

# 1.5 PersonProject 中級

- ログイン画面の設定
- LoginControllerクラスの設定
- BlogDaoの設定
- BlogServiceクラスの設定
- AdminBlogControllerクラスの設定
- ブロガー一覧画面の作成

# 目次

- 1 ログイン画面の設定
- 2 LoginControllerクラスの設定
- 3 BlogDaoの設定
- 4 BlogServiceクラスの設定
- 5 AdminBlogControllerクラス  
の設定
- 6 ブロガー一覧画面の作成

# ログイン画面の作成①

- admin\_login.htmlを作成し、以下のコードを書き写してください。

```
admin_login.html ×
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8   <title>Document</title>
9   <link rel="stylesheet" th:href="@{/css/reset.css}">
10  <link rel="stylesheet" th:href="@{/css/login.css}">
11 </head>
12
13 <body>
14   <header class="pc">
15     <nav class="pc-nav">
16       <!--<div class="pc-nav-left">
17         <a href=""></a>
18       </div>
19       <div class="pc-nav-right">
20         <div><a href=""></a></div>
21         <div><a href=""></a></div>
22       </div>-->
23     </nav>
24   </header>
25   <header class="sp">
26     <nav class="sp-nav">
27       <!-- <div class="sp-nav-left">
28         <a href=""></a>
29       </div> -->
30     </nav>
31   </header>
```



# ログイン画面の作成②

- admin\_login.htmlを作成し、以下のコードを書き写してください。

```
32<main>
33  <section>
34    <h2>ブログ管理システム</h2>
35    <form th:action="@{/admin/login}" method="POST">
36      <table>
37        <tr>
38          <th>email</th>
39          <td><input type="text" name="userEmail"></td>
40        </tr>
41        <tr>
42          <th>password</th>
43          <td class="login-pass"><input type="text" name="password"></td>
44        </tr>
45      </table>
46      <div class="submit-button">
47        <button id="login" class="active_btn">ログイン</button>
48      </div>
49    </form>
50  </section>
51</main>
52<script src="https://code.jquery.com/jquery-3.6.0.min.js" integrity="sha256-/xUj+30JU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4=" crossorigin="anonymous"></script>
53</body>
54
55</html>
```

# ログイン画面の作成③

## ● ソースの解説

WebSecurityConfigクラス設定した内容に合わせるようにする。

```
<section>
  <h2>ブログ管理システム</h2>
  <form th:action="@{/admin/login}" method="POST">
    <table>
      <tr>
        <th>email</th>
        <td><input type="text" name="userEmail"></td>
      </tr>
      <tr>
        <th>password</th>
        <td class="login-pass"><input type="text" name="password"></td>
      </tr>
    </table>
    <div class="submit-button">
      <button id="login" class="active_btn">ログイン</button>
    </div>
  </form>
</section>
```

```
public class WebSecurityConfig {
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity
        http.formLogin(login -> login
            .loginPage("/admin/login")
            .defaultSuccessUrl("/admin/blog/all", true)
            .usernameParameter("userEmail") // リクエストパラメータ
            .passwordParameter("password")
            .failureUrl("/admin/login?error")
            .permitAll()
```

Thymeleafの書き方は、以下のURLを見て書き方を覚えてください。  
[https://qiita.com/oh\\_yeah\\_sayryo/items/913646e31bd2064ba5c9](https://qiita.com/oh_yeah_sayryo/items/913646e31bd2064ba5c9)

# ログイン画面の作成④

- cssフォルダにlogin.cssを作成し、以下の内容を書き写してください。

```
login.css x
1 |
2
3 /*-----
4     スマートフォン(ヘッダー)
5     -----*/
6 .pc{
7     display: none;
8 }
9 .sp{
10     background-color: #71c7c8 ;
11     padding:1rem;
12 }
13
14 /*-----
15     メイン
16     -----*/
17 section{
18     width:90%;
19     margin: 5rem auto;
20 }
21
22 section h2{
23     font-size:1.5rem;
24     font-weight:bold;
25     text-align: center;
26     color: #2db3b5 ;
27     line-height: 2rem;
28 }
29
30 table{
31     margin:1rem auto;
32 }
33 th{
34     font-size: 1rem;
35     text-align:left;
36     padding:0.6rem;
37 }
38 td{
39     padding:1rem;
40     font-size: 1rem;
41     text-align:right;
42     box-sizing: border-box;
43 }
44 input{
45     height:1.3rem;
46 }
47 .login-pass{
48     position: relative;
49 }
```

続き

```
50 .login-pass img{
51     width:16px;
52     position: absolute;
53     right:20px;
54     top:20px;
55 }
56 .submit-button{
57     text-align: center;
58     margin:1.5rem auto;
59 }
60
61 .btn_disable{
62     background-color: #c2c5c5 ;
63     color: #fff;
64     padding: 0.5rem 1rem;
65     border: 1px solid #798081 ;
66     font-weight:bold;
67 }
68 .active_btn{
69     background-color: #71c7c8 ;
70     color: #fff;
71     padding: 0.5rem 1rem;
72     border: 1px solid #53c0c2 ;
73     font-weight:bold;
74 }
75
76 @media screen and (min-width: 768px) {
77     /*-----
78         PC(ヘッダー)
79         -----*/
80     .sp{
81         display: none;
82     }
83     .pc{
84         display:block;
85         background-color: #b2dbdc ;
86         padding:1rem;
87     }
88     .pc-nav{
89         display: flex;
90         justify-content:space-between;
91         padding:0rem 3rem;
92 }
```

# ログイン画面の作成⑤

- cssフォルダにlogin.cssを作成し、以下の内容を書き写してください。

```

93 .pc-nav-right{
94     display: flex;
95     vertical-align: middle;
96     justify-content:center;
97     align-items:center;
98     gap: 0 1.5rem;
99 }
100
101 /*-----
102             メイン
103 -----*/
104 section h2{
105     font-size:2rem;
106     margin-bottom:1.5rem;
107 }
108
109 h2 br{
110     display:none;
111 }
112
113 th{
114     font-size: 1.3rem;
115     text-align:left;
116     padding:1rem;
117 }
118 td{
119     font-size: 1.3rem;
120 }
121 input{
122     height:1.5rem;
123     width:15rem;
124 }
125
126 .login-pass img{
127     right:25px;
128     top:22px;
129 }
    
```

続き

```

130 .submit-button{
131     text-align: center;
132     margin:1.5rem auto;
133 }
134
135 .btn_disable{
136
137     padding: 0.8rem 2rem;
138 }
139
140 .active_btn{
141     padding: 0.8rem 2rem;
142 }
143 }
    
```

