1.4 PersonProject 初級

- LightHouse Academy
- UserDaoクラスの設定
- UserServiceクラスの設定
- RegisterControllerの設定
- WebConfigの設定
- 登録画面の作成

≈ H ∧

www.lighthouseit.academy

- 1 UserDaoの設定
- 2 UserServiceクラスの設定
- 3 WebSecurityConfigの設定
- 4 RegisterControllerの設定
- 5 登録画面の作成

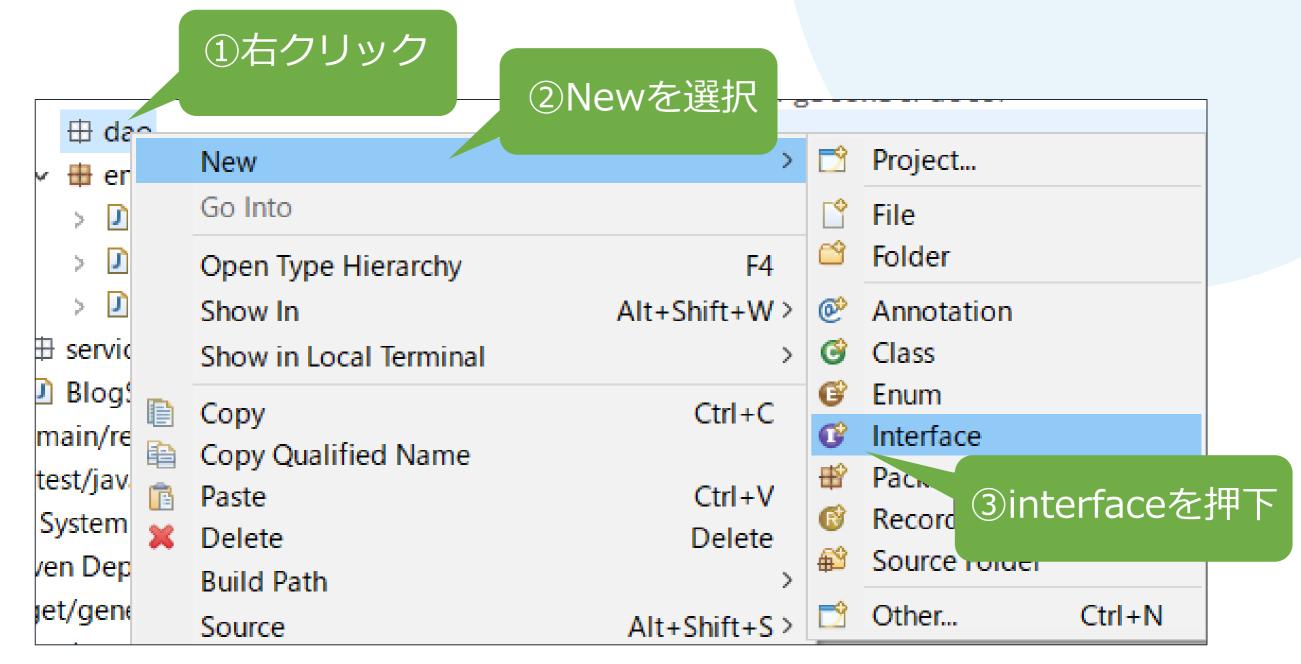
# 目 次





### UserDaoインターフェースの作成①

● UserDaoクラスの作成方法を説明します。

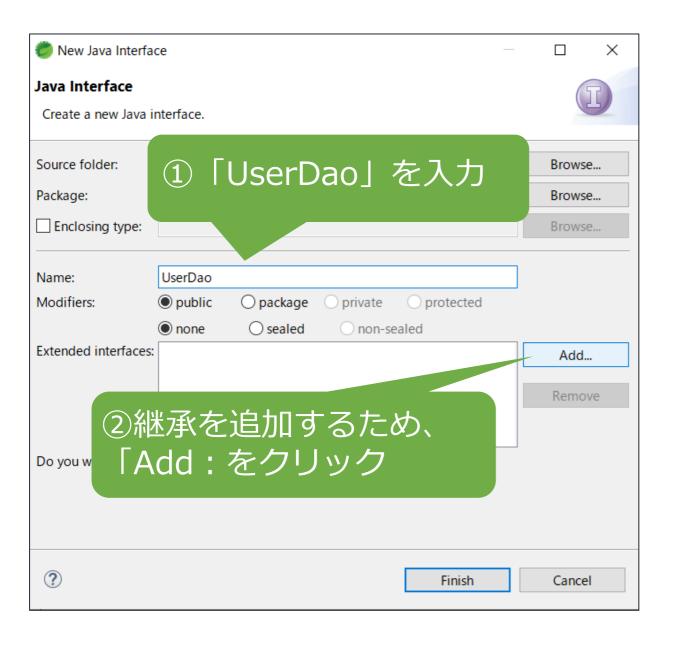


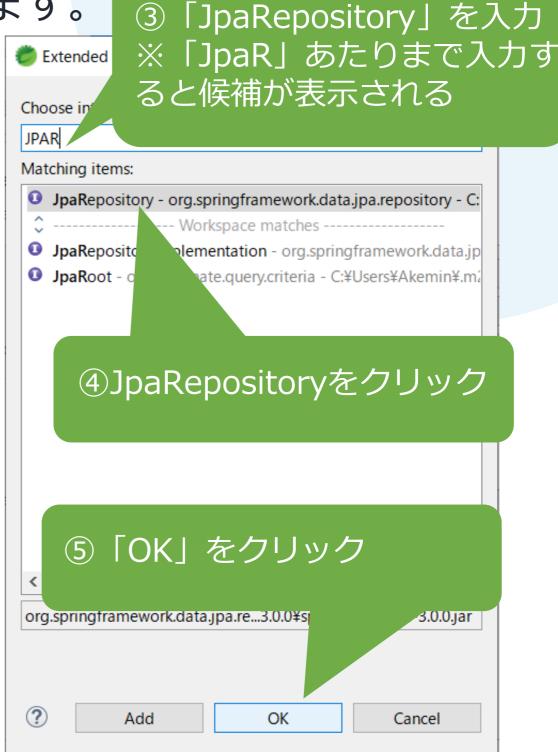




### UserDaoインターフェースの作成②

● UserDaoクラスの作成方法を説明します。



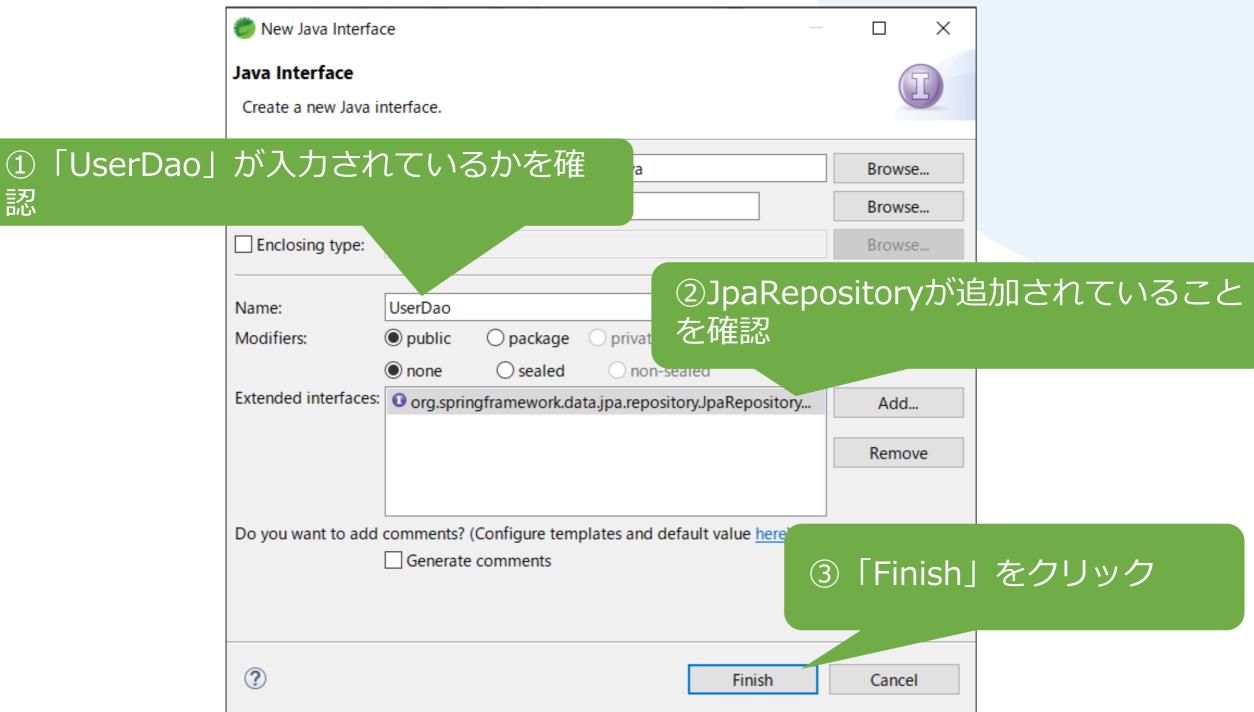






### UserDaoインターフェースの作成③

● UserDaoクラスの作成方法を説明します。







#### UserDaoインターフェースの作成4

● UserDaoクラスの内容を説明します。

```
UserDao.java ×

1 package blog.example.model.dao;
2 import org.springframework.data.jpa.repository.JpaRepository;
4 public interface UserDao extends JpaRepository<I, ID> {
6 7 }
8 Entityにアクセスするため
ものEntity名を記載する
```

Entityにアクセスするため ものなので、Entityに記載 したIDの型を記載する





#### UserDaoインターフェースの作成⑤

● UserDaoインターフェースのコンパイルエラーを修正しま

```
9 UserDao.java ×
1 package blog.example.model.dao;
2 import org.springframework.data.jpa.repository.JpaRepository;
4 public interface UserDao extends JpaRepository<I, ID> {
6 7 }
2 この部分のコンパイルエラーの修正
```





#### UserDaoインターフェースの作成⑥

● @Repositoryを追加しよう

```
UserDao.java × package blog.example.model.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import blog.example.model.entity.UserEntity;

@Repository

public berface UserDao extends JpaRepository<UserEntity, Long> {
```

@Repositoryは、データベースに関連するメ ソッドを提供するDao層のインターフェース でることをSoringに伝える役割を持ちます。





#### UserDaoインターフェースの作成⑦

メソッドを追加しよう

```
UserDao.java ×
                                                        他にもメソッドがたくさんありま
 package blog.example.model.dao;
                                                         す。下記のリンクを参考に他のJPAメ
3回import java.util.List;
                                                        ソッドも
 import org.springframework.data.jpa.repository.JpaRepository;
                                                        調べてみよう!
 import org.springframework.stereotype.Repository;
                                                         https://b1san-
 import blog.example.model.entity.UserEntity;
                                                        blog.com/post/spring/spring-jpa/
 @Repository
 public interface UserDao extends JpaRepositorv<UserEntity, Long> {
     // UserServiceから渡されるユーザ情報(メールアドレス)を条件にDB検索
    UserEntity findByUserEmail(String userEmail);
     // UserServiceから渡されるユーザ情報を基にDBへ保存する
    UserEntity save(UserEntity userEntity);
     // ユーザ情報一覧を取得
     List<UserEntity> findAll();
```





## UserDaoインターフェースの作成®

メソッドの解説

select user\_id,password user\_email,user\_name from account where user email=?



上のSQL文を省略している

// UserServiceから渡されるユーザ情報(メールアドレス)を条件にDB検索

UserEntity findByUserEmail(String userEmail);





## UserDaoインターフェースの作成9

メソッドの解説

// UserServiceから渡されるユーザ情報を基にDBへ保存する UserEntity save(UserEntity userEntity);



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。





# UserDaoインターフェースの作成⑩

メソッドの解説

