LightHouse Academy

# 1.4 PersonProject　初級

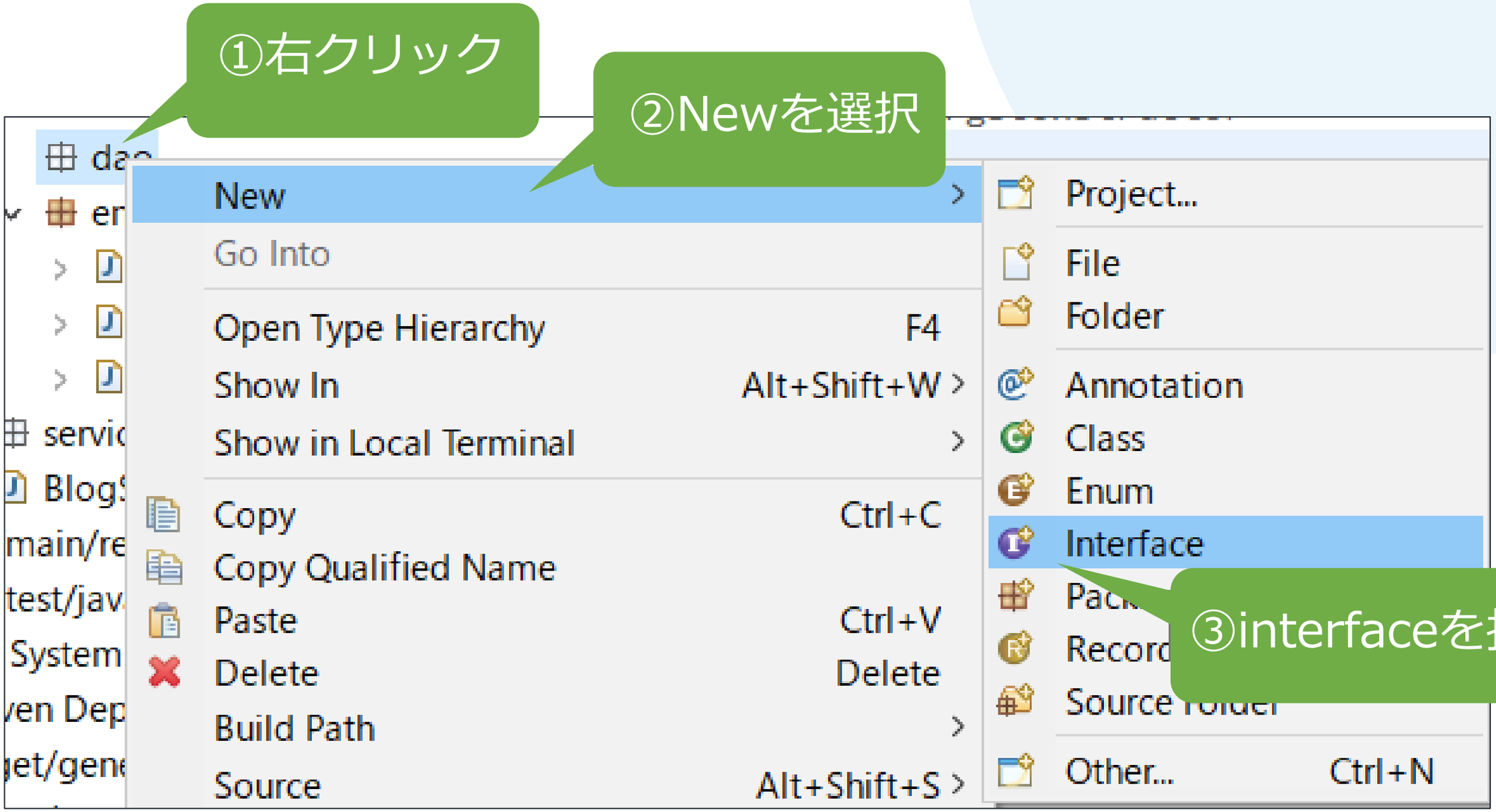
- UserDaoクラスの設定
- UserServiceクラスの設定
- RegisterControllerの設定
- WebConfigの設定
- 登録画面の作成