# 目次

- 1 ログイン画面の設定
- 2 **LoginControllerクラスの設定**
- 3 BlogDaoの設定
- 4 BlogServiceクラスの設定
- 5 AdminBlogControllerクラス  
の設定
- 6 ブロッガー一覧画面の作成



# LoginControllerの作成①

- LoginControllerを作成し、以下のコードを書き写してください。

```

LoginController.java ×
1 package blog.example.controller;
2
3 import org.springframework.stereotype.Controller;
4
5
6
7 @Controller
8 @RequestMapping("/admin")
9 public class LoginController {
10
11     //ログイン画面を表示
12     @GetMapping("/login")
13     public String getLoginPage() {
14         return "admin_login.html";
15     }
16
17
18 }

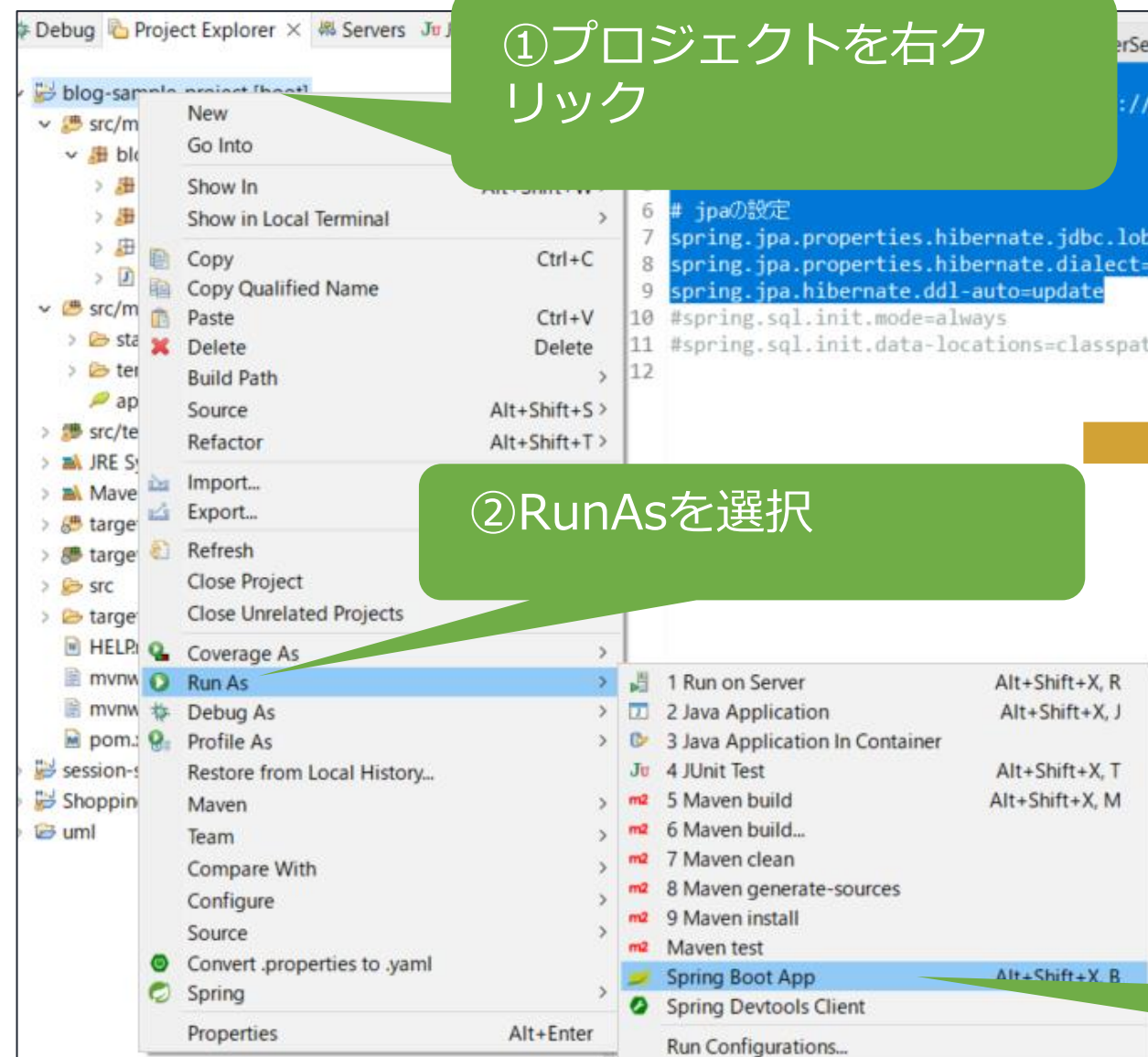
```

URLが「/admin/～」となるように設定

ログイン画面を表示させる  
admin\_login.htmlを表示させる

# 動作確認①

- 最後は、登録できるかどうかの確認をしていきます。



## 動作確認②

- ログイン画面が表示できるかの確認をしていきます。

Document x +

localhost:8080/admin/login

### ブログ管理システム

email

password

ログイン

①ブラウザの検索欄に以下のURLを入力Enter  
`http://localhost:8080/admin/login`

# 動作確認③

- 管理者登録後、ログイン画面にリダイレクトされるか確認

ブラウザの検索欄に以下のURLを入力Enter  
http://localhost:8080/admin/register

Document x +

localhost:8080/admin/register

ログイン

管理者登録画面

UserName 大久保太郎

UserEmail otest@test.com

Password password123

登録 戻る

②3項目を入力し、「登録」ボタンを押下

登録

Document x +

localhost:8080/admin/login

ブログ管理システム

email

password

ログイン

③ログイン画面にリダイレクトされることを確認。

	user_id [PK] bigint	user_name character varying (255)	user_email character varying (255)	password character varying (255)
1	1	前沢昭	matest@test.com	password123
2	2	大久保太郎	otest@test.com	password123

④保存ができているかの確認



# 目次

- 1 ログイン画面の設定
- 2 LoginControllerクラスの設定
- 3 BlogDaoの設定
- 4 BlogServiceクラスの設定
- 5 AdminBlogControllerクラス  
の設定
- 6 ブロガー一覧画面の作成