select user\_id,password user\_email,user\_name from account



上のSQL文を省略している

```
// ユーザ情報一覧を取得
List<UserEntity> findAll();
```

以上でUserDaoの設定は終わります。



- 1 UserDaoの設定
- 2 UserServiceクラスの設定
- 3 WebSecurityConfigの設定
- 4 RegisterControllerの設定
- 5 登録画面の作成

# 目 次





#### UserServiceクラスの作成①

```
🛃 UserService.java 🔀
 1 package blog.example.service;
 3⊝ import java.util.List;
 5 import org.springframework.beans.factory.annotation.Autowired;
                                                                  UserServiceクラスを作成後、左側のソース
 6 import org.springframework.stereotype.Service;
                                                                  コードを書き写してください。
 8 import blog.example.model.dao.UserDao;
 9 import blog.example.model.entity.UserEntity;
                                                                  次のページでソースコードを解説します。
10
11 @Service
12 public class UserService {
      @Autowired
13⊖
14
      private UserDao userDao;
15
        //ユーザの情報を保存する
      public boolean createAccount(String userName,String userEmail, String password) {
16⊜
          //RegisterControllerから渡されるユーザ情報(ユーザーメールアドレス)を条件にDB検索で検索する
17
          UserEntity userEntity = userDao.findByUserEmail(userEmail);
18
          //RegisterControllerから渡されるユーザ情報(ユーザ名、パスワード)を条件にDB検索で検索した結果
          //なかった場合には、保存
20
          if (userEntity==null) {
             userDao.save(new UserEntity(userName, userEmail, password));
             WebSecurityConfig.addUser(userEmail, password);
             return true:
          } else {
26
             return false;
27
      //ユーザの一覧を取得する
29
30⊝
      public List<UserEntity> getAllAccounts() {
31
          return userDao.findAll();
32
33
      //idを見つけるために
34
35
      //コントローラーでわたってきたuserEmailを基にしてDBを検索
36⊜
      public UserEntity selectById(String userEmail) {
37
          return userDao.findByUserEmail(userEmail);
38
39
40 }
```





#### UserServiceクラスの作成②

@Serviceアノテーション →SpringにService層のクラスであることを 伝えます。

@Autowiredアノテーション→Springが自動的にインターフェースを実装して、インスタンス化させるようになります。

UserServiceクラスは、accountテーブルにアクセスする必要がある DBとやり取りをするDaoをメンバ変数として 宣言し、Daoのメソッドを使用してDaoとや り取りをできるようにする





# UserServiceクラスの作成③

User情報の保存が成功する場合の処理のイメージを持ってください。

#### RegisterController



#### UserService



UserDao







①ユーザーから名前と パスワード、Emailの 保存処理の要求がありました。 UserServiceさん作業を お願いします。

②わかりました。 UserDaoさん、Email を使ってすでに登録があるかを はないです。 見てもらってもいいですか?

③わかりました。 探した結果該当レコード

**④ありがとうございま** す。では、UserDaoさん 登録処理をしてもらって もいいですか?

















⑧無事に登録できたのか な!ログインしてみよう!

⑦ありがとうございます 結果がtrueなのでログイン画面 にリダイレクトさせます。

⑥ありがとうございます。! 今回は、正常に保存ができたので、⑤もちろんです! RegisterControllerさん 戻り値はtrueになります。

UserServiceさん登録 処理が終わりましたよ!





#### UserServiceクラスの作成(4)

User情報を保存することができない処理のイメージを持ってください。

#### RegisterController



①ユーザーから名前と パスワード、Emailの UserServiceさん作業を お願いします。

#### UserService



②わかりました。 UserDaoさん、Email 保存処理の要求がありました。を使ってすでに登録があるかを 見てもらってもいいですか?

#### UserDao



③わかりました。 探した結果該当レコード がありました。



**④ありがとうございま** す。保存処理はできない ため戻り値は、falseで す。 RegisterController



に連絡します。





⑥あ!すでに登録してたら から登録できないといわれ てる!そういえば前に登録 したからそりゃそうだ!