www.lighthouseit.academy

# 目次

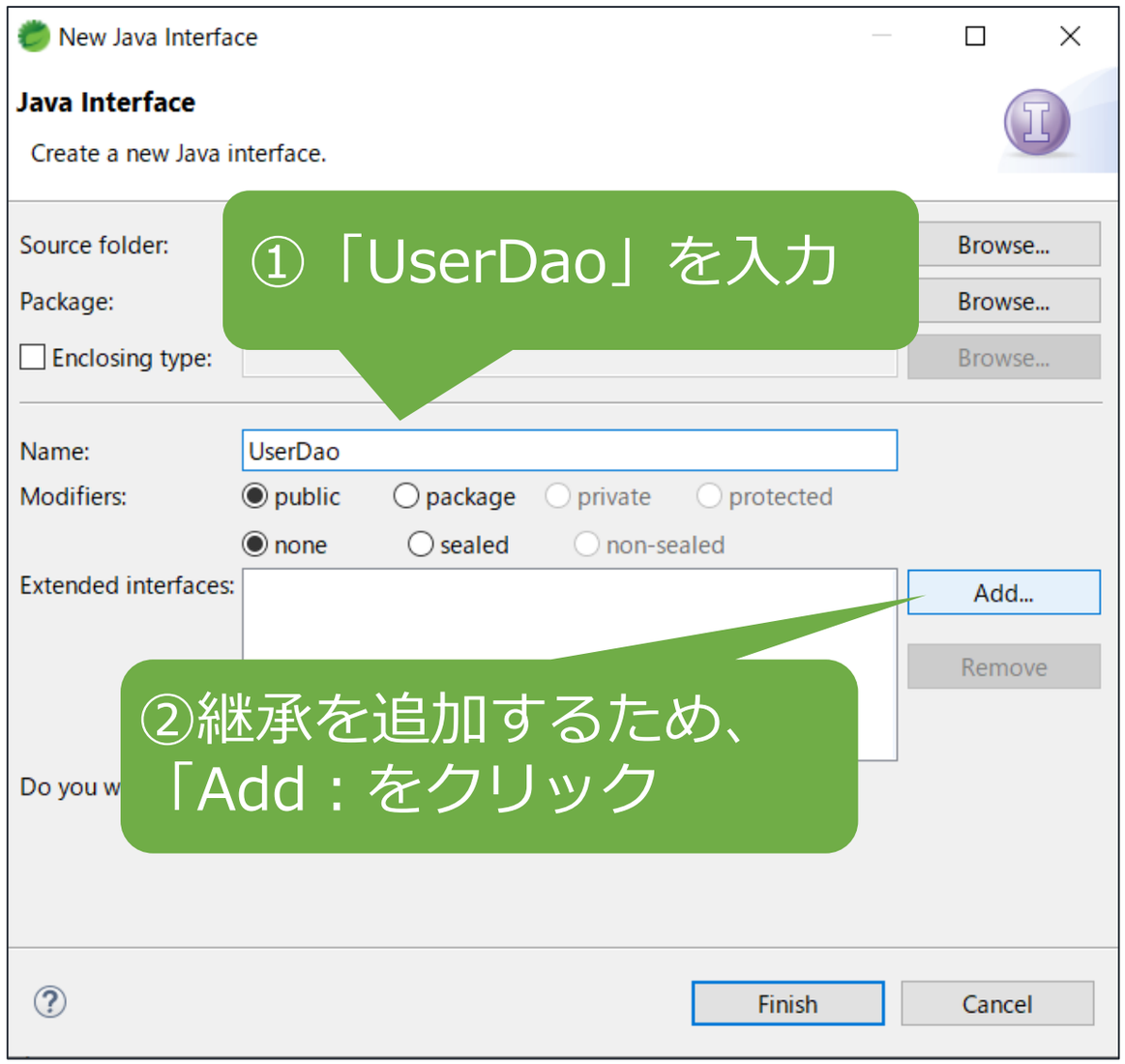
# UserDaoインターフェースの作成①

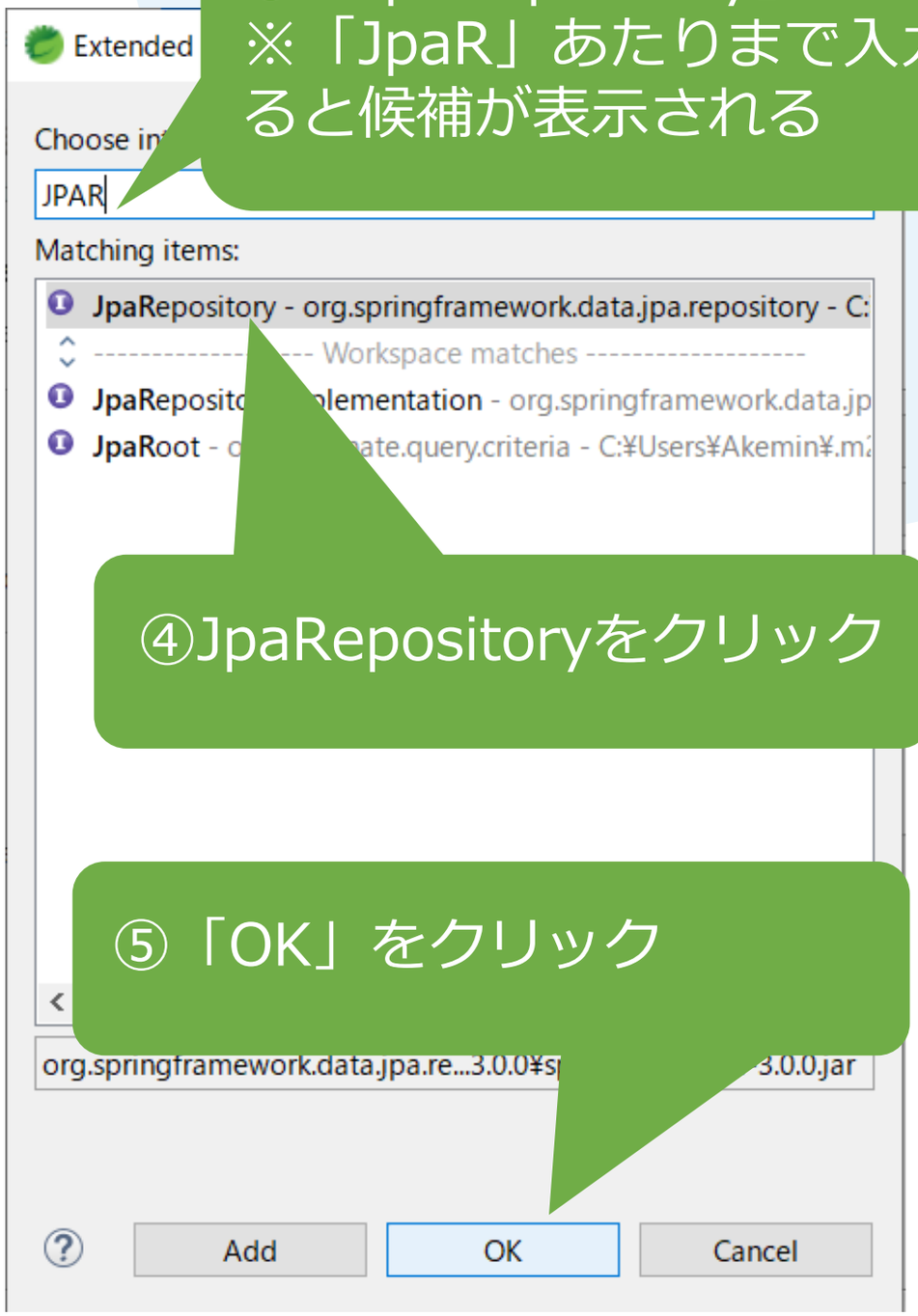
- UserDaoクラスの作成方法を説明します。



# UserDaoインターフェースの作成②

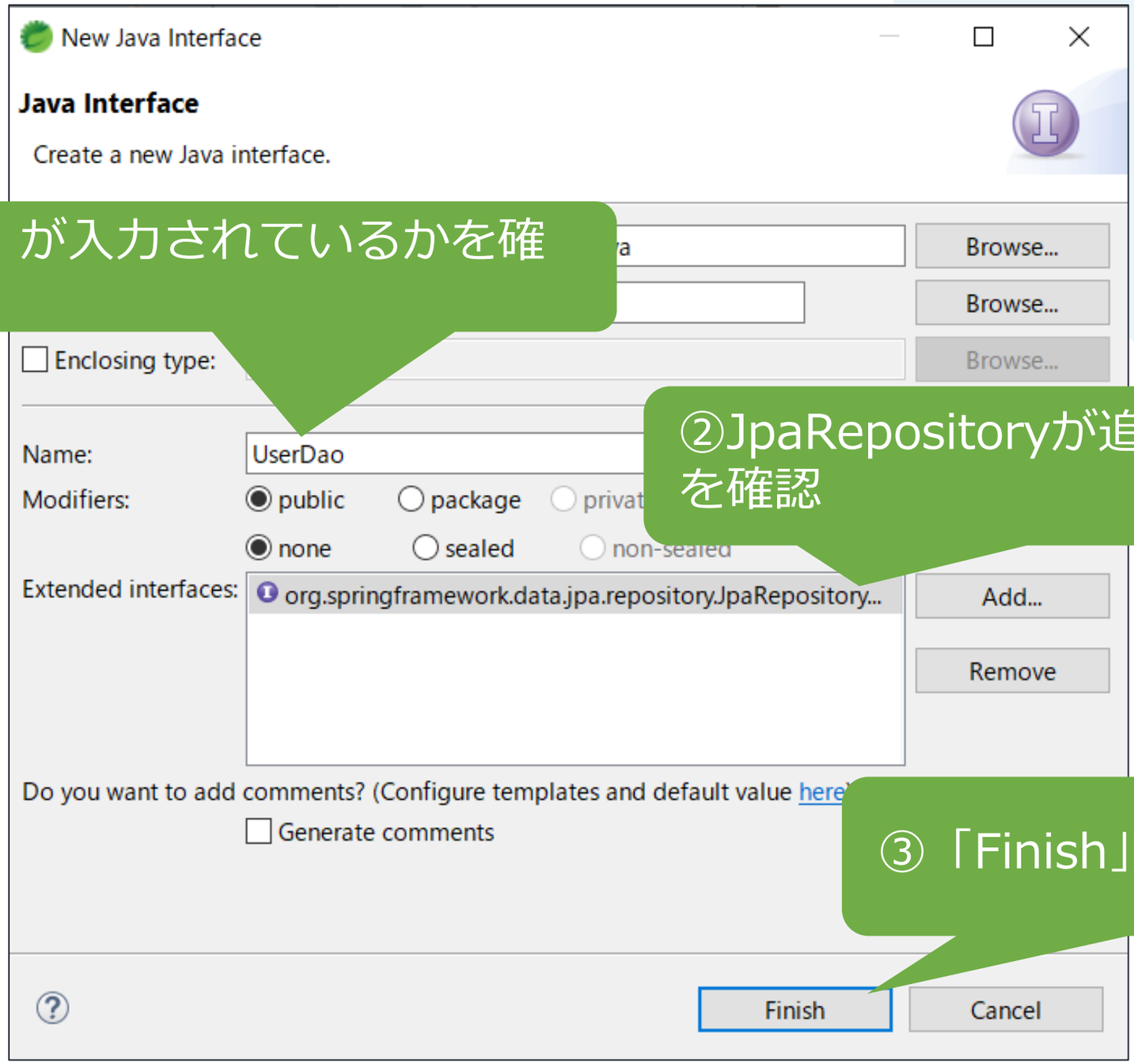
- UserDaoクラスの作成方法を説明します。



# 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

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

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

public interface UserDao extends JpaRepository<T, ID> {

}

```

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す

UserDao.java

```
1  package blog.example.model.dao;
2
3  import org.springframework.data.jpa.repository.JpaRepository;
4
5  public interface UserDao extends JpaRepository<T, ID> {
6
7  }
```

この部分のコンパイルエラーの修正

UserDao.java

```
1  package blog.example.model.dao;
2
3  import org.springframework.data.jpa.repository.JpaRepository;
4  import org.springframework.stereotype.Repository;
5
6  import blog.example.model.entity.UserEntity;
7
8  @Repository
9  public interface UserDao extends JpaRepository<UserEntity, Long> {
10
11 }
```

UserEntityのimportを忘れずに行う

# UserDaoインターフェースの作成⑥

- @Repositoryを追加しよう

UserDao.java

```
1  package blog.example.model.dao;
2
3  import org.springframework.data.jpa.repository.JpaRepository;
4  import org.springframework.stereotype.Repository;
5
6  import blog.example.model.entity.UserEntity;
7
8  @Repository
9  public interface UserDao extends JpaRepository<UserEntity, Long> {
10
11 }
```

@Repositoryアノテーションのimportを忘れずに行う。

@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

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

import java.util.List;

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

import blog.example.model.entity.UserEntity;

@Repository
public interface UserDao extends JpaRepository<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();
}
```

他にもメソッドがたくさんあります。下記のリンクを参考に他のJPAメソッドも
調べてみよう！
https://b1san-blog.com/post/spring/spring-jpa/

# UserDaoインターフェースの作成⑧

- 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

```
3       // 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
4       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。

# 目次

# UserServiceクラスの作成①

UserService.java

```java
package blog.example.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import blog.example.model.dao.UserDao;
import blog.example.model.entity.UserEntity;

@Service
public class UserService {
    @Autowired
    private UserDao userDao;
      //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;
        }
    }
    //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);
    }

}
```

UserServiceクラスを作成後、左側のソースコードを書き写してください。
次のページでソースコードを解説します。

# 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。

```
11 @Service
12 public class UserService {
13     @Autowired
14     private UserDao userDao;
```

@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

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。

UserService

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
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
はないです。

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？

⑤もちろんです！
UserServiceさん登録
処理が終わりましたよ！

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、
RegisterControllerさん
戻り値はtrueになります。

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。

Userさん

⑧無事に登録できたのか
な！ログインしてみよう！