# BlogDaoインターフェースの作成①

- BlogDaoクラスの作成方法を説明します。

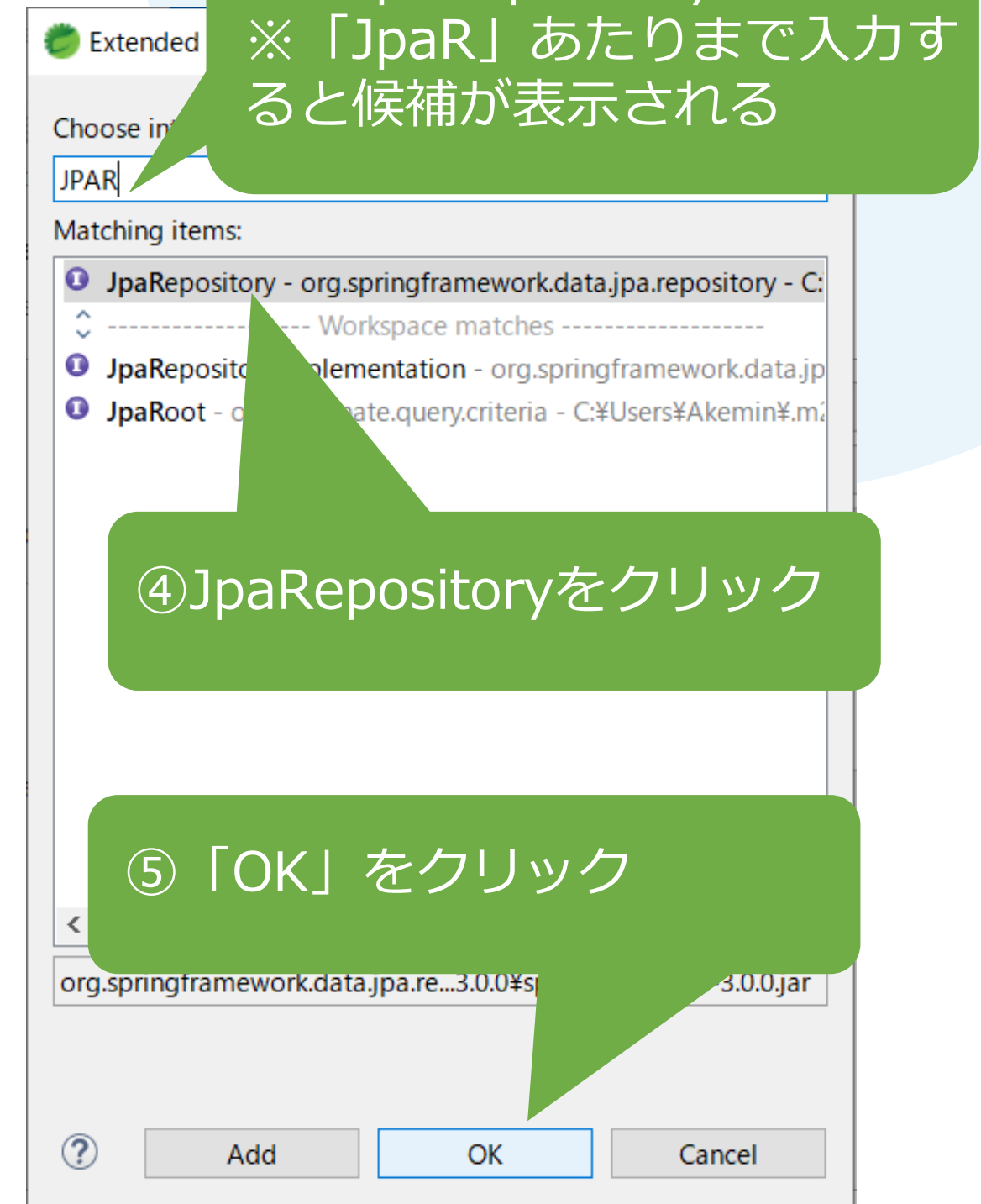
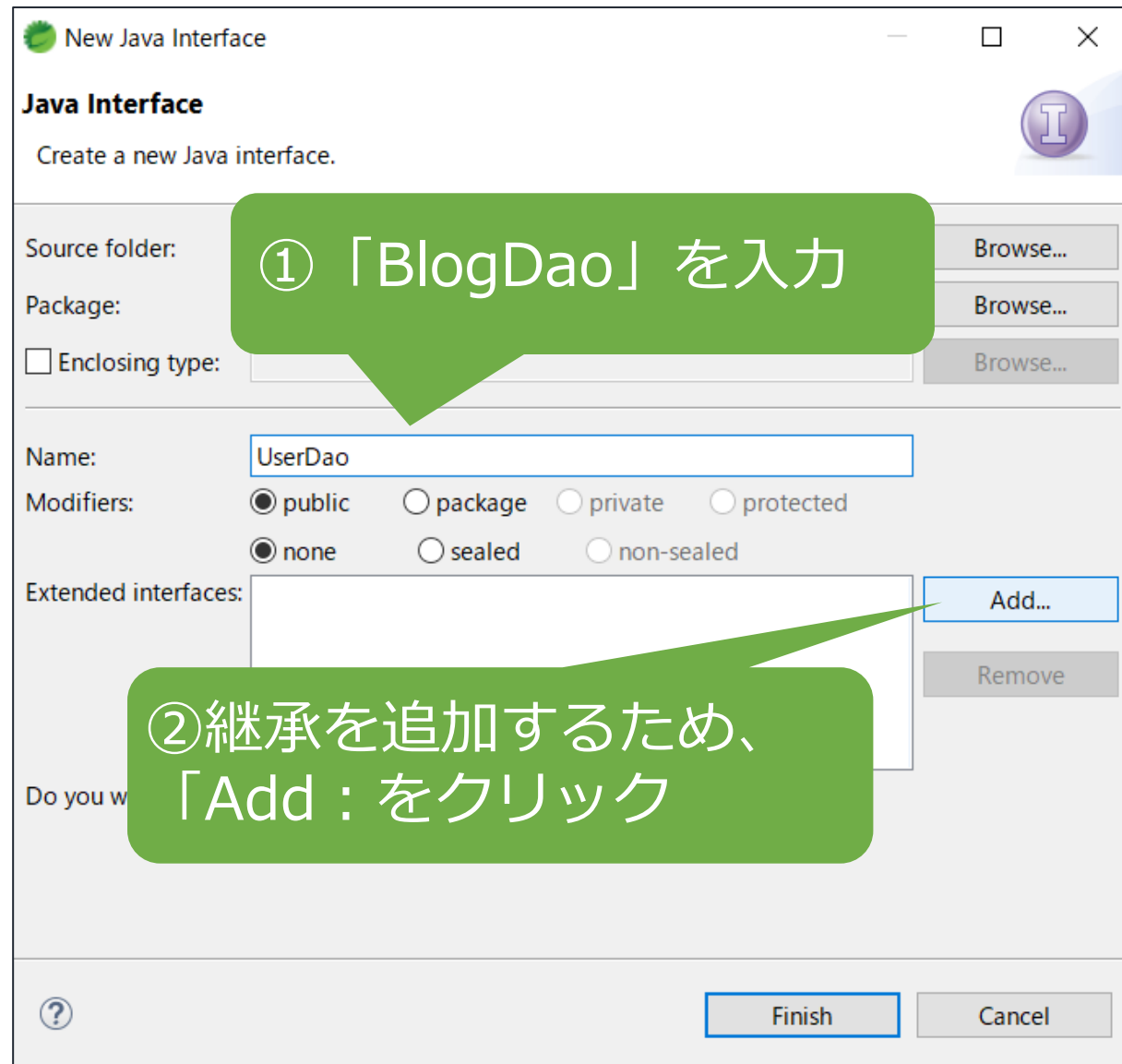
①右クリック