⑤ありがとうございます。! 今回は、保存ができてないので、登 録画面に戻して、エラー文が出るよ うに画面にします!





#### UserServiceクラスの作成⑤

● ※WebSecurityConfigは後ほど作成するので、今はエラーのままで大丈夫です。

RegisterControllerからパラメーターを受け 取ります。

```
//ユーザの情報を保存する
public boolean createAccount(String userName,String userEmail, String password) {
   //RegisterControllerから渡されるユーザ情報(ユーザーメールアドレス)を条件にDB検索で検索する
   UserEntity userEntity = userDao.findByUserEmail(userEmail);
   //RegisterController UserDaoのメソッドの引数にuserEmailを渡して検を
   //なかった場合には、保存
                        行い、結果をuserEntityに格納する
   if (userEntity==null) {
      userDao.save(new UserEntity(userName,userEmail, password));
      WebSecurityConfig.addUser(userEmail, password);
                         結果が見つかった場合は、DBに保存処理をかけるのと
      return true;
   } else {
                         同時に、WebSecurityConfigのaddUserで、maneger
      return false;
                          に対してログインできるユーザーを追加する。
}
```





#### UserServiceクラスの作成⑥

```
//ユーザの一覧を取得する
public List<UserEntity> getAllAccounts() {
   return userDao.findAll();
}

//idを見つけるために
//コントローラーでわたってきたuserEmailを基にしてDBを検索
public UserEntity selectById(String userEmail) {
   return userDao.findByUserEmail(userEmail);
}
```

ユーザの一覧を取得してそのデータを 戻り値としている。

ユーザのidを取得するために検索を書けているメソッド。 検索結果が戻りとしてContrllerに渡す。

 ※WebSecurityConfig作成後にimportをするためにもう一度 UserServiceクラスに戻ってきます。



- 1 UserDaoの設定
- 2 UserServiceクラスの設定
- 3 WebSecurityConfigの設定
- 4 RegisterControllerの設定
- 5 登録画面の作成

# 目 次





# WevSecurityCondigクラスの作成①

まずは、ソースを書き写して下さい。

```
package blog.example.config;
                                                                                                                       return http.build();
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
                                                                                                               public static UserDetailsManager manager = null;
import org.springframework.boot.autoconfigure.security.servlet.PathRequest;
                                                                                                               @Autowired
import org.springframework.context.annotation.Bean;
                                                                                                               private UserService userService;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
                                                                                                               public UserDetailsService userDetailsService() {
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
                                                                                                                   List<UserDetails> users = userService.getAllAccounts().stream().map(
import org.springframework.security.core.userdetails.User;
                                                                                                                            account -> User.withDefaultPasswordEncoder()
import org.springframework.security.core.userdetails.UserDetails;
                                                                                                                            .username(account.getUserEmail())
import org.springframework.security.core.userdetails.UserDetailsService;
                                                                                                                            .password(account.getPassword())
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.provisioning.UserDetailsManager;
                                                                                                                            .roles("USER")
import org.springframework.security.web.SecurityFilterChain;
                                                                                                                            .build()
                                                                                                                           ).toList();
import blog.example.service.UserService;
                                                                                                                   manager = new InMemoryUserDetailsManager(users);
@Configuration
                                                                                                                   return manager;
@EnableWebSecurity
public class WebSecurityConfig {
        public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
                                                                                                               public static void addUser(String userEmail, String password) {
            http.formLogin(login -> login
                                                                                                                   manager.createUser(User.withDefaultPasswordEncoder()
                    .loginPage("/admin/login")
                                                                                                                            .username(userEmail)
                    .defaultSuccessUrl("/admin/blog/all",true)
                                                                                                                            .password(password)
                    .usernameParameter("userEmail") //リクエストパラメータのname属性を明示
                                                                                                                            .roles("USER")
                    .passwordParameter("password")
                                                                                                                            .build());
                    .failureUrl("/admin/login?error")
                    .permitAll()
            ).logout(logout -> logout
                    .logoutSuccessUrl("/admin/login")
            ).authorizeHttpRequests(authz -> authz
                    .requestMatchers(PathRequest.toStaticResources().atCommonLocations()).permitAll()
                    .requestMatchers("/blog/**","/admin/register","/css/**", "/js/**","/blog-image/**","/images/**").permitAll()
                    .anvRequest().authenticated()
```



return http.build();



# WevSecurityCondigクラスの作成②

● ソースの説明

```
カスタムなログイン画面を設定
package blog.example.config;
import java.util.List;
                                                                                  defaultSuccessUrl(String defaultSuccessUrl,
                                                                                  boolean alwaysUse)
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.autoconfigure.security.servlet.PathRequest;
                                                                                  ログイン成功後のURLを設定
import org.springframework.context.annotation.Bean;
                                                                                  ユーザーが認証前に保護されたページにアクセ
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
                                                                                  スしてない場合、またはaleaysUseがtrueの場
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.core.userdetails.User;
                                                                                  合に、認証に成功した後にゆーざーが移動する
import org.springframework.security.core.userdetails.UserDetails;
                                                                                  場所を指定する。
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.provisioning.UserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;
                                                                                   ログインする際のリクエスト
import blog.example.service.UserService;
                                                                                   パラメーターのname属性の設定
@Configuration
@EnableWebSecurity
public class WebSecurityConfig {
                                                                                  ログインに失敗した際のURLの設定
      public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
          http.formLogin(login -> login
                 .loginPage("/admin/login")
                                                                                   ログアウトに成功した際のURLの設定
                 .detauItSuccessUrI("/admin/blog/all",true)
                 .usernameParameter("userEmail") //ツクエストバラメータのname属性を明示
                 .passwordParameter("password")
                 .failureUrl("/admin/login?error")
                                                                                   URLごとの認可設定記述開始
                 .permitAll()
          ).logout(logout -> logout
                 .logoutSuccessUrl("/admin/login")
          ).authorizeHttpRequests(authz -> authz
                 .requestMatchers(PathRequest.toStaticResources().atCommonLocations()).permitAll()
                .requestMatchers("/blog/**","/admin/register","/css/**", "/js/**","/blog-image/**","/images/**").permitAll()
                 .anyRequest().authenticated()
                                                                                   ログイン無しでもアクセス可能なURLや
          return http.build();
                                                                                   パスの設定
```





# WevSecurityCondigクラスの作成③

● ソースの説明

```
return http.build();
public static UserDetailsManager manager = null;
@Autowired
private UserService userService;
public UserDetailsService userDetailsService() {
    List<UserDetails> users = userService.getAllAccounts().stream().map(
            account -> User.withDefaultPasswordEncoder()
            .username(account.getUserEmail())
            .password(account.getPassword())
            .roles("USER")
            .build()
            ).toList();
    manager = new InMemoryUserDetailsManager(users);
    return manager;
public static void addUser(String userEmail, String password) {
   manager.createUser(User.withDefaultPasswordEncoder()
            .username(userEmail)
                                                                     ユーザーの作成
            .password(password)
            .roles("USER")
            .build());
```