# 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
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。

UserService

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
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。

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！

Userさん

⑥あ！すでに登録してた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！

# 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から渡
        //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;
        }
    }
```

UserDaoのメソッドの引数にuserEmailを渡して検を行い、結果をuserEntityに格納する

結果が見つかった場合は、DBに保存処理をかけるのと同時に、WebSecurityConfigのaddUserで、maneger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。

# 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。

```java
package blog.example.config;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.autoconfigure.security.servlet.PathRequest;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
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;

import blog.example.service.UserService;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig  {
      @Bean
        public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
            http.formLogin(login -> login
                    .loginPage("/admin/login")
                    .defaultSuccessUrl("/admin/blog/all",true)
                    .usernameParameter("userEmail") //リクエストパラメータのname属性を明示
                    .passwordParameter("password")
                    .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()
                    .anyRequest().authenticated()
            );
            return http.build();
        }
```

続き

```java
            );
            return http.build();
        }

    public static UserDetailsManager manager = null;
    @Autowired
    private UserService userService;
    @Bean
    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)
                .password(password)
                .roles("USER")
                .build());
    }


}
```

# WevSecurityCondigクラスの作成②

- ソースの説明

```java
package blog.example.config;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.autoconfigure.security.servlet.PathRequest;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
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;

import blog.example.service.UserService;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig  {
      @Bean
        public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
            http.formLogin(login -> login
                    .loginPage("/admin/login")
                    .defaultSuccessUrl("/admin/blog/all",true)
                    .usernameParameter("userEmail") //リクエストパラメータのname属性を明示
                    .passwordParameter("password")
                    .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()
                    .anyRequest().authenticated()
            );
            return http.build();
        }
```

カスタムなログイン画面を設定

**defaultSuccessUrl(String defaultSuccessUrl, boolean alwaysUse)**
ログイン成功後のURLを設定
ユーザーが認証前に保護されたページにアクセスしてない場合、またはaleaysUseがtrueの場合に、認証に成功した後にゆーざーが移動する場所を指定する。

ログインする際のリクエスト
パラメーターのname属性の設定

ログインに失敗した際のURLの設定

ログアウトに成功した際のURLの設定

URLごとの認可設定記述開始

ログイン無しでもアクセス可能なURLや
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;
    @Bean
    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)
                .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。