②Newを選択

③interfaceを押下

# BlogDaoインターフェースの作成②

- BlogDaoクラスの作成方法を説明します。



# BlogDaoインターフェースの作成③

- BlogDaoクラスの作成方法を説明します。

**New Java Interface**

**Java Interface**  
Create a new Java interface.

☐ Enclosing type:

Name: UserDao

Modifiers: ☒ public ☐ package ☐ private ☒ none ☐ sealed ☐ non-sealed

Extended interfaces: ☒ org.springframework.data.jpa.repository.JpaRepository...

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

**Finish** **Cancel**

① 「BlogDao」が入力されているかを確認

② JpaRepositoryが追加されていることを確認

③ 「Finish」をクリック



# BlogDaoインターフェースの作成④

- 以下の内容を書き写してください。

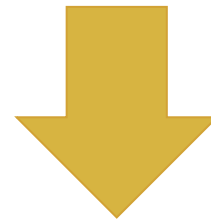
```

BlogDao.java ×
1 package blog.example.model.dao;
2
3 import java.util.List;
4
5
6
7 import org.springframework.data.jpa.repository.JpaRepository;
8 import org.springframework.data.jpa.repository.Query;
9 import org.springframework.stereotype.Repository;
10
11 import blog.example.model.entity.BlogEntity;
12 import jakarta.transaction.Transactional;
13
14
15 @Repository
16 public interface BlogDao extends JpaRepository<BlogEntity, Long> {
17     //ブログの内容を保存
18     BlogEntity save(BlogEntity blogEntity);
19     //ブログテーブルのuser_idとアカウントテーブルのuserIdを使ってテーブルを結合させてuserIdで検索をかけてデータを取得
20     @Query(value="SELECT b.blog_id,b.blog_title,b.blog_image,b.category_name,"
21         + "b.message,b.user_id From blog b "
22         + "INNER JOIN account a ON b.user_id = a.user_id"
23         + " WHERE b.user_id=?1",nativeQuery = true)
24     List<BlogEntity> findByUserId(Long userId);
25
26     //blogIdを使用してDBに検索をかける
27     BlogEntity findById(Long blogId);
28     //ブログテーブルのすべての情報を取得
29     List<BlogEntity> findAll();
30     //カテゴリー名を使用して、DBに検索をかける
31     List<BlogEntity> findByCategoryName(String categoryName);
32
33     //blogIdを取得して該当するブログ情報を削除する
34     @Transactional
35     List<BlogEntity> deleteByBlogId(Long blogId);
36 }
    
```

# BlogDaoインターフェースの作成⑤

- ソースの説明

```
//ブログの内容を保存  
BlogEntity save(BlogEntity blogEntity);
```



saveメソッドは要注意  
insertとupdateの両方の役割を持っている  
1：更新対象のキーでセレクト  
2：同じキーのレコードがあるかをチェック  
    1：あったらupdate  
    2：なかったらinsert  
という順で処理をしている。

# BlogDaoインターフェースの作成⑥

## ● ソースの説明。

ログインしている人の記事一覧取得

```

19 //ブログテーブルのuser_idとアカウントテーブルのuserIdを使ってテーブルを結合させてuserIdで検索をかけてデータを取得
20 @Query(value="SELECT b.blog_id,b.blog_title,b.blog_image,b.category_name,"
21         + "b.message,b.user_id From blog b "
22         + "INNER JOIN account a ON b.user_id = a.user_id"
23         + " WHERE b.user_id=?1",nativeQuery = true)
24 List<BlogEntity>findByUserId(Long userId);

```

Blogテーブルのデータ  
※テストデータを挿入している

	blog_id [PK] bigint	blog_image character varying (255)	blog_title character varying (255)	category_name character varying (255)	message character varying (255)	user_id bigint
1	2	x05.jpg	冬の鍋	もつ鍋	もつ鍋最高です！	1
2	102	x06.jpg	回転ずし	お寿司	久しぶりに回転寿司に...	2
3	152	x02.jpg	食べ放題	焼肉	焼肉食べ放題！カルビ...	1
4	153	x06.jpg	寿司の季節	お寿司	回らないお寿司 めちゃ...	1

accountのuser\_idとblogの  
user\_idで結合

SELECT b.blog\_id,b.blog\_title,b.blog\_image,b.category\_name,b.message,b.user\_id From blog b  
INNER JOIN account a ON b.user\_id = a.user\_id

結合結果


	blog_id [PK] bigint	blog_title character varying (255)	blog_image character varying (255)	category_name character varying (255)	message character varying (255)	user_id bigint
1	153	寿司の季節	x06.jpg	お寿司	回らないお寿司 めちゃ...	1
2	152	食べ放題	x02.jpg	焼肉	焼肉食べ放題！カルビ...	1
3	2	冬の鍋	x05.jpg	もつ鍋	もつ鍋最高です！	1
4	102	回転ずし	x06.jpg	お寿司	久しぶりに回転寿司に...	2

# BlogDaoインターフェースの作成⑦

## ● ソースの説明（先ほどの続き）。

```
SELECT b.blog_id,b.blog_title,b.blog_image,b.category_name,b.message,b.user_id From blog b
INNER JOIN account a ON b.user_id = a.user_id
```

結合結果ここからログインしている人のuser\_idを使って検索する必要がある




	blog_id [PK] bigint	blog_title character varying (255)	blog_image character varying (255)	category_name character varying (255)	message character varying (255)	user_id bigint
1	153	寿司の季節	x06.jpg	お寿司	回らないお寿司 めちゃ...	1
2	152	食べ放題	x02.jpg	焼肉	焼肉食べ放題！カルビ...	1
3	2	冬の鍋	x05.jpg	もつ鍋	もつ鍋最高です！	1
4	102	回転ずし	x06.jpg	お寿司	久しぶりに回転寿司に...	2