ユーザー情報をすべて取得して 一人一人の情報をメモリに保存している。





# UserServiceクラスのimport

● import文を追加してください。

```
1 package blog.example.service;
 3⊝ import java.util.List;
 5 import org.springframework.beans.factory.annotation.Autowired;
 6 import org.springframework.stereotype.Service;
 8
   import blog.example.model.dao.UserDao;
10 import blog.example.model.entity.UserEntity;
12 @Service
13 public class UserService {
        @Autowired
15
        private UserDao userDao;
16
          //ユーザの情報を保存する
17⊝
        public boolean createAccount(String userName,String userEmail, String password) {
18
            //RegisterControllerから渡されるユーザ情報(ユーザーメールアドレス)を条件にDB検索で検索する
19
            UserEntity userEntity = userDao.findByUserEmail(userEmail);
20
            //RegisterControllerから渡されるユーザ情報(ユーザ名、パスワード)を条件にDB検索で検索した結果
21
            //なかった場合には、保存
22
            if (userEntity==null) {
23
                userDao.save(new UserEntity(userName,userEmail, password));
                 webSecurityConfig.addUser(userEmail, password);
25
                  - Import "WebSecurityConfig" (blog.example.config)
26

    Create class 'WebSecurityConfig'

27
                  © Create record 'WebSecurityConfig'
28
                                                   import文をクリック
29
                  Create interface 'WebSecurityConfig'
30
        //ユーザのー • Create constant 'WebSecurityConfig'
        31⊖
32
                  © Create local variable 'WebSecurityConfig'
33
34
                  Change to 'WebFluxConfig' (org.springframework.boot.autoconfigure.web.reactive.WebFlux
35
        //idを見つ 🛊 Change to 'WebSecurity' (org.springframework.security.config.annotation.web.builders)
36
                 Change to 'WebSecurityConfiguration' (org.springframework.security.config.annotation.web
37⊝
                  Change to 'WebSecurityConfigurer' (org.springframework.security.config.annotation.web)
38
39
40
41 }
```

```
UserService.iava ×
   package blog.example.service;
 3⊖ import java.util.List;
   import org springframework beans factory appotation. Autowired;
   import org.springframework.stereotype.Service;
 8 import blog.example.config.WebSecurityConfig;
                                                importが追加されて
 9 import blog.example.model.dao.UserDao;
10 import blog.example.model.entity.UserEntity;
                                                 いることを確認
11
12 @Service
   public class UserService {
       @Autowired
       private UserDao userDao;
16
         //ユーザの情報を保存する
17⊜
       public boolean createAccount(String userName,String userEmail, String password) {
18
           //RegisterControllerから渡されるユーザ情報(ユーザーメールアドレス)を条件にDB検索で検索する
19
           UserEntity userEntity = userDao.findByUserEmail(userEmail);
           //RegisterControllerから渡されるユーザ情報(ユーザ名、パスワード)を条件にDB検索で検索した結果
           //なかった場合には、保存
           if (userEntity==null) {
               userDao.save(new UserEntity(userName,userEmail, password));
               WebSecurityConfig.addUser(userEmail, password);
               return true;
           } else {
               return false:
30
       //ユーザの一覧を取得する
       public List<UserEntity> getAllAccounts() {
31∈
32
           return userDao.findAll();
34
35
       //idを見つけるために
36
       //コントローラーでわたってきたuserEmailを基にしてDBを検索
37⊝
       public UserEntity selectById(String userEmail) {
38
           return userDao.findByUserEmail(userEmail);
39
40
41 }
```



- 1 UserDaoの設定
- 2 UserServiceクラスの設定
- 3 WebSecurityConfigの設定
- 4 RegisterControllerの設定
- 5 登録画面の作成

# 目 次





# RegisterControllerの作成①

● 下記のソースを書き写してください。

```
1 package blog.example.controller;
 30 import org.springframework.beans.factory.annotation.Autowired;
 4 import org.springframework.stereotype.Controller;
 5 import org.springframework.web.bind.annotation.GetMapping;
 6 import org.springframework.web.bind.annotation.PostMapping;
 7 import org.springframework.web.bind.annotation.RequestMapping;
 8 import org.springframework.web.bind.annotation.RequestParam;
 9 import blog.example.service.UserService;
10
11 @Controller
12 @RequestMapping("/admin")
13 public class ReigisterController {
149
       @Autowired
       private UserService accountService;
16
       //登録画面を表示
       @GetMapping("/register")
19
       public String getRegisterPage() {
20
           return "admin_register.html";
       //ユーザー情報の登録
23⊜
       @PostMapping("/register")
       public String register(@RequestParam String userName,@RequestParam String userEmail,
               @RequestParam String password) {
           //もし保存をした場合には、login.htmlへ遷移する
           if (accountService.createAccount(userName, userEmail,password)) {
               return "redirect:/admin/login";
           } else {
               //そうでない場合には、register.htmlに遷移する
               return "admin register.html";
```





## RegisterControllerの作成②

● ソースの説明

@Controller HTMLを返すControllerであることを示す

@RequestMappingコントローラークラスで共通で利用するパスを指定できる。@GetMapiing("/register")であれば、URLは「/admin/register」となる





# RegisterControllerの作成③

#### サンースの説明

```
11 @Controller
12 @RequestMapping("/admin")
  public class ReigisterController {
140
      @Autowired
15
      private UserService accountService;
16
                                             指示をしている。
      //登録画面を表示
17
189
      @GetMapping("/register")
19
      public String getRegisterPage() {
                                             味する。
          return "admin register.html";
20
21
22
      //ユーザー情報の登録
23⊜
      @PostMapping("/register")
      public String register(@RequestParam String userName,@RequestParam String userEmail,
24
              @RequestParam String password) {
          //もし保存をした場合には、login.htmlへ遷移する
          if (accountService.createAccount(userName, userEmail,password)) {
28
              return "redirect:/admin/login";
          } else {
                                                        @PostMapping
              //そうでない場合には、register.htmlに遷移する
              return "admin register.html";
33
34
35
```

@GetMapping ユーザーがURL

「http://localhost:8080/register/admin」にア クセルするときのHTTPリクエストに応答した時、 このgetRegisterPage()メソッドを使用するように

Getメソッドによるリクエストに応答することを意

「localhost:8080/admin/register」に 送信されるPOSTリクエストを処理するこ とを示す。HTMLのformタグのactionと methodに対応している。





# RegisterControllerの作成4

● ソースの説明

```
このリクエストが
  @Controller
  @RequestMapping("/admin")
                                           「userName」と「userEmail」と「password」
  public class ReigisterController {
                                          の3つのパラメータを持つことを示す。
      @Autowired
148
15
      private UserService accountService;
                                         inputタグやtextareaタグ、selectタグ等にある
16
                                          name属性に対応している。
      //登録画面を表示
17
188
      @GetMapping("/register")
19
      public String getRegisterPage() {
20
         return "admin_register.html";
21
22
      //ユーザー情報の登録
238
      @PostMapping("/register")
24
      public String register(@RequestParam String userName,@RequestParam String userEmail,
25
             @RequestParam String password) {
         //もし保存をした場合には、login.htmlへ遷移する
26
         if (accountService.createAccount(userName, userEmail,password)) {
             return "redirect:/admin/login";
         } else {
             //そうでない場合には、register.htmlに遷移する
             return "admin register.html";
```

@RequestParam



# RegisterControllerの作成⑤

```
<section class="register-section">
   <h2>管理者登録画面</h2>
                                                                       HTMLの各属性が
   <form method="POST" th:action="@{/admin/register}">
      <div class="register-section-details">
                                                                       Controllerクラスの何の
          <div class="register-section-details flex">
                                                                      パラメータやアノテーショ
             <div>UserName</div>
             <input type="text" name="userName" value="">
                                                                       ンに対応しているかを必ず
          </div>
          <div class="register-section-details flex">
                                                                       理解してください。
             <div>UserEmail</div>
             <input type="text" name="userEmail" value="">
          </div>
          <div class="register-section-details flex">
             <div>Password</div>
             <input type="text" name="password" value="">
          </div>
      </div>
      <div class="register-section-details flex">
             <button id="register" class="edit disable">登録</button>
           <button class="delete" onclick="history.back();" type="button">戻る</button>
                                     //ユーザー情報の啓録
      </div>
                                     @PostMapping("/register")
   </form>
                                     public String register(@RequestParam String userName,@RequestParam String userEmail,
 section>
                                            @RequestParam String password) {
                                         //もし保存をした場合には、login.htmlへ遷移する
                                         if (accountService.createAccount(userName, userEmail,password)) {
                                            return "redirect:/admin/login";
                                         } else {
                                            //そうでない場合には、register.htmlに遷移する
                                            return "admin register.html";
```