ユーザーの作成

# UserServiceクラスのimport

- import文を追加してください。

UserService.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
8
9  import blog.example.model.dao.UserDao;
10 import blog.example.model.entity.UserEntity;
11
12 @Service
13 public class UserService {
14     @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));
24             WebSecurityConfig.addUser(userEmail, password);
25             r
26         } els
27             r
28         }
29     }
30     //ユーザの一
31     public Li
32         retur
33     }
34
35     //idを見つ
36     //コントロー
37     public Us
38         retur
39     }
40
41 }
```

import 'WebSecurityConfig' (blog.example.config)
Create class 'WebSecurityConfig'
Create record 'WebSecurityConfig'
Create interface 'WebSecurityConfig'
Create constant 'WebSecurityConfig'
Create enum 'WebSecurityConfig'
Create local variable 'WebSecurityConfig'
Change to 'WebFluxConfig' (org.springframework.boot.autoconfigure.web.reactive.WebFlux
Change to 'WebSecurity' (org.springframework.security.config.annotation.web.builders)
Change to 'WebSecurityConfiguration' (org.springframework.security.config.annotation.web
Change to 'WebSecurityConfigurer' (org.springframework.security.config.annotation.web)

import文をクリック

UserService.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
8  import blog.example.config.WebSecurityConfig;
9  import blog.example.model.dao.UserDao;
10 import blog.example.model.entity.UserEntity;
11
12 @Service
13 public class UserService {
14     @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));
24             WebSecurityConfig.addUser(userEmail, password);
25             return true;
26         } else {
27             return false;
28         }
29     }
30     //ユーザの一覧を取得する
31     public List<UserEntity> getAllAccounts() {
32         return userDao.findAll();
33     }
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 }
```

importが追加されていることを確認

# 目次

# RegisterControllerの作成①

- 下記のソースを書き写してください。

```java
package blog.example.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import blog.example.service.UserService;

@Controller
@RequestMapping("/admin")
public class ReigisterController {
    @Autowired
    private UserService accountService;

    //登録画面を表示
    @GetMapping("/register")
    public String getRegisterPage() {
        return "admin_register.html";
    }
    //ユーザー情報の登録
    @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

```
11 @Controller
12 @RequestMapping("/admin")
13 public class RegisterController {
14     @Autowired
15     private UserService accountService;
16
17     //登録画面を表示
18     @GetMapping("/register")
19     public String getRegisterPage() {
20         return "admin_register.html";
21     }
```

@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

- ソースの説明

```
11 @Controller
12 @RequestMapping("/admin")
13 public class ReigisterController {
14     @Autowired
15     private UserService accountService;
16
17     //登録画面を表示
18     @GetMapping("/register")
19     public String getRegisterPage() {
20         return "admin_register.html";
21     }
22     //ユーザー情報の登録
23     @PostMapping("/register")
24     public String register(@RequestParam String userName,@RequestParam String userEmail,
25             @RequestParam String password) {
26         //もし保存をした場合には、login.htmlへ遷移する
27         if (accountService.createAccount(userName, userEmail,password)) {
28             return "redirect:/admin/login";
29         } else {
30             //そうでない場合には、register.htmlに遷移する
31             return "admin_register.html";
32
33         }
34
35     }
36 }
```

@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指示をしている。
Getメソッドによるリクエストに応答することを意味する。

@PostMapping
「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

```java
@Controller
@RequestMapping("/admin")
public class ReigisterController {
    @Autowired
    private UserService accountService;

    //登録画面を表示
    @GetMapping("/register")
    public String getRegisterPage() {
        return "admin_register.html";
    }
    //ユーザー情報の登録
    @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";

        }

    }
}
```

@RequestParam
このリクエストが
「userName」と「userEmail」と「password」の3つのパラメータを持つことを示す。
inputタグやtextareaタグ、selectタグ等にあるname属性に対応している。

# RegisterControllerの作成⑤

```html
<section class="register-section">
    <h2>管理者登録画面</h2>
    <form method="POST" th:action="@{/admin/register}">
        <div class="register-section-details">
            <div class="register-section-details_flex">
                <div>UserName</div>
                <input type="text" name="userName" value="">
            </div>
            <div class="register-section-details_flex">
                <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>

        </div>
    </form>
</section>
```

HTMLの各属性がControllerクラスの何のパラメータやアノテーションに対応しているかを必ず理解してください。

```java
    //ユーザー情報の登録
    @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⑥

- ソースの説明

```
11 @Controller
12 @RequestMapping("/admin")
13 public class ReigisterController {
14     @Autowired
15     private UserService accountService;
16
17     //登録画面を表示
18     @GetMapping("/register")
19     public String getRegisterPage() {
20         return "admin_register.html";
21     }
22     //ユーザー情報の登録
23     @PostMapping("/register")
24     public String register(@RequestParam String userName,@RequestParam String userEmail,
25             @RequestParam String password) {
26         //もし保存をした場合には、login.htmlへ遷移する
27         if (accountService.createAccount(userName, userEmail,password)) {
28             return "redirect:/admin/login";
29         } else {
30             //そうでない場合には、register.htmlに遷移する
31             return "admin_register.html";
32
33         }
34
35     }
36 }
```

登録画面を表示させる
admin_registerを表示する

@RequestParamでview側からパラメータを受け取る
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

# 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
https://fonts.google.com/specimen/Gochi+Hand?query=gochi

admin_register.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" th:href="@{/css/reset.css}">
    <link rel="stylesheet" th:href="@{/css/style.css}">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Kiwi+Maru:wght@300;400;500&display=swap" rel="stylesheet">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Gochi+Hand&family=Kiwi+Maru:wght@300;400;500&display=swap">
</head>
```

Thymeleafを使用してcssを読み込む

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

# 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を使用して画像を読み込む

```
<body>
<header>
        <!--スマートフォン-->
        <nav class="menu">
            <div class="logo">
                <a href="#"><img th:src="@{\images\logo.png}"></a>
            </div>
            <div class="menu-contents">
                <div class="menu-inner">
                    <ul>
                        <li class="menu__item"><a th:href="@{/login}">ログイン</a></li>
                    </ul>
                </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>
                <ul class="pc-list">
                <li class="pc-list__item"><a th:href="@{/login}">ログイン</a></li>
                </ul>
            </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。