結合後にuser\_idで検索

```
SELECT b.blog_id,b.blog_title,b.blog_image,b.category_name,b.message,b.user_id From blog b
INNER JOIN account a ON b.user_id = a.user_id WHERE b.user_id=1
```

結合後結果



	blog_id [PK] bigint	blog_title character varying (255)	blog_image character varying (255)	category_name character varying (255)	message character varying (255)	user_id bigint
1	2	冬の鍋	x05.jpg	もつ鍋	もつ鍋最高です！	1
2	152	食べ放題	x02.jpg	焼肉	焼肉食べ放題！カルビ最高！	1
3	153	寿司の季節	x06.jpg	お寿司	回らないお寿司 めちゃくちゃ値段が高い(;w; `)	1



# BlogDaoインターフェースの作成⑧

- ソースの説明。

ブログ記事の編集の際に使用

```
//blogIdを使用してDBに検索をかける  
BlogEntity findByBlogId(Long blogId);
```

➡ SELECT blog\_id, blog\_title, blog\_image, category\_name, message, user\_id FROM  
blog WHERE blog\_id = ?1

```
//ブログテーブルのすべての情報を取得  
List<BlogEntity> findAll();
```

ユーザー側のブログ記事一覧取得の際に使用

➡ SELECT \* FROM blog

ユーザー側のカテゴリ検索の際に使用

```
//カテゴリ名を使用して、DBに検索をかける  
List<BlogEntity> findByCategoryName(String categoryName);
```

➡ SELECT blog\_id, blog\_title, blog\_image, category\_name, message, user\_id FROM blog  
WHERE category\_name=?1

# BlogDaoインターフェースの作成⑨

- ソースの説明。

```
//blogIdを取得して該当するブログ情報を削除する
@Transactional
List<BlogEntity> deleteByBlogId(Long blogId);
```

ブログ記事の削除の際に使用

delete from blog where blog\_id=?1



@Transactionalをつけないとエラーが発生

No EntityManager with actual transaction available for current thread - cannot reliably process 'remove' call

現在のスレッドで使用可能な実際のトランザクションを持つEntityManagerがありません-「削除」呼び出しを確実に処理できません



@TransactionalはDBトランザクション処理を行うアノテーション  
例外発生時にロールバックをしてくれるため、心強いが、  
JPAがdeleteメソッドを自動生成する場合に、@Transactionalがないと  
弾くようになっている。

# 目次

- 1 ログイン画面の設定
- 2 LoginControllerクラスの設定
- 3 BlogDaoの設定
- 4 BlogServiceクラスの設定
- 5 AdminBlogControllerクラス  
の設定
- 6 ブロガー一覧画面の作成

# BlogServiceクラスの作成①

- BlogServiceクラスを作成し、下記のソースを書き写してください。

```
BlogService.java ×
1 package blog.example.service;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7 import blog.example.model.dao.BlogDao;
8 import blog.example.model.entity.BlogEntity;
9
10
11 @Service
12 public class BlogService {
13     @Autowired
14     BlogDao blogDao;
15
16     //内容を保存
17     public void insert(String blogTitle,String fileName,String categoryName,String message,Long userId) {
18         blogDao.save(new BlogEntity(blogTitle,fileName,categoryName,message,userId));
19     }
20     //ブローガー一覧
21     public List<BlogEntity> selectByUserId(Long userId){
22         return blogDao.findByUserId(userId);
23     }
24 }
```



# BlogServiceクラスの作成②

- BlogServiceクラスを作成し、下記のソースを書き写してください。

```
24
25 //ブログ詳細
26 public BlogEntity selectByBlogId(Long blogId){
27     return blogDao.findByBlogId(blogId);
28 }
29 //内容を更新
30 public void update(Long blogId,String blogTitle,String fileName,String categoryName,String message,Long userId) {
31     blogDao.save(new BlogEntity(blogId,blogTitle,fileName,categoryName,message,userId));
32 }
33
34 //ユーザープロガー覧
35 public List<BlogEntity> selectByAll(){
36     return blogDao.findAll();
37 }
38
39 //カテゴリー—事の内容
40 public List<BlogEntity> selectByCategoryName(String categoryName){
41     return blogDao.findByCategoryName(categoryName);
42 }
43 //削除
44 public List<BlogEntity> deleteBlog(Long blogId){
45     return blogDao.deleteByBlogId(blogId);
46 }
47
48 }
```

# BlogServiceクラスの作成③

- BlogServiceクラスのソース全体（書き漏れがないかを確認して下さい）。

```
BlogService.java ×
1 package blog.example.service;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7 import blog.example.model.dao.BlogDao;
8 import blog.example.model.entity.BlogEntity;
9
10
11 @Service
12 public class BlogService {
13     @Autowired
14     BlogDao blogDao;
15
16     //内容を保存
17     public void insert(String blogTitle,String fileName,String categoryName,String message,Long userId) {
18         blogDao.save(new BlogEntity(blogTitle,fileName,categoryName,message,userId));
19     }
20     //ブローガー一覧
21     public List<BlogEntity> selectByUserId(Long userId){
22         return blogDao.findByUserId(userId);
23     }
24
25     //ブログ詳細
26     public BlogEntity selectByBlogId(Long blogId){
27         return blogDao.findById(blogId);
28     }
29     //内容を更新
30     public void update(Long blogId,String blogTitle,String fileName,String categoryName,String message,Long userId) {
31         blogDao.save(new BlogEntity(blogId,blogTitle,fileName,categoryName,message,userId));
32     }
33
34     //ユーザーブローガー一覧
35     public List<BlogEntity> selectByAll(){
36         return blogDao.findAll();
37     }
38
39     //カテゴリー一事の内容
40     public List<BlogEntity> selectByCategoryName(String categoryName){
41         return blogDao.findByCategoryName(categoryName);
42     }
43     //削除
44     public List<BlogEntity> deleteBlog(Long blogId){
45         return blogDao.deleteByBlogId(blogId);
46     }
47
48 }
```

# BlogServiceクラスの作成④

- ソースの解説とBlogEntity追加模写

BlogDaoのメソッドを使用できるように設定

BlogControllerから値を受け取る

```

11 @Service
12 public class BlogService {
13     @Autowired
14     BlogDao blogDao;
15
16     //内容を保存
17     public void insert(String blogTitle,String fileName,String categoryName,String message,Long userId) {
18         blogDao.save(new BlogEntity(blogTitle,fileName,categoryName,message,userId));
19     }