# RegisterControllerの作成⑥

● ソースの説明

```
@Controller
12 @RequestMapping("/admin")
  public class ReigisterController {
     @Autowired
140
15
      private UserService accountService;
16
17
      //啓録画面を表示
                                            登録画面を表示させる
      @GetMapping("/register")
189
                                            admin_registerを表示する
19
      public String getRegisterPage() {
         return "admin_register.html";
20
21
      //フーザー情報の啓録
22
      @PostMapping("/register")
      public String register (@RequestParam String userName, @RequestParam String userEmail,
            @RequestParam String password) {
         //もし保存をした場合には、login.htmlへ遷移する
         if (accountService.createAccount(userName, userEmail,password)) {
            return "redirect:/admin/login";
                                                  @RequestParamでview側からパラメ-
         } else {
            //そうでない場合には、register.htmlに遷移する
                                                 夕を受け取る
            return "admin_register.html";
                                                 Serviceクラスのメソッドを使って保存が
                                                 できた場合には、ログイン画面へリダイ
                                                  レクト
                                                 そうでない場合は、登録画面へ遷移する
```



- 1 UserDaoの設定
- 2 UserServiceクラスの設定
- 3 WebSecurityConfigの設定
- 4 RegisterControllerの設定
- 5 登録画面の作成

# 目 次





### 登録画面の作成①

● Templatesフォルダの中に「admin\_register.html」を作成し、 ソースを書き写してください

```
フォントは下記のURLから自分で探してください。
                                           https://fonts.google.com/specimen/Kiwi+Maru
admin_register.html ×
                                           https://fonts.google.com/specimen/Gochi+Hand
 1 <!DOCTYPE html>
 20 <html lang="en">
                                          ?query=gochi
 3
 48 <head>
       <meta charset="UTF-8">
       <meta http-equiv="X-UA-Compatible" content="IE=edge">
 6
       <meta name="viewport" content="width=device-width, initial-scale=1.0">
 7
                                                                       Thymeleafを使用してcss
 8
       <title>Document</title>
       <link rel="stylesheet" th:href="@{/css/reset.css}">
                                                                       を読み込む
 9
       <link rel="stylesheet" th:href="@{/css/style.css}">
10
       k rel="preconnect" href="https://fonts.googleapis.com">
11
       k rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
12
       <link href="https://fonts.googleapis.com/css2?family=Kiwi+Maru:wght@300;400;500&display=swap" rel="stylesheet">
13
       k rel="preconnect" href="https://fonts.googleapis.com">
14
       k rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
15
       <link href="https://fonts.googleapis.com/css2?family=Gochi+Hand&family=Kiwi+Maru:wght@300;400;500&display=swap">
16
17 </head>
```

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/913646 e31bd2064ba5c9





### 登録画面の作成②

● Templatesフォルダの中に「admin\_register.html」を作成し、 ソースを書き写してください。

```
Thymeleafを使用して画
<body>
<header>
                                                  像を読み込む
      <!--スマートフォン-->
      <nav class="menu">
         <div class="logo">
            <a href="#"><img th:src="@{\images\logo.png}"></a>
         <div class="menu-contents">
            <div class="menu-inner">
                <l
                   <a th:href="@{/login}">ログイン</a>
                </div>
            <div class="menu-toggle_btn">
                <span></span>
                <span></span>
                <span></span>
            </div>
         </div>
      </nav>
      <!--pc-->
      <nav class="pc">
         <div class="pc-inner">
            <div class="pc-logo">
                <a href=""><img th:src="@{\images\logo.png}"></a>
            </div>
            <a th:href="@{/login}">ログイン</a>
            </div>
      </nav>
   </header>
```



## 登録画面の作成③

● Templatesフォルダの中に「admin\_register.html」を作成し、 ソースを書き写してください。

```
<main>
        <div class="main-inner">
            <section class="register-section">
                <h2>管理者登録画面</h2>
                <form method="POST" th:action="@{/admin/register}":</pre>
                    <div class="register-section-details">
                        <div class="register-section-details flex">
                            <div>UserName</div>
                            <input type="text" name="userName" value="">
                        <div class="register-section-details_flex">
                            <div>UserEmail</div>
                            <input type="text" name="userEmail" value="">
                        <div class="register-section-details flex">
                            <div>Password</div>
                            <input type="text" name="password" value="">
                        </div>
                    </div>
                    <div class="register-section-details flex">
                            <button id="register" class="edit_disable">登録</button>
                          <button class="delete" onclick="history.back();" type="button">戻る</button>
                    </div>
                </form>
            </section>
        </div>
    </main>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"</pre>
            integrity="sha256-/xUj+30JU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4=" crossorigin="anonymous"></script>
        <script th:src="@{/js/common.js}"></script>
</body>
</html>
```

Thymeleafを使用して action先を設定

JqeuryのCDNは以下のURLから探して 該当するものを使用してください。 https://releases.jquery.com/jquery



## 登録画面の作成4

● staticフォルダーに指定したフォルダを作成してください

- - static
    - blog-image
    - > > CSS
    - images
    - 🗦 🗁 js
  - templates
    - application.properties





### 登録画面の作成⑤

● staticフォルダーに指定したフォルダ画像を入れてください。



下記のリンクからimages.zipを ダウンロードしてください https://abiding-sandal-008.notion.site/84e4acd264bd4cad ba7df2a4d4ac4ba9







### 登録画面の作成⑤

● cssフォルダにreset.cssとstyle.cssを作成してください。

```
✓ ₾ src/main/resources
✓ ৶ static
→ ৶ blog-image
✓ ৶ css
☑ reset.css
☑ style.css
→ ৶ images
→ ৶ js
→ ৶ templates
✓ application.properties
```





### 登録画面の作成6

```
    □ reset.css ×

 1⊖/* http://meyerweb.com/eric/tools/css/reset/
 2 v2.0 20110126
 3 License: none (public domain)
 4 */
 6 html, body, div, span, applet, object, iframe,
 7 h1, h2, h3, h4, h5, h6, p, blockquote, pre,
 8 a, abbr, acronym, address, big, cite, code,
9 del, dfn, em, img, ins, kbd, q, s, samp,
10 small, strike, strong, sub, sup, tt, var,
11 b, u, i, center,
12 dl, dt, dd, ol, ul, li,
13 fieldset, form, label, legend,
14 table, caption, tbody, tfoot, thead, tr, th, td,
15 article, aside, canvas, details, embed,
16 figure, figcaption, footer, header, hgroup,
17 menu, nav, output, ruby, section, summary,
18⊖ time, mark, audio, video {
19 margin: 0;
     padding: 0;
    border: 0;
21
22
     font-size: 100%;
23
    font: inherit;
       vertical-align: baseline;
24
25 }
26 /* HTML5 display-role reset for older browsers */
27 article, aside, details, figcaption, figure,
28⊖ footer, header, hgroup, menu, nav, section {
       display: block;
30 }
31<sup>©</sup> body {
       line-height: 1;
33 }
34⊖ ol, ul {
       list-style: none;
36 }
37⊝ a{
38
       text-decoration: none;
39 }
40⊖ blockquote, q {
       quotes: none;
```

```
25 }
26 /* HTML5 display-role reset for older browsers */
27 article, aside, details, figcaption, figure,
28⊖ footer, header, hgroup, menu, nav, section {
       display: block;
29
30 }
31<sup>©</sup> body {
       line-height: 1;
32
33 }
34⊖ ol, ul {
       list-style: none;
35
36 }
37⊖ a{
       text-decoration: none;
38
39 }
40⊖ blockquote, q {
41
       quotes: none;
42 }
43 blockquote:before, blockquote:after,
44⊖ q:before, q:after {
       content: '';
45
        content: none;
46
47 }
48⊖ table {
       border-collapse: collapse;
49
       border-spacing: 0;
50
```