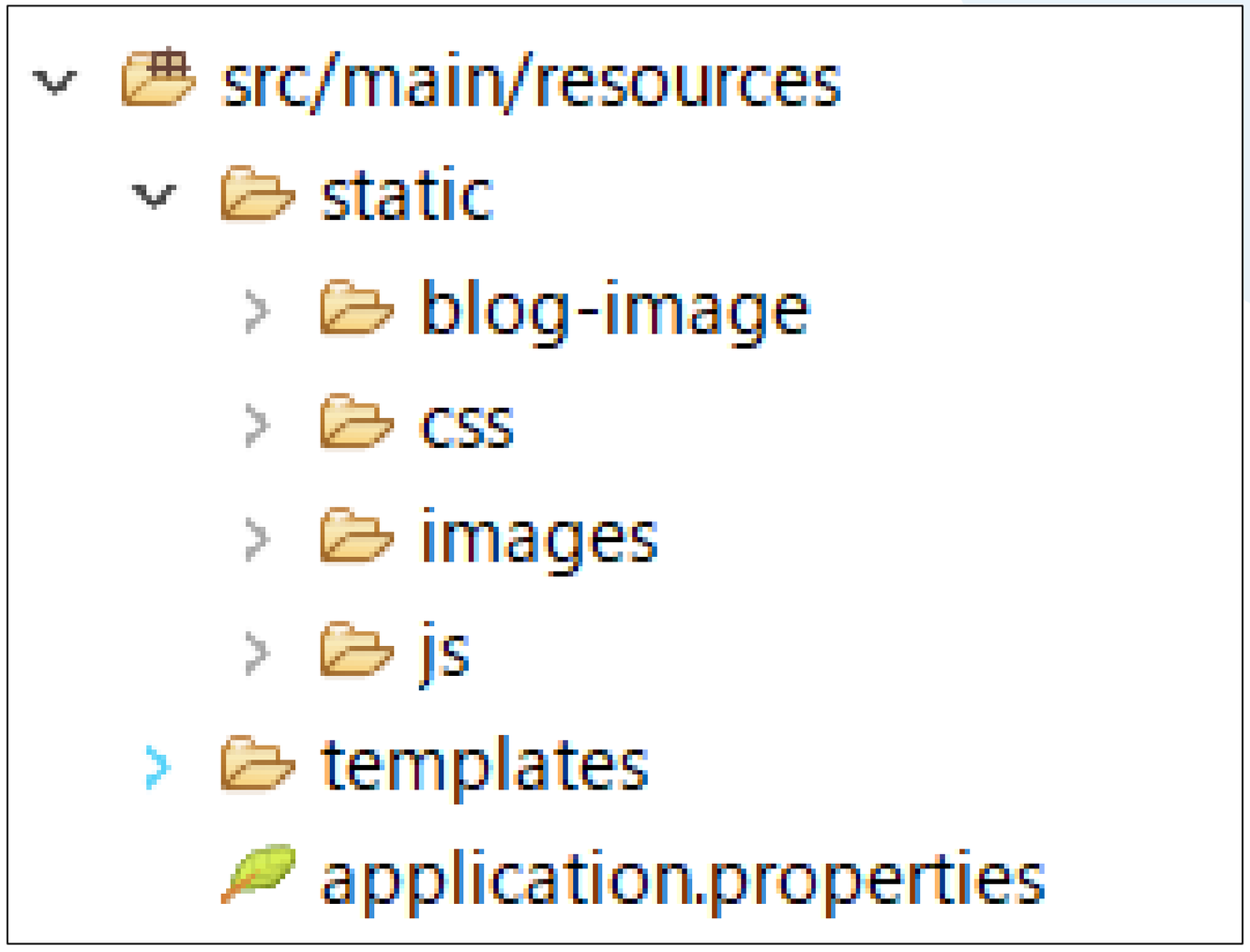
```
<main>
    <div class="main-inner">
        <section class="register-section">
            <h2>管理者登録画面</h2>
            <form method="POST" th:action="@{/admin/register}">
                <div class="register-section-details">
                    <div class="register-section-details_flex">
                        <div>UserName</div>
                        <input type="text" name="userName" value="">
                    </div>
                    <div class="register-section-details_flex">
                        <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>

                </div>
            </form>
        </section>
    </div>
</main>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"
        integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/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

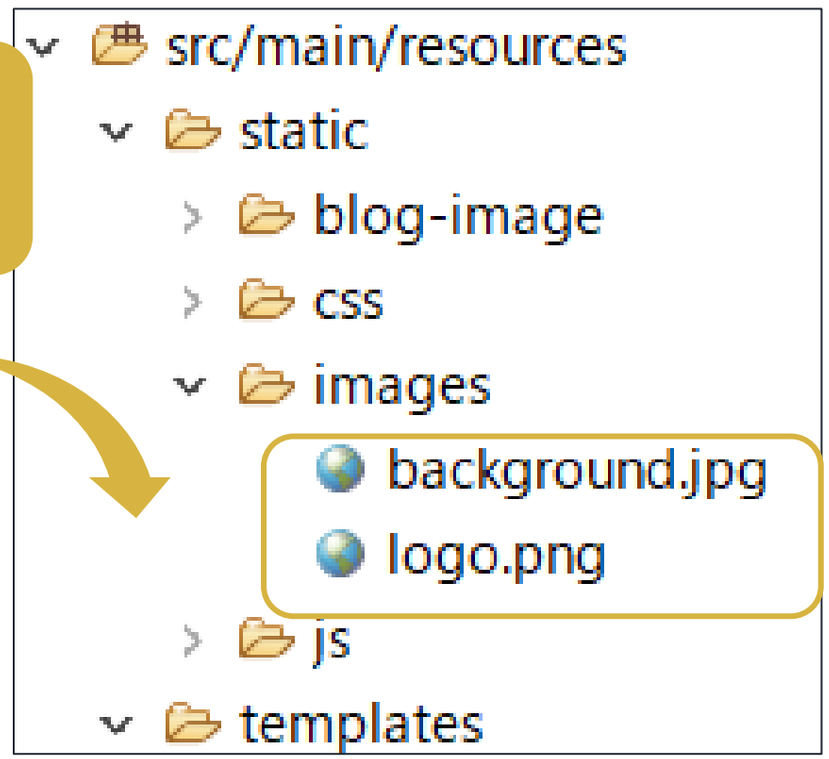
- staticフォルダーに指定したフォルダを作成してください



# 登録画面の作成⑤

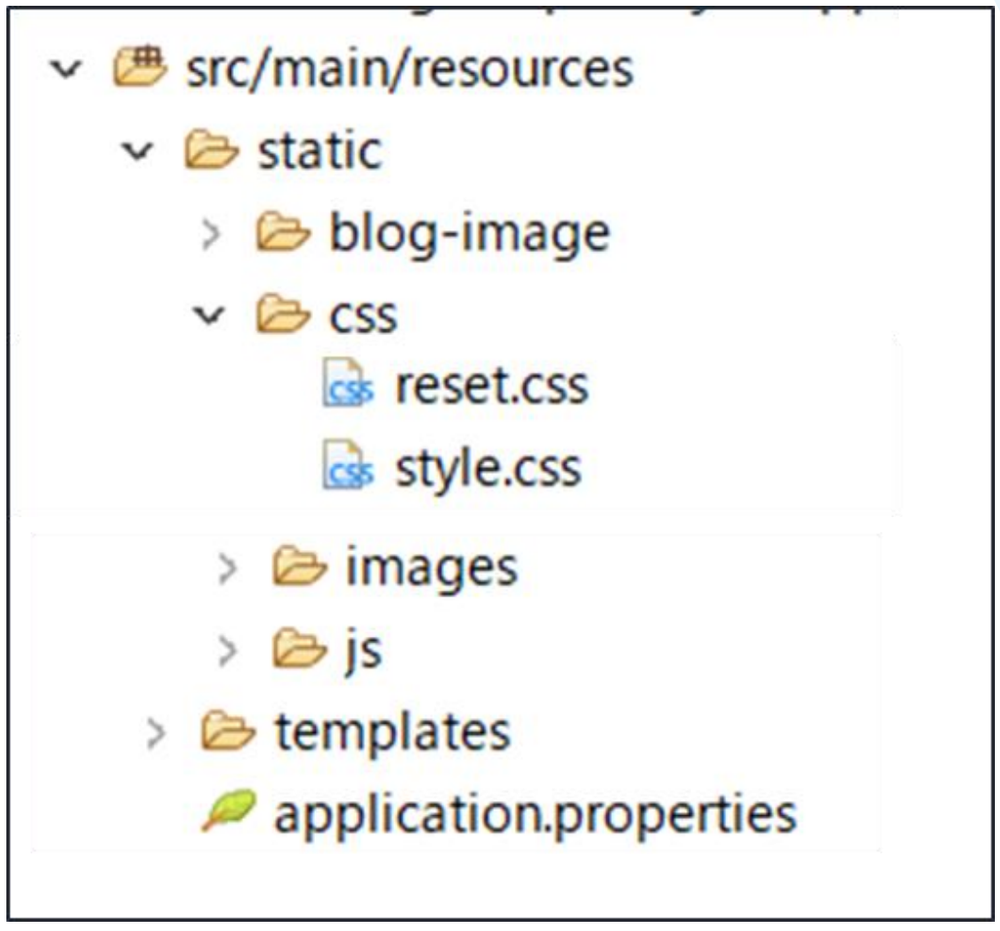
- staticフォルダーに指定したフォルダ画像を入れてください。


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/84e4acd264bd4cadba7df2a4d4ac4ba9

# 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。



# 登録画面の作成⑥

- reset.cssファイルに下記のソースを書き写してください。

reset.css