```

保存処理をするメソッドをに引数をセットし、BlogDaoに渡して保存処理を行う。

```

20 @Table(name="blog")
21 public class BlogEntity {
22     public BlogEntity(String blogTitle, String fileName, String categoryName, String message, Long userId) {
23         this.blogTitle = blogTitle;
24         this.blogImage = fileName;
25         this.categoryName = categoryName;
26         this.message = message;
27         this.userId = userId;
28     }

```

saveメソッドでBlogEntityにセットした内容のコンストラクタは存在しないため、BlogEntityに追加のコンストラクタを記載する

# BlogServiceクラスの作成⑤

## ● ソースの解説。

```
//ブログ一覧
public List<BlogEntity> selectByUserId(Long userId){
    return blogDao.findByUserId(userId);
}

//ブログ詳細
public BlogEntity selectByBlogId(Long blogId){
    return blogDao.findByBlogId(blogId);
}

//内容を更新
public void update(Long blogId,String blogTitle,String fileName,String categoryName,String message,Long userId) {
    blogDao.save(new BlogEntity(blogId,blogTitle,fileName,categoryName,message,userId));
}

//ユーザーブログ一覧
public List<BlogEntity> selectByAll(){
    return blogDao.findAll();
}

//カテゴリ——事の内容
public List<BlogEntity> selectByCategoryName(String categoryName){
    return blogDao.findByCategoryName(categoryName);
}

//削除
public List<BlogEntity> deleteBlog(Long blogId){
    return blogDao.deleteByBlogId(blogId);
}
```

管理者側のブログ記事一覧取得の際に使用  
BlogControllerからuserIdを受け取り、BlogDao  
のfindByUserIdメソッドを使用して検索かける

管理者側のブログ編集画面とユーザー側のブログ  
記事詳細画面のデータ取得の際に使用

ブログ編集内容を更新する際に使用

ユーザー側のブログ記事一覧取得の際に使用

ユーザー側のカテゴリ検索の際に使用

ブログ記事の削除に使用



# 目次

- 1 ログイン画面の設定
- 2 LoginControllerクラスの設定
- 3 BlogDaoの設定
- 4 BlogServiceクラスの設定
- 5 AdminBlogController  
クラスの設定
- 6 ブロッガー一覧画面の作成

# AdminBlogControllerクラスの作成①

- AdminBlogControllerクラスを作成し、下記のソースを書き写してください。ソースの解説はコメントをみて理解を進めてください。

```
*AdminBlogController.java ×
32
33 @Controller
34 @RequestMapping("/admin/blog")
35 public class AdminBlogController {
36     /**
37      * accountテーブルを操作するための
38      * Serviceクラス
39      */
40     @Autowired
41     private UserService userService;
42     /**
43      * blogテーブルを操作するための
44      * Serviceクラス
45      */
46     @Autowired
47     BlogService blogService;
48
49     /**
50     * 管理者側の処理
51     */
52
53     /**
54     * 現在ログインしている人が記載した記事の一覧を出すための処理です。
55     * -ログインしている人のメールアドレスを使用して、ログインしている人のuserIdとuserNameを取得します。
56     * -取得したuserIdを使用してログインしている人のブログ記事を取得します。
57     * -取得した情報をセットしてが円から参照可能にします。
58     */
59 }
```

# AdminBlogControllerクラスの作成②

- AdminBlogControllerクラスを作成し、下記のソースを書き写してください。ソースの解説はコメントをみて理解を進めてください。

```

54  /**
55   * 現在ログインしている人が記載した記事の一覧を出すための処理です。
56   * -ログインしている人のメールアドレスを使用して、ログインしている人のuserIdとuserNameを取得します。
57   * -取得したuserIdを使用してログインしている人のブログ記事を取得します。
58   * -取得した情報をセットしてが円から参照可能にします。
59   */
60  //管理者側のブログ一覧を表示
61  @GetMapping("/all")
62  public String getLoginPage(Model model) {
63      // 現在のリクエストに紐づく Authentication を取得するには SecurityContextHolder.getContext().getAuthentication() とする。
64      // SecurityContextHolder.getContext() は、現在のリクエストに紐づく SecurityContext を返している。
65      // Authentication.getAuthorities() で、現在のログインユーザーに付与されている権限(GrantedAuthority のコレクション)を取得できる。
66      Authentication auth = SecurityContextHolder.getContext().getAuthentication();
67
68      //ログインした人のメールアドレスを取得
69      String userEmail = auth.getName();
70      //accountテーブルの中から、ユーザーのEmailで検索をかけて該当するユーザーのID情報を引っ張り出す。
71      UserEntity user = userService.selectById(userEmail);
72
73      //accountテーブルの中からログインしているユーザーの名前の取得
74      String userName = user.getUserName();
75
76      //accountテーブルの中からログインしているユーザーのIDを取得
77      Long userId = user.getUserId();
78
79      //ブログテーブルの中からユーザーIDを使って、そのユーザーが書いたブログ記事のみを取得する
80      List<BlogEntity> blogList = blogService.selectByUserId(userId);
81      //blogList(ブログの記事一覧情報)とuserName(管理者の名前)をmodelにセットし
82      //admin_blog_all.htmlから参照可能にする。
83      model.addAttribute("blogList", blogList);
84      model.addAttribute("userName", userName);
85      return "admin_blog_all.html";
86  }
87
88
89  }

```

# 目次

- 1 ログイン画面の設定
- 2 LoginControllerクラスの設定
- 3 BlogDaoの設定
- 4 BlogServiceクラスの設定
- 5 AdminBlogControllerクラス  
の設定
- 6 ブロガー一覧画面の作成

# ブロッガー覧画面の作成①

- admin\_blog\_all.htmlを作成し、下記の内容を書き写してください。

```
admin_blog_all.html ×
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <meta name="viewport" content="width=device-width, initial-scale=1.0">
8     <title>Document</title>
9     <link rel="stylesheet" th:href="@{/css/reset.css}">
10    <link rel="stylesheet" th:href="@{/css/style.css}">
11    <link rel="preconnect" href="https://fonts.googleapis.com">
12    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
13    <link href="https://fonts.googleapis.com/css2?family=Kiwi+Maru:wght@300;400;500&display=swap" rel="stylesheet">
14    <link rel="preconnect" href="https://fonts.googleapis.com">
15    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
16    <link href="https://fonts.googleapis.com/css2?family=Gochi+Hand&family=Kiwi+Maru:wght@300;400;500&display=swap">
17 </head>
```