### 登録画面の作成で

```
1 /*====== 共通設定CSS =======*/
  2⊖img {
        width: 100%;
        height: auto;
        vertical-align: top;
  6 }
  7
  8⊖ main {
        margin-top: 6rem;
 10 }
 11
 12
 13 /*======= ヘッダーのCSS =======*/
 14⊖ header {
       background-color: ☐ rgba(255, 255, 255, .7);
 15
        height: 6rem;
        position: fixed;
 17
 18
       width: 100%;
 19
        z-index: 100;
 20
        top: 0:
 21
        border-bottom: 1px solid ■ black;
 22 }
 23
 24 /*ヘッダーの幅を綺麗にそろえるために要素の大きさを均一にする*/
 25⊖.logo, .menu-contents {
        width: 4rem;
                                                続き
 27 }
 28
 29⊖.pc {
 30
        display: none;
 31 }
 32
 33⊖.menu {
        display: flex;
        justify-content: space-between;
 35
        align-items: center;
 36
 37
        width: 90%;
 38
        margin: 0 auto;
 39
        padding-top: 0.5rem;
 40 }
```

```
33⊖.menu {
34
       display: flex;
35
       justify-content: space-between;
36
       align-items: center;
37
       width: 90%;
38
       margin: 0 auto;
39
       padding-top: 0.5rem;
40 }
41
42⊖.logo img {
43
       width: 8rem;
44
       height: 1.6rem;
45 }
46
47 /*ボタン外側*/
48⊖ .menu-toggle btn {
       position: relative;
       /*ボタン内側の基点となるためrelativeを指定*
50
51
52
       cursor: pointer;
53
       width: 4rem;
54
       height: 4rem;
55
       border-radius: 0.5rem:
56 }
57
58 /*ボタン内側*/
59⊖ .menu-toggle_btn span {
       display: inline-block;
       transition: all .4s;
       /*アニメーションの設定*/
       position: absolute;
       left: 1rem;
65
       height: 3px;
       border-radius: 2px;
66
67
       background: #EC6015;
68
       width: 45%;
69 }
70
71⊖.menu-toggle_btn span:nth-of-type(1) {
72
       top: 1.5rem;
73 }
```

```
719 .menu-toggle_btn span:nth-of-type(1) {
72
       top: 1.5rem;
73 }
74
75⊖.menu-toggle btn span:nth-of-type(2) {
       top: 2rem;
77 }
78
79⊖ .menu-toggle btn span:nth-of-type(3) {
       top: 2.5rem;
81 }
82
83 /*activeクラスが付与されると線が回転して×に*/
85⊖.menu-toggle_btn.active span:nth-of-type(1) {
86
       top: 1.5rem;
87
       left: 1.5rem;
88
       transform: translateY(6px) rotate(-45deg);
89
       width: 30%;
90
       z-index: 1;
91 }
93⊖.menu-toggle btn.active span:nth-of-type(2) {
       opacity: 0;
       /*真ん中の線は透過*/
96
       z-index: 1:
97 }
99⊖ .menu-toggle_btn.active span:nth-of-type(3) {
       top: 2.25rem;
101
       left: 1.5rem;
       transform: translateY(-6px) rotate(45deg);
102
103
       width: 30%;
104
       z-index: 1;
105 }
106
107
```



#### 登録画面の作成®

```
107
1089 /*-----
109 * メニュー本体
110 *----*/
111⊖ .menu-inner {
       display: none;
113 }
114
115⊖ .menu-inner.active {
       position: fixed;
117
       top: 6rem;
118
       right: 0;
119
       width: 100vw;
120
       height: 100vh;
121
       display: flex;
122
       flex-direction: column;
123
       align-items: center;
124
       justify-content: center;
125
       background: □ #fff;
126
       z-index: 1;
127 }
128
129⊖ .menu item {
       margin-bottom: 3rem;
130
       text-align: center;
131
132 }
133
134⊖ .menu_item a {
135
       background-color: #EC6015;
       padding-top: 1rem;
136
137
       padding-bottom: 1rem;
       display: block;
138
139
       color: □#fff;
       font-family: Kiwi Maru;
140
141
       text-align: center;
142
       text-decoration: none;
143
       width: 10rem;
144 }
145
```



```
144 }
145
146⊖ .menu item a:hover {
      background-color: □ #ffefcc;
148
      color: ■ black;
149 }
150
151 /*-----*/
153
       メインの設定
154 -----*/
155⊝ * {
156
      box-sizing: border-box;
157 }
158
159⊖ .main-inner {
160
      width: 100%;
161
      margin: 0 auto;
162 }
163
164 /*-----*/
165 .btn--orange,
166⊖ a.btn--orange {
167
      color: □#fff;
168
      background-color: #eb6100;
169
      padding: 0.5rem 1rem;
170
      box-shadow: 0 5px 0 ■ #aaaaaa;
      transition: 0.3s;
171
172
      font-family: Kiwi Maru;
173
174 }
```





### 登録画面の作成9

```
174 }
175
176 .btn--orange:hover,
177⊖ a.btn--orange:hover {
     background-color: □#ffefcc;
       color: ■ black;
180
       transform: translateY(3px);
       text-decoration: none;
182
       box-shadow: 0 2px 0 #aaaaaa;
183 }
185⊖a.btn--radius {
     border-radius: 3rem;
187 }
188
190 /*------編集ボタン-----*/
191⊖ .edit disable {
     /*background-color:#71c7c8;*/
       background-color: #a6a197;
193
       color: □#fff;
195
       border: ■ gray;
       padding: 0.5rem 1rem;
196
197
       box-shadow: 0 5px 0 ■ #aaaaaa;
198
       transition: 0.3s;
199 }
200
201⊖ .edit {
        /*background-color:#71c7c8;*/
202
203
       background-color: #f59e0b;
       color: □#fff;
205
       border: ■ gray;
206
        padding: 0.5rem 1rem;
207
       box-shadow: 0 5px 0 ■ #aaaaaa;
        transition: 0.3s;
208
209 }
210
211⊖ .edit:hover {
212
       transform: translateY(3px);
213
       text-decoration: none;
       box-shadow: 0 2px 0 ■ #aaaaaa;
215 }
```



```
215 }
216
217
218 /*-----
220⊖ .back-btn {
      background-color: #615b51;
222
      color: □#fff;
223
      border: ■ gray;
224
      padding: 0.5rem 1rem;
225
      box-shadow: 0 5px 0 #aaaaaa;
226
      transition: 0.3s;
227 }
228
229⊖ .back-btn:hover {
      transform: translateY(3px);
231
      text-decoration: none;
232
      box-shadow: 0 2px 0 ■ #696868;
233 }
234 /*----*/
235⊖ .delete {
      background-color: #715494;
236
237
      color: □#fff;
238
      border: gray;
239
      padding: 0.5rem 1rem;
      box-shadow: 0 5px 0 #aaaaaaa;
240
241
      transition: 0.3s;
242 }
243
244⊖ .delete:hover {
      transform: translateY(3px);
245
246
      text-decoration: none;
247
      box-shadow: 0 2px 0 ■ #aaaaaa;
248 }
```