```css
/* http://meyerweb.com/eric/tools/css/reset/
   v2.0 | 20110126
   License: none (public domain)
*/

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    font: inherit;
    vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
    display: block;
}
body {
    line-height: 1;
}
ol, ul {
    list-style: none;
}
a{
    text-decoration: none;
}
blockquote, q {
    quotes: none;
}
```

続き

```css
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
    display: block;
}
body {
    line-height: 1;
}
ol, ul {
    list-style: none;
}
a{
    text-decoration: none;
}
blockquote, q {
    quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
    content: '';
    content: none;
}
table {
    border-collapse: collapse;
    border-spacing: 0;
}
```

# 登録画面の作成⑦

- style.cssファイルに下記のソースを書き写してください。

style.css

```css
/*========== 共通設定CSS ===============*/
img {
    width: 100%;
    height: auto;
    vertical-align: top;
}

main {
    margin-top: 6rem;
}


/*========== ヘッダーのCSS ===============*/
header {
    background-color: rgba(255, 255, 255, .7);
    height: 6rem;
    position: fixed;
    width: 100%;
    z-index: 100;
    top: 0;
    border-bottom: 1px solid black;
}

/*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*/
.logo, .menu-contents {
    width: 4rem;
}

.pc {
    display: none;
}

.menu {
    display: flex;
    justify-content: space-between;
    align-items: center;
    width: 90%;
    margin: 0 auto;
    padding-top: 0.5rem;
}
```

続き

```css
.menu {
    display: flex;
    justify-content: space-between;
    align-items: center;
    width: 90%;
    margin: 0 auto;
    padding-top: 0.5rem;
}

.logo img {
    width: 8rem;
    height: 1.6rem;
}

/*ボタン外側*/
.menu-toggle_btn {
    position: relative;
    /*ボタン内側の基点となるためrelativeを指定*/

    cursor: pointer;
    width: 4rem;
    height: 4rem;
    border-radius: 0.5rem;
}

/*ボタン内側*/
.menu-toggle_btn span {
    display: inline-block;
    transition: all .4s;
    /*アニメーションの設定*/
    position: absolute;
    left: 1rem;
    height: 3px;
    border-radius: 2px;
    background: #EC6015;
    width: 45%;
}

.menu-toggle_btn span:nth-of-type(1) {
    top: 1.5rem;
}
```

続き

```css
.menu-toggle_btn span:nth-of-type(1) {
    top: 1.5rem;
}

.menu-toggle_btn span:nth-of-type(2) {
    top: 2rem;
}

.menu-toggle_btn span:nth-of-type(3) {
    top: 2.5rem;
}

/*activeクラスが付与されると線が回転して×に*/

.menu-toggle_btn.active span:nth-of-type(1) {
    top: 1.5rem;
    left: 1.5rem;
    transform: translateY(6px) rotate(-45deg);
    width: 30%;
    z-index: 1;
}

.menu-toggle_btn.active span:nth-of-type(2) {
    opacity: 0;
    /*真ん中の線は透過*/
    z-index: 1;
}

.menu-toggle_btn.active span:nth-of-type(3) {
    top: 2.25rem;
    left: 1.5rem;
    transform: translateY(-6px) rotate(45deg);
    width: 30%;
    z-index: 1;
}
```

# 登録画面の作成⑧

- style.cssファイルに下記のソースを書き写してください。

```
107
108 /*----------------------------
109  * メニュー本体
110  *----------------------------*/
111 .menu-inner {
112     display: none;
113 }
114
115 .menu-inner.active {
116     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 {
130     margin-bottom: 3rem;
131     text-align: center;
132 }
133
134 .menu__item a {
135     background-color: #EC6015;
136     padding-top: 1rem;
137     padding-bottom: 1rem;
138     display: block;
139     color: #fff;
140     font-family: Kiwi Maru;
141     text-align: center;
142     text-decoration: none;
143     width: 10rem;
144 }
145
```

続き

```
144 }
145
146 .menu__item a:hover {
147     background-color: #ffefcc;
148     color: black;
149 }
150
151 /*-------------------------------------------------------*/
152 /*-------------------------------------------------------
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 /*----------------登録ボタン---------------------*/
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;
171     transition: 0.3s;
172     font-family: Kiwi Maru;
173
174 }
```

# 登録画面の作成⑨

- style.cssファイルに下記のソースを書き写してください。

```
174 }
175
176 .btn--orange:hover,
177 a.btn--orange:hover {
178     background-color: #ffefcc;
179     color: black;
180     transform: translateY(3px);
181     text-decoration: none;
182     box-shadow: 0 2px 0 #aaaaaa;
183 }
184
185 a.btn--radius {
186     border-radius: 3rem;
187 }
188
189 /*-------------------------------------------------*/
190 /*---------------編集ボタン---------------------*/
191 .edit_disable {
192     /*background-color:#71c7c8;*/
193     background-color: #a6a197;
194     color: #fff;
195     border: gray;
196     padding: 0.5rem 1rem;
197     box-shadow: 0 5px 0 #aaaaaa;
198     transition: 0.3s;
199 }
200
201 .edit {
202     /*background-color:#71c7c8;*/
203     background-color: #f59e0b;
204     color: #fff;
205     border: gray;
206     padding: 0.5rem 1rem;
207     box-shadow: 0 5px 0 #aaaaaa;
208     transition: 0.3s;
209 }
210
211 .edit:hover {
212     transform: translateY(3px);
213     text-decoration: none;
214     box-shadow: 0 2px 0 #aaaaaa;
215 }
```

続き