# ブロッカー一覧画面の作成②

- admin\_blog\_all.htmlを作成し、下記の内容を書き写してください。

```

18
19<body>
20  <header>
21    <!--スマートフォン-->
22    <nav class="menu">
23      <div class="logo">
24        <a href="#"></a>
25      </div>
26      <div class="menu-contents">
27        <div class="menu-inner">
28          <ul>
29
30            <li class="menu__item"><a th:href="@{/admin/category/all}">カテゴリ一覧</a></li>
31            <li class="menu__item"><a th:href="@{/admin/blog/all}">投稿記事一覧</a></li>
32            <li class="menu__item">
33              <form th:action="@{/logout}" method="post"><a href="#"
34                onclick="this.parentNode.submit()">ログアウト</a></form>
35            </li>
36
37          </ul>
38        </div>
39        <div class="menu-toggle_btn">
40          <span></span>
41          <span></span>
42          <span></span>
43        </div>
44      </div>
45    </nav>

```

# ブロッガー覧画面の作成③

- admin\_blog\_all.htmlを作成し、下記の内容を書き写してください。

```

46      <!--pc-->
47      <nav class="pc">
48          <div class="pc-inner">
49              <div class="pc-logo">
50                  <a href=""></a>
51              </div>
52              <ul class="pc-list">
53                  <li class="pc-list__item" th:text=${userName}></li>
54                  <li class="pc-list__item"><a th:href="@{/admin/category/all}">カテゴリー一覧</a></li>
55                  <li class="pc-list__item"><a th:href="@{/admin/blog/all}">投稿記事一覧</a></li>
56                  <li class="pc-list__item">
57                      <form th:action="@{/logout}" method="post"><a href="#"
58                          onclick="this.parentNode.submit()">ログアウト</a></form>
59                  </li>
60              </ul>
61          </div>
62      </nav>
63  </header>
64

```

# ブロッガー一覧画面の作成④

- admin\_blog\_all.htmlを作成し、下記の内容を書き写してください。

```

65<main>
66  <div class="main-inner">
67    <section class="all-view-section">
68      <h2>記事一覧</h2>
69      <div class="all-view-register__button">
70        <a th:href="@{/admin/blog/register}" class="btn btn--orange btn--radius">記事登録</a>
71      </div>
72      <div class="all-view__flex">
73
74        <div class="all-view-article" th:each="blog:${blogList}">
75          <a th:href="'/admin/blog/edit/'+${blog.blogId}" class="all-view-colum">
76            <div class="all-view__category">
77              <p th:text="${blog.categoryName}">
78            </p>
79          </div>
80          
81        </a>
82        <div class="colum__box" th:text="${blog.blogTitle}">
83
84      </div>
85    </div>
86
87  </div>
88  </section>
89</div>
90</main>
91<script src="https://code.jquery.com/jquery-3.6.0.min.js"
92  integrity="sha256-/xUj+30JU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4=" crossorigin="anonymous"></script>
93<script th:src="@{/js/common.js}"></script>
94</body>
95
96</html>

```



# ブロッガー一覧画面の作成⑤

## ● ソースの説明

```
//blogList(ブログの記事一覧情報)とuserName(管理者の名前)をmodelにセットし
//admin_blog_all.htmlから参照可能にする。
model.addAttribute("blogList",blogList);
model.addAttribute("userName",userName);
return "admin_blog_all.html";
```

th:each="変数:\${コレクション}"  
変数.フィールド (今回はBlogEntity  
のフィールド) で値を表示できる

コレクションから取り出した値を代入している  
変数をさらに専用の変数itemsに代入する。  
以下の記事と一緒にみておこう  
<https://pointsandlines.jp/server-side/java/image-path-connect>

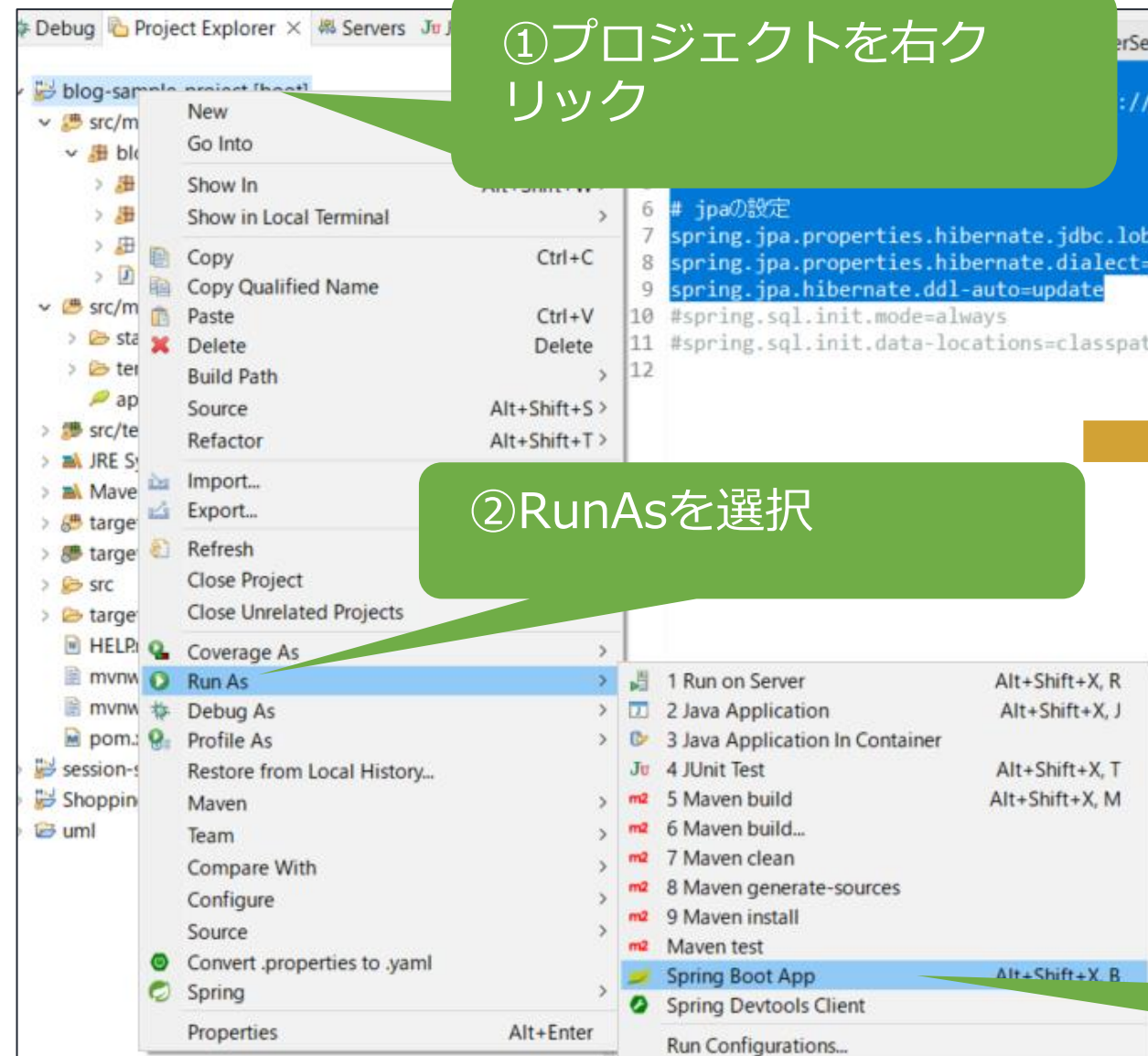
```
<div class="all-view-article" th:each="blog:${blogList}">
  <a th:href="'/admin/blog/edit/'+${blog.blogId}" class="all-view-colum">
    <div class="all-view_category">
      <p th:text=" ${blog.categoryName}">
    </p>
    </div>
    
  </a>
  <div class="colum_box" th:text=" ${blog.blogTitle}">
</div>
</div>
```