### 登録画面の作成⑩

```
250 /*----*/
2519 /*-----
                     ジャンル一覧
252
254⊖ .genre {
255 width: 90%;
256
      margin: 0 auto;
257
      padding-top: 2rem;
258 }
259
260⊖ .genre-register button {
      text-align: right;
261
262 }
263
264⊖ .genre table {
    width: 100%;
266
      margin: 2rem auto;
267
      text-align: center;
268
      border: 1px solid ■ gray;
269 }
270
271⊖ .genre th, .genre td {
      border: 1px solid ■ gray;
273
      text-align: center;
274
      padding: 1rem 0.5rem;
275
276 }
277
278⊖ .genre th {
279
      background-color: #71c7c8;
280
      color: □#fff;
281 }
282

  284
  登録画面

  285
  -----*/

286⊖ .register-section {
      width: 90%;
288
      margin: 0 auto;
289
      padding-top: 2rem;
290 }
```



```
290 }
291
292 .register-section h2 {
        font-family: Kiwi Maru;
294
        font-size: 1.3rem:
295
        text-align: center;
296
        color: ■ #594A4E;
297
        font-weight: bold;
298 }
299
300 .register-section-details {
        width: 100%;
301
302
        margin: 3rem auto;
303 }
304
305 svg {
        width: 30px;
306
307
        height: 30px;
308
        vertical-align: middle
309 }
310
311 .register-section-details_flex {
312
        display: flex;
313
        justify-content: center;
314
        gap: 0 1rem;
315
        align-items: center;
316
        margin: 3rem 0;
317 }
318⊖ .register-section-details flex>img {
319
        width: 252px;
320 }
3210 .register-section-details_flex div:first-child {
        color: ■ #594A4E;
322
323
        font-family: Kiwi Maru;
324
        width: 6rem;
325
        line-height: 1.5rem;
326
        font-size: 1rem;
327 }
```





### 登録画面の作成①

```
328 /*--登録のボタンの位置の設定--*/
329 #store add box{
330 margin-left: 15rem;
333 input[type="text"] {
       padding: 0.5rem 1rem;
335
       border-radius: 4px;
336
       border: none;
337
       box-shadow: 0 0 0 1px #ccc inset;
338
       appearance: none;
339
       -webkit-appearance: none;
340
       -moz-appearance: none;
341
       font-size: 1rem;
342
       color: ■ #594A4E;
343
       font-family: Kiwi Maru;
344
       width: 250px;
345 }
347⊖ input[type="text"]:focus {
       outline: 0;
349
        box-shadow: 0 0 0 2px #71c7c8 inset;
350 }
351
352⊖ input[type="time"] {
       padding: 0.5rem 1rem;
       border-radius: 4px;
354
355
       border: none;
356
       box-shadow: 0 0 0 1px #ccc inset;
357
       appearance: none;
        -webkit-appearance: none;
359
        -moz-appearance: none;
360
       font-size: 1rem;
361
       color: ■ #594A4E;
362
       font-family: Kiwi Maru;
363
       width: 250px;
364 }
365
366⊖ input[type="time"]:focus {
367
       outline: 0;
368
       box-shadow: 0 0 0 2px ■ #71c7c8 inset;
369 }
```

```
369 }
370
371⊖ textarea {
372
        resize: vertical;
373
        padding: 0.5rem 1rem;
374
        border-radius: 4px;
375
        border: none;
376
        box-shadow: 0 0 0 1px □ #ccc inset;
377
        appearance: none;
378
        -webkit-appearance: none;
379
        -moz-appearance: none;
380
        font-size: 1rem;
381
        color: #594A4E;
382
        font-family: Kiwi Maru;
383
        width: 250px;
384 }
385
386⊖ textarea: focus {
387
        outline: 0:
388
        box-shadow: 0 0 0 2px #71c7c8 inset;
389 }
390
391⊖ select {
        padding: 0.5rem 1rem;
392
        border-radius: 4px;
393
394
        border: none;
395
        box-shadow: 0 0 0 1px #ccc inset;
396
        font-size: 1rem;
397
        color: ■ #594A4E;
398
        font-family: Kiwi Maru;
399
        width: 250px;
400 }
401
402⊖ select:focus {
403
        outline: 0;
404
        box-shadow: 0 0 0 2px #71c7c8 inset;
```





## 登録画面の作成⑫

```
405 }
406
407⊖.checkbox__flex {
        width: 250px;
409 }
410
411 input[type="checkbox"] {
412
        margin-right: 1rem;
413
        margin-bottom: 1rem;
414 }
415
416⊖ .photo_img {
417
        width: 250px;
418 }
419
420 .photo_img img {
       width: 80%;
        box-shadow: 0 0 0 1px □ #ccc inset;
423 }
424 .photo_img video {
        display: block;
        width: 80%;
427
        box-shadow: 0 0 0 1px □ #ccc inset;
428 }
429
434 .all-view-section {
435
        width: 90%;
        margin: 0 auto;
437
        padding-top: 2rem;
438 }
439
440⊖ .all-view-section h2 {
       font-family: Kiwi Maru;
        font-size: 1.3rem;
        text-align: center;
444
        color: ■ #594A4E;
445
        font-weight: bold;
446 }
```



```
447
448 .all-view-register button {
       text-align: right;
        margin: 3rem 0;
451 }
452
453@.all-view-article {
       border: 1px solid #594A4E;
       margin-bottom: 2rem;
456
       box-shadow: 0 5px 0 #aaaaaa;
457
458 }
459
460 .all-view-colum {
       position: relative;
462
        font-family: Kiwi Maru;
463 }
464
465@.all-view-colum img {
       background-color: #fff;
467
        border-bottom: 1px solid #594A4E;
468 }
469
4709 .all-view category {
       position: absolute;
472
       z-index: 1;
473
        left: 0;
474
       background-color: #594A4E;
        color: □#fff;
475
476
        padding: 0.5rem 1rem;
477 }
478
4798 .message {
        box-sizing: border-box;
481
        width: 90%;
482
        padding: 8px 19px;
483
        margin: 2em auto;
484
        color: #2c2c2f;
485
        background: #fff;
        border-top: solid 5px ■ #594A4E;
487
        border-bottom: solid 5px ■ #594A4E;
488 }
489
```





### 登録画面の作成(13)

```
488 }
489
4900 message p {
       margin: 0;
492
       padding: 0;
493
       word-wrap: break-word;
494 }
495
カテゴリー設定
498 -----*/
499 .colum box {
500
      padding: 1rem;
       background-color: #fff;
501
       color: #594A4E;
502
503
       font-family: Kiwi Maru;
       display: flex;
505
       justify-content: center;
506
       gap: 1rem;
507
       align-items: center;
508 }
509⊕.colum box noflex{
510
       padding: 1rem;
511
       background-color: #fff;
512
       color: #594A4E;
513
       font-family: Kiwi Maru;
514 }
515 .colum box_noflex p:nth-child(1) {
516
       text-align: left:
       font-size: 1rem:
517
518 }
519
520 .colum box noflex p:nth-child(2) {
521
       text-align: left;
522
       padding: 1.2rem 0;
523
       font-size: 1.2rem;
524 }
525
526 .colum box noflex p:nth-child(3) {
527
       text-align: right;
528
       font-size: 1rem;
529 }
```



```
530 @media screen and (min-width: 768px) {
        /*----*/
532
533⊜
534
           display: block;
535
           padding-top: 1.5rem;
536
537
538⊜
        .menu {
539
            display: none;
540
541
    .pc-inner {
543
           display: flex;
           justify-content: space-between;
545
           width: 90%;
546
           margin: 0 auto;
547
           padding-top: 1rem;
548
549
5508
       .pc-logo {
           display: flex;
552
           text-align: center;
553
           gap: 0 10px;
554
            align-items: center;
555
556
557
        .pc-logo img {
558
           width: 10rem;
559
           height: 2rem;
560
561
562⊕
        .pc-logo a:hover {
563
           line-height: 1.2rem;
564
           color: #ffefcc;
565
            font-family: Kiwi Maru;
566
```





### 登録画面の作成4

● style.cssファイルに下記のソースを書き写してください。 このページでstyle.cssは完了です。

```
566
567
568
        .pc-list {
569
            display: flex;
570
            align-items: center;
571
            gap: 0 1.5rem;
572
573
574⊖
        .pc-list a {
575
            font-size: 1.2rem:
576
            font-weight: 600;
577
            color: #594A4E;
578
            font-family: Kiwi Maru;
579
580
581⊕
        .pc-list a:hover {
582
            color: = #EC6015;
583
584
585
                             一覧画面
586
587
588
589⊕ .all-view-section {
590
        width: 800px;
        margin: 0 auto;
591
592
        padding-top: 2rem;
593 }
594⊕ .all-view flex{
595
        display: flex;
596
        flex-wrap: wrap;
597
        width: 800px;
598
        gap: 10px;
599
        justify-content: center;
600 }
```