```
215 }
216
217
218 /*-------------------------------------------------*/
219 /*---------------戻るボタン---------------------*/
220 .back-btn {
221     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 {
230     transform: translateY(3px);
231     text-decoration: none;
232     box-shadow: 0 2px 0 #696868;
233 }
234 /*---------------削除ボタン---------------------*/
235 .delete {
236     background-color: #715494;
237     color: #fff;
238     border: gray;
239     padding: 0.5rem 1rem;
240     box-shadow: 0 5px 0 #aaaaaa;
241     transition: 0.3s;
242 }
243
244 .delete:hover {
245     transform: translateY(3px);
246     text-decoration: none;
247     box-shadow: 0 2px 0 #aaaaaa;
248 }
249
250 /*-------------------------------------------------*/
```

# 登録画面の作成⑩

- style.cssファイルに下記のソースを書き写してください。

```
/*-------------------------------------------------*/
/*---------------------------------------------------------
                        ジャンル一覧
---------------------------------------------------------*/
.genre {
    width: 90%;
    margin: 0 auto;
    padding-top: 2rem;
}

.genre-register__button {
    text-align: right;
}

.genre table {
    width: 100%;
    margin: 2rem auto;
    text-align: center;
    border: 1px solid gray;
}

.genre th, .genre td {
    border: 1px solid gray;
    text-align: center;
    padding: 1rem 0.5rem;

}

.genre th {
    background-color: #71c7c8;
    color: #fff;
}

/*---------------------------------------------------------
                        登録画面
---------------------------------------------------------*/
.register-section {
    width: 90%;
    margin: 0 auto;
    padding-top: 2rem;
}
```

続き

```
}

.register-section h2 {
    font-family: Kiwi Maru;
    font-size: 1.3rem;
    text-align: center;
    color: #594A4E;
    font-weight: bold;
}

.register-section-details {
    width: 100%;
    margin: 3rem auto;
}

svg {
    width: 30px;
    height: 30px;
    vertical-align: middle
}

.register-section-details_flex {
    display: flex;
    justify-content: center;
    gap: 0 1rem;
    align-items: center;
    margin: 3rem 0;
}
.register-section-details_flex>img {
    width: 252px;
}
.register-section-details_flex div:first-child {
    color: #594A4E;
    font-family: Kiwi Maru;
    width: 6rem;
    line-height: 1.5rem;
    font-size: 1rem;
}
```

# 登録画面の作成⑪

- style.cssファイルに下記のソースを書き写してください。

```
}
/*--登録のボタンの位置の設定--*/
#store_add_box{
  margin-left: 15rem;
}
/*--------------------------------*/
input[type="text"] {
    padding: 0.5rem 1rem;
    border-radius: 4px;
    border: none;
    box-shadow: 0 0 0 1px #ccc inset;
    appearance: none;
    -webkit-appearance: none;
    -moz-appearance: none;
    font-size: 1rem;
    color: #594A4E;
    font-family: Kiwi Maru;
    width: 250px;
}

input[type="text"]:focus {
    outline: 0;
    box-shadow: 0 0 0 2px #71c7c8 inset;
}

input[type="time"] {
    padding: 0.5rem 1rem;
    border-radius: 4px;
    border: none;
    box-shadow: 0 0 0 1px #ccc inset;
    appearance: none;
    -webkit-appearance: none;
    -moz-appearance: none;
    font-size: 1rem;
    color: #594A4E;
    font-family: Kiwi Maru;
    width: 250px;
}

input[type="time"]:focus {
    outline: 0;
    box-shadow: 0 0 0 2px #71c7c8 inset;
}
```

続き

```
}

textarea {
    resize: vertical;
    padding: 0.5rem 1rem;
    border-radius: 4px;
    border: none;
    box-shadow: 0 0 0 1px #ccc inset;
    appearance: none;
    -webkit-appearance: none;
    -moz-appearance: none;
    font-size: 1rem;
    color: #594A4E;
    font-family: Kiwi Maru;
    width: 250px;
}

textarea:focus {
    outline: 0;
    box-shadow: 0 0 0 2px #71c7c8 inset;
}

select {
    padding: 0.5rem 1rem;
    border-radius: 4px;
    border: none;
    box-shadow: 0 0 0 1px #ccc inset;
    font-size: 1rem;
    color: #594A4E;
    font-family: Kiwi Maru;
    width: 250px;
}

select:focus {
    outline: 0;
    box-shadow: 0 0 0 2px #71c7c8 inset;
}
```

# 登録画面の作成⑫

- style.cssファイルに下記のソースを書き写してください。

```
}

.checkbox__flex {
    width: 250px;
}

input[type="checkbox"] {
    margin-right: 1rem;
    margin-bottom: 1rem;
}

.photo_img {
    width: 250px;
}

.photo_img img {
    width: 80%;
    box-shadow: 0 0 0 1px #ccc inset;
}
.photo_img video {
    display: block;
    width: 80%;
    box-shadow: 0 0 0 1px #ccc inset;
}

/*------------------------------------------------------
                    一覧画面
------------------------------------------------------*/

.all-view-section {
    width: 90%;
    margin: 0 auto;
    padding-top: 2rem;
}

.all-view-section h2 {
    font-family: Kiwi Maru;
    font-size: 1.3rem;
    text-align: center;
    color: #594A4E;
    font-weight: bold;
}
```

続き

```
}

.all-view-register__button {
    text-align: right;
    margin: 3rem 0;
}

.all-view-article {
    border: 1px solid #594A4E;
    margin-bottom: 2rem;
    box-shadow: 0 5px 0 #aaaaaa;

}

.all-view-colum {
    position: relative;
    font-family: Kiwi Maru;
}

.all-view-colum img {
    background-color: #fff;
    border-bottom: 1px solid #594A4E;
}

.all-view__category {
    position: absolute;
    z-index: 1;
    left: 0;
    background-color: #594A4E;
    color: #fff;
    padding: 0.5rem 1rem;
}

.message {
    box-sizing: border-box;
    width: 90%;
    padding: 8px 19px;
    margin: 2em auto;
    color: #2c2c2f;
    background: #fff;
    border-top: solid 5px #594A4E;
    border-bottom: solid 5px #594A4E;
}
```

# 登録画面の作成⑬

- style.cssファイルに下記のソースを模写してください。

```
488 }
489
490 .message p {
491     margin: 0;
492     padding: 0;
493     word-wrap: break-word;
494 }
495
496 /*==================================
497                 カテゴリー設定
498 ==================================*/
499 .colum__box {
500     padding: 1rem;
501     background-color: #fff;
502     color: #594A4E;
503     font-family: Kiwi Maru;
504     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;
517     font-size: 1rem;
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 }
```

続き

```
529 }
530 @media screen and (min-width: 768px) {
531
532     /*----------------ヘッダー----------------*/
533     .pc {
534         display: block;
535         padding-top: 1.5rem;
536     }
537
538     .menu {
539         display: none;
540     }
541
542     .pc-inner {
543         display: flex;
544         justify-content: space-between;
545         width: 90%;
546         margin: 0 auto;
547         padding-top: 1rem;
548     }
549
550     .pc-logo {
551         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⑭

