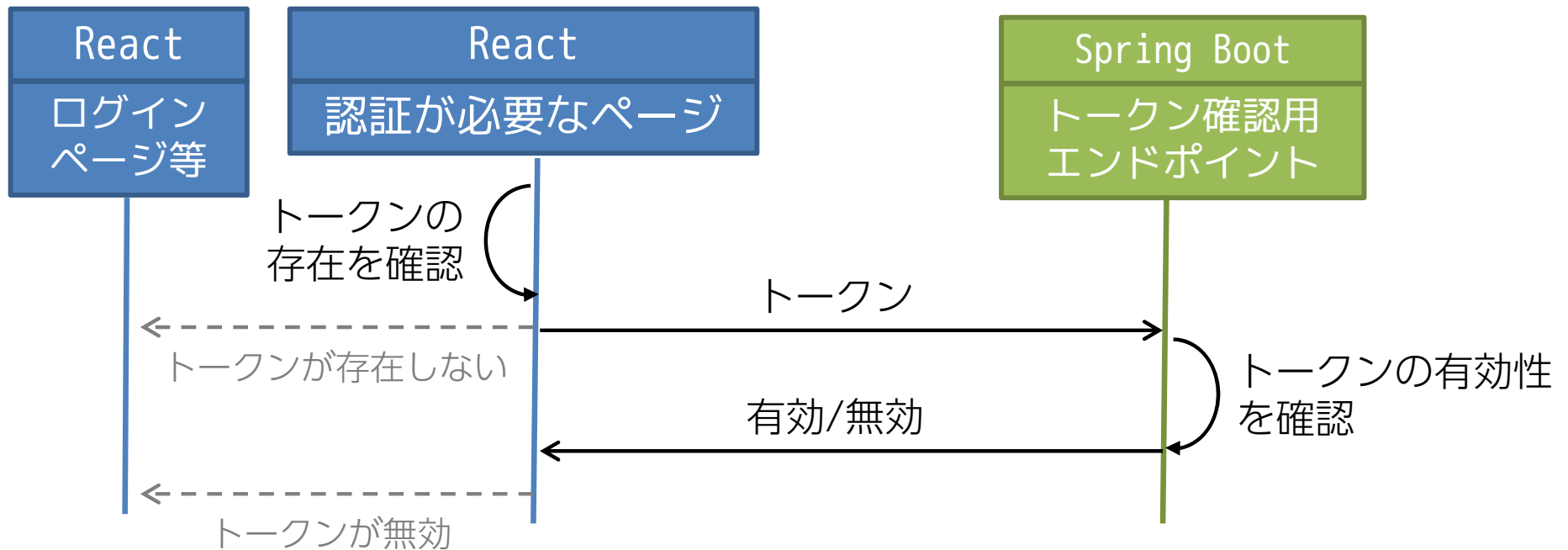
```
...
// リクエスト・インターセプターでトークンを自動追加
customAxios.interceptors.request.use(
  (config) => {
    // ローカルストレージからトークンを取得
    const loginUserInStorage =
      JSON.parse(localStorage.getItem("loginUser"));
    const token = loginUserInStorage?.token;
    if (token) { config.headers['Authorization'] = token; }
    return config;
  },
  (error) => {
    return Promise.reject(error); //エラーを次の処理に渡す
  }
);
```

練習問題

- 練習09-4
- 練習09-5

トークンの確認

- React内でログインが必要なページにアクセスする場合、トークンの有無を確認するだけでなく、そのトークンが有効なものかをSpring Boot側で確認する



トークン確認用メソッドの実装例

- トークンの存在と有効性をチェックする

Spring

```
@GetMapping("/api/verify-token")
public ResponseEntity<String> verify(HttpServletRequest request) {
    // トークンの取得
    String token = request.getHeader("Authorization");
    // トークンが存在しない、またはトークンが不正
    if (token == null ||
        request.getSession().getAttribute(token) == null) {
        return new ResponseEntity<>("token is invalid",
                                    HttpStatus.UNAUTHORIZED);
    }

    return new ResponseEntity<>("token is valid", HttpStatus.OK);
}
```

React内でのアクセス制御例

- React内でアクセス制御が必要なページ(コンポーネント)では、保持しているトークンなどの情報を確認する
 - 必要な情報を有していない場合は、リダイレクト処理などの対応をする

アクセス制御が必要なページのコンポーネントの記述例

```
// ローカルストレージ内のトークンを取得
const loginUser = JSON.parse(localStorage.getItem("loginUser"));

if (!loginUser || !loginUser.token) {
  console.log("トークンが存在しません");
  return <Navigate to="/login" replace />;
}

...
```

React

React内でのアクセス制御例

前頁の続き

...

React

```
// チェック用エンドポイントにトークンを送信
const url = "http://localhost:8080/api/verify-token";
const response = await axios.get(url, {
  withCredentials: true,
  headers: { "Authorization": loginUser.token }
});

// トークンが不正な場合
if(response.status !== 200) {
  return <Navigate to="/login" replace />;
}
```

練習問題

- 練習09-6