BlogEntityのフィールド

```
private Long blogId;
private String blogTitle;
private String blogImage;
private String message;
private String categoryName;
```

# 動作確認

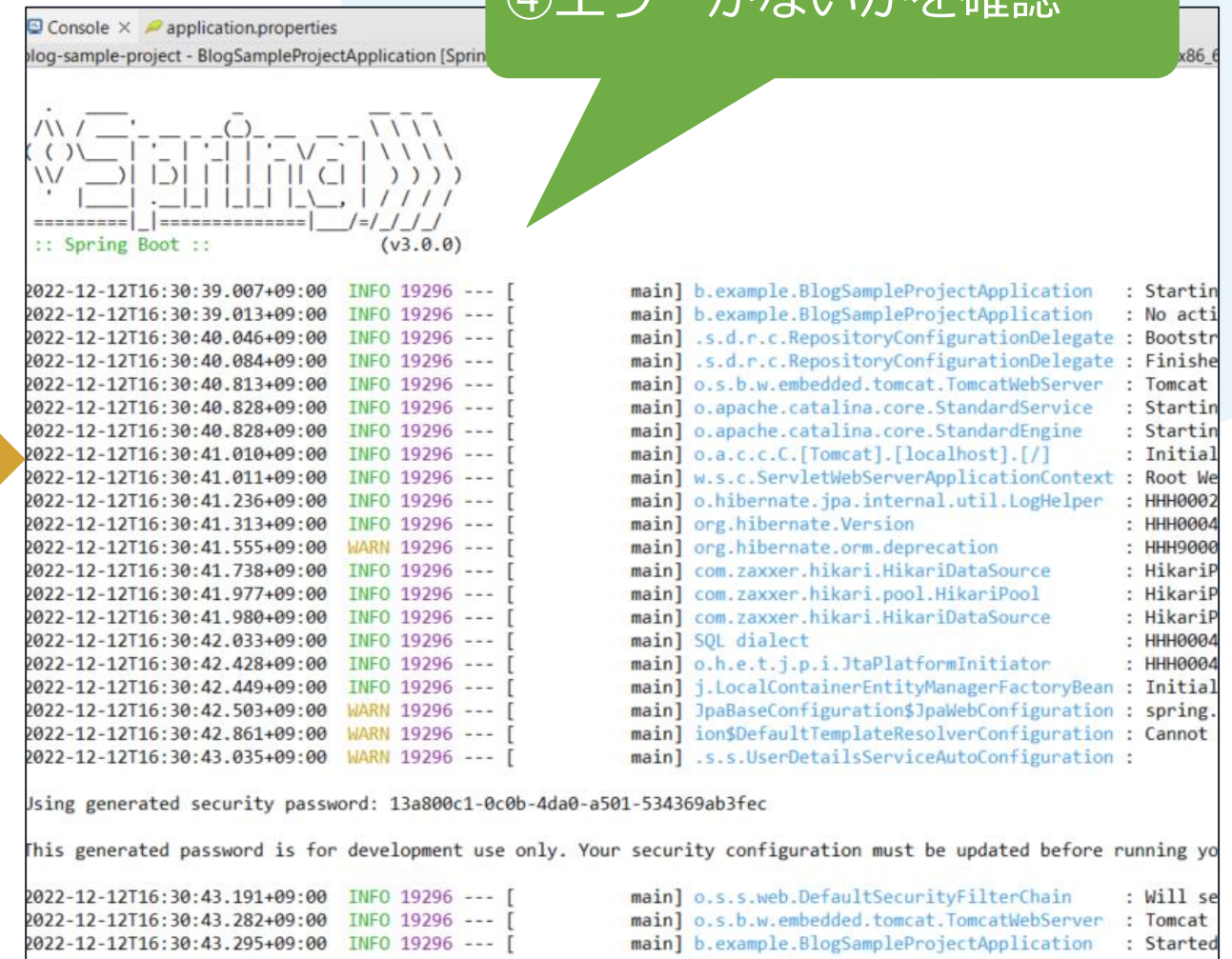
- 最後は、登録できるかどうかの確認をしていきます。



①プロジェクトを右クリック

②RunAsを選択

③SpringBootApplicationをクリック



④エラーがないかを確認



# 動作確認

- ログイン画面してブロッグ一覧画面が見れるかを確認

①ブラウザの検索欄に以下のURLを入力Enter

http://localhost:8080/admin/login

Document x +

localhost:8080/admin/login

## ブログ管理システム

email

password

ログイン

②SBに登録した内容を使ってログイン

Document x +

localhost:8080/admin/blog/all

前沢昭 カテゴリー一覧 投稿記事一覧 ログアウト

## 記事一覧

記事登録

③管理者ブログ記事一覧に遷移できていれば成功  
headerの名前もログインしている人と一致しているかを確認

	user_id [PK] bigint	user_name character varying (255)	user_email character varying (255)	password character varying (255)
1	1	前沢昭	matest@test.com	password123
2	2	大久保太郎	otest@test.com	password123



*Light in Your Career.*

**THANK YOU!**