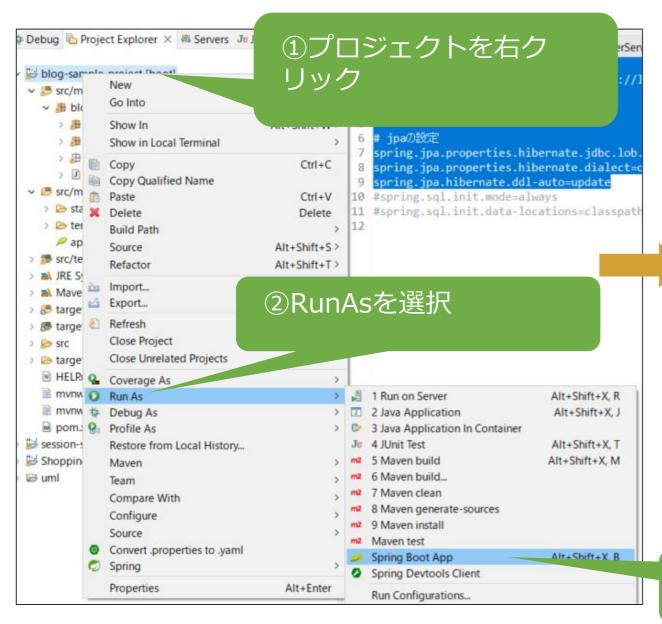
```
600 }
601 .all-view-article {
        width: 33%;
602
        box-sizing: border-box;
603
694
605 }
606
607 .colum box p:nth-child(1) {
        text-align: left;
608
        font-size: 1rem;
609
610 }
611
612 .colum box p:nth-child(2) {
        text-align: left;
613
        padding: 1.2rem 0;
614
        font-size: 1.2rem;
615
616 }
617
618@.colum box p:nth-child(3) {
        text-align: right;
619
        font-size: 1rem;
620
621 }
622
623
624 }
```





## 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。



```
④エラーがないかを確認
Console × papplication.properties
log-sample-project - BlogSampleProjectApplication [Sprin
 :: Spring Boot ::
2022-12-12T16:30:39.007+09:00 INFO 19296 ---
                                                         main] b.example.BlogSampleProjectApplication : Startin
                                                         main] b.example.BlogSampleProjectApplication : No acti
2022-12-12T16:30:39.013+09:00
2022-12-12T16:30:40.046+09:00
                              INFO 19296 ---
                                                         main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstr
2022-12-12T16:30:40.084+09:00
                                                         main] .s.d.r.c.RepositoryConfigurationDelegate : Finishe
2022-12-12T16:30:40.813+09:00
                                                         main] o.s.b.w.embedded.tomcat.TomcatWebServer
2022-12-12T16:30:40.828+09:00
                                                         main] o.apache.catalina.core.StandardService
                                                                                                        : Startin
                                                         main] o.apache.catalina.core.StandardEngine
2022-12-12T16:30:40.828+09:00
                                                                                                        : Startin
2022-12-12T16:30:41.010+09:00
                                                         main] o.a.c.c.C.[Tomcat].[localhost].[/]
                                                                                                        : Initial
2022-12-12T16:30:41.011+09:00
                                                         main] w.s.c.ServletWebServerApplicationContext : Root We
2022-12-12T16:30:41.236+09:00
                                                         main] o.hibernate.jpa.internal.util.LogHelper :
                                                                                                          HHH0002
2022-12-12T16:30:41.313+09:00
                              TNFO 19296 ---
                                                         main] org.hibernate.Version
                                                                                                        · HHH0004
2022-12-12T16:30:41.555+09:00
                                                         main] org.hibernate.orm.deprecation
                                                                                                        : HHH9000
2022-12-12T16:30:41.738+09:00
                              INFO 19296 ---
                                                         main] com.zaxxer.hikari.HikariDataSource
                                                                                                        : HikariP
2022-12-12T16:30:41.977+09:00
                                                         main] com.zaxxer.hikari.pool.HikariPool
                                                                                                        : HikariP
2022-12-12T16:30:41.980+09:00
                                                         main] com.zaxxer.hikari.HikariDataSource
                                                                                                        : HikariP
                                                         main | SOL dialect
                                                                                                        : HHH0004
0022-12-12T16:30:42.033+09:00
2022-12-12T16:30:42.428+09:00
                                                         main] o.h.e.t.j.p.i.JtaPlatformInitiator
2022-12-12T16:30:42.449+09:00
                                                         main] j.LocalContainerEntityManagerFactoryBean : Initial
2022-12-12T16:30:42.503+09:00
                                                         main] JpaBaseConfiguration$JpaWebConfiguration : spring.
2022-12-12T16:30:42.861+09:00
                              WARN 19296 ---
                                                         main] ion$DefaultTemplateResolverConfiguration : Cannot
2022-12-12T16:30:43.035+09:00 WARN 19296 ---
                                                         main] .s.s.UserDetailsServiceAutoConfiguration :
Using generated security password: 13a800c1-0c0b-4da0-a501-534369ab3fec
his generated password is for development use only. Your security configuration must be updated before running yo
2022-12-12T16:30:43.191+09:00 INFO 19296 ---
                                                         main] o.s.s.web.DefaultSecurityFilterChain
                                                         main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat
022-12-12T16:30:43.282+09:00 INFO 19296 ---
2022-12-12T16:30:43.295+09:00 INFO 19296 ---
                                                         main] b.example.BlogSampleProjectApplication
```

③SpringBootAppをクリック





## 動作確認②

● 最後は、登録できるかどうかの確認をしていきます。

①ブラウザの検索欄に以下のURLを入力Enter

http://localhost:8080/admin/register → C ① localhost:8080/admin/register 🎹 アプリ Ĉ Stageee - ステージー 🐕 Chatwork 쥥 ショッピングリスト 🛆 COACHTEO ログイン 管理者登録画面 UserName UserEmail 使用するブラウザーは 「chrome」にしてください。 Password Baiduは使用を禁止します。 ②3項目を入力して「登録」 ボタンを押下してください

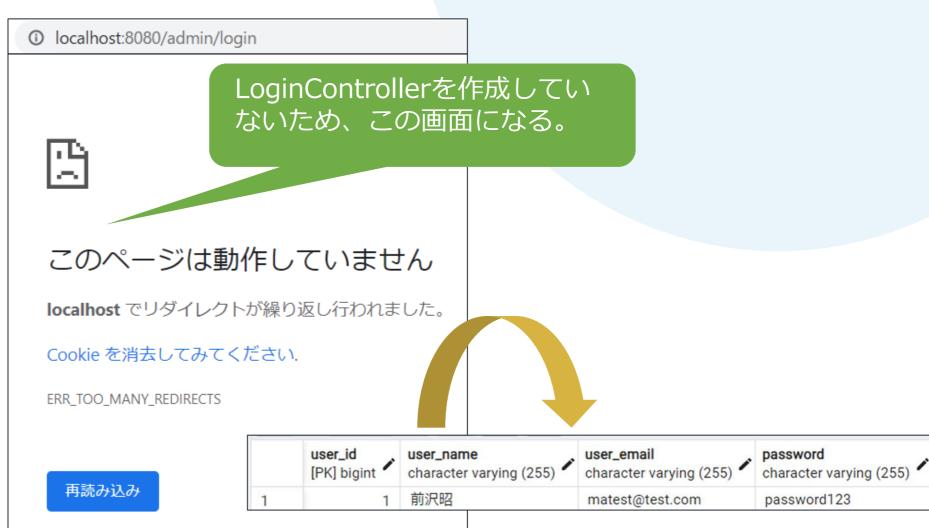




# 動作確認③

● 最後は、登録できるかどうかの確認をしていきます。





DBに保存できているかを確認 する。保存されていれば成功







Light in Gour Career.

**THANK YOU!**