- 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。

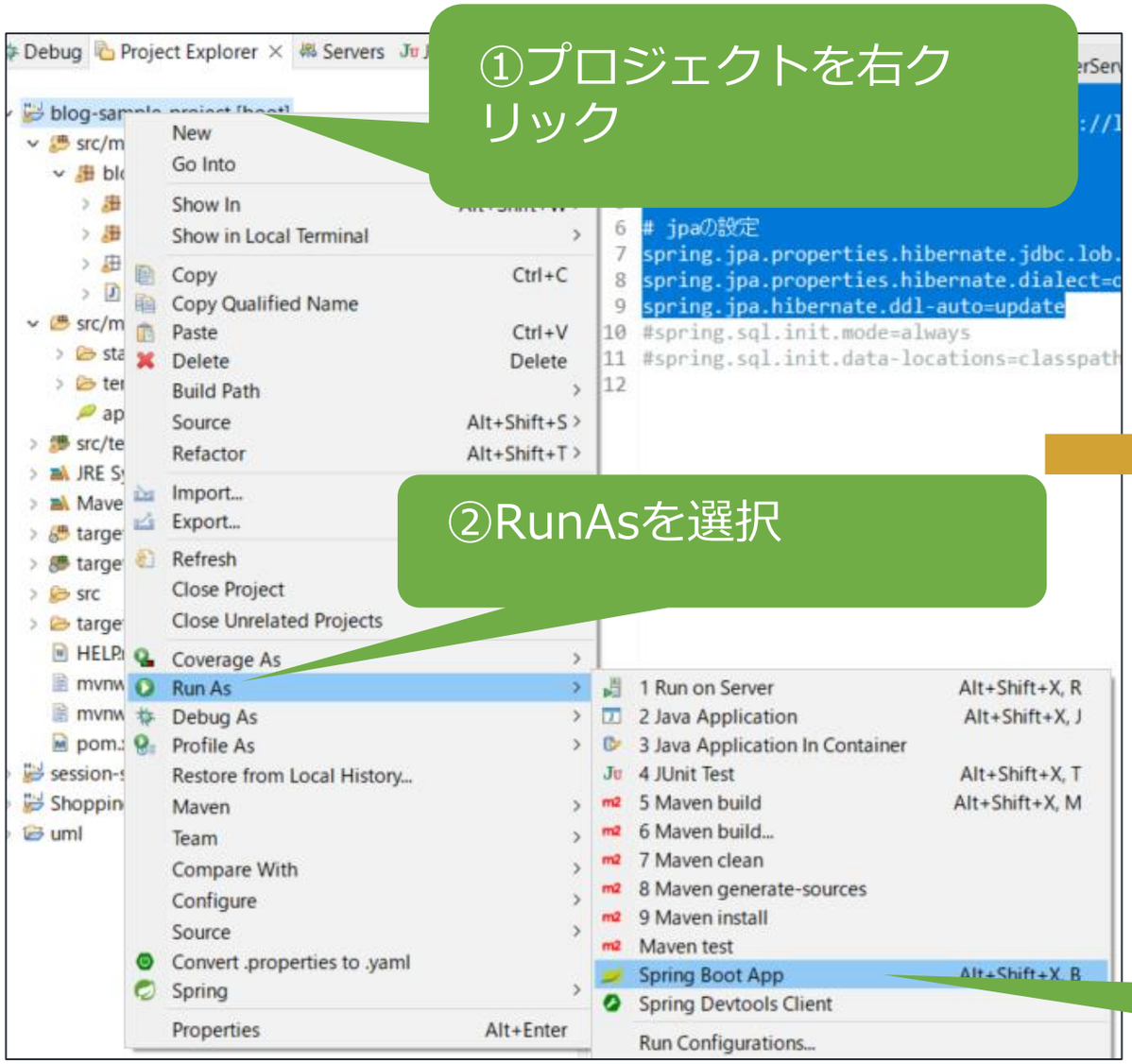
```
566     }
567
568     .pc-list {
569         display: flex;
570         align-items: center;
571         gap: 0 1.5rem;
572     }
573
574     .pc-list a {
575         font-size: 1.2rem;
576         font-weight: 600;
577         color: #594A4E;
578         font-family: Kiwi Maru;
579     }
580
581     .pc-list a:hover {
582         color: #EC6015;
583     }
584
585     /*-------------------------------------------------------
586                             一覧画面
587     -------------------------------------------------------*/
588
589 .all-view-section {
590     width: 800px;
591     margin: 0 auto;
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 }
```

続き

```
600 }
601 .all-view-article {
602     width: 33%;
603     box-sizing: border-box;
604
605 }
606
607 .colum__box p:nth-child(1) {
608     text-align: left;
609     font-size: 1rem;
610 }
611
612 .colum__box p:nth-child(2) {
613     text-align: left;
614     padding: 1.2rem 0;
615     font-size: 1.2rem;
616 }
617
618 .colum__box p:nth-child(3) {
619     text-align: right;
620     font-size: 1rem;
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。




# 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

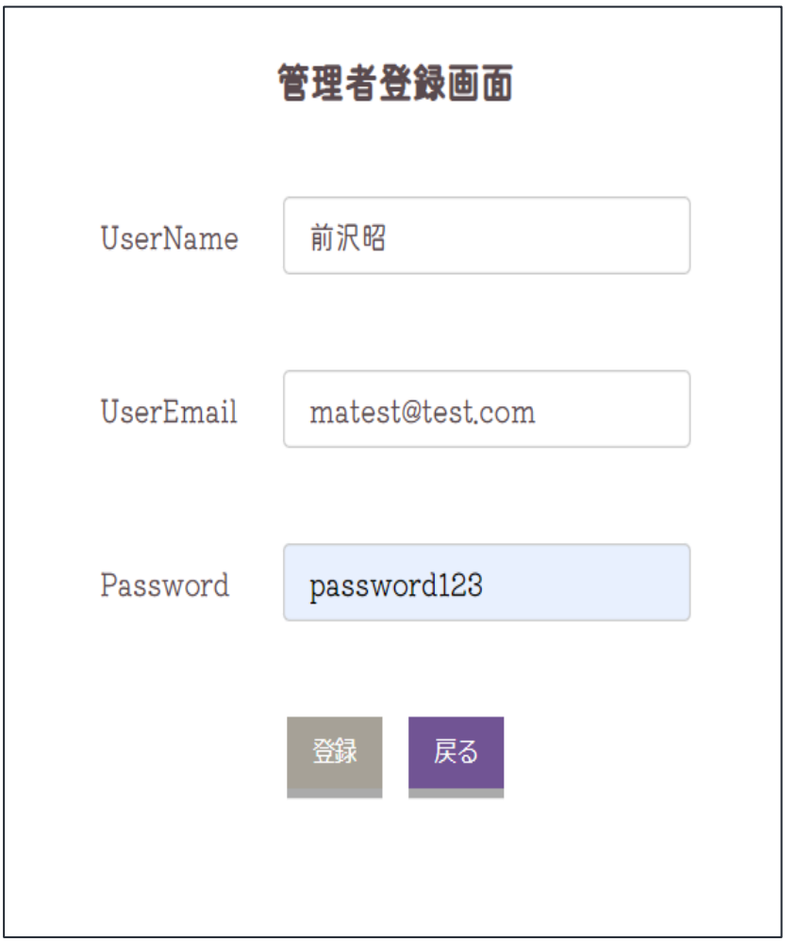
http://localhost:8080/admin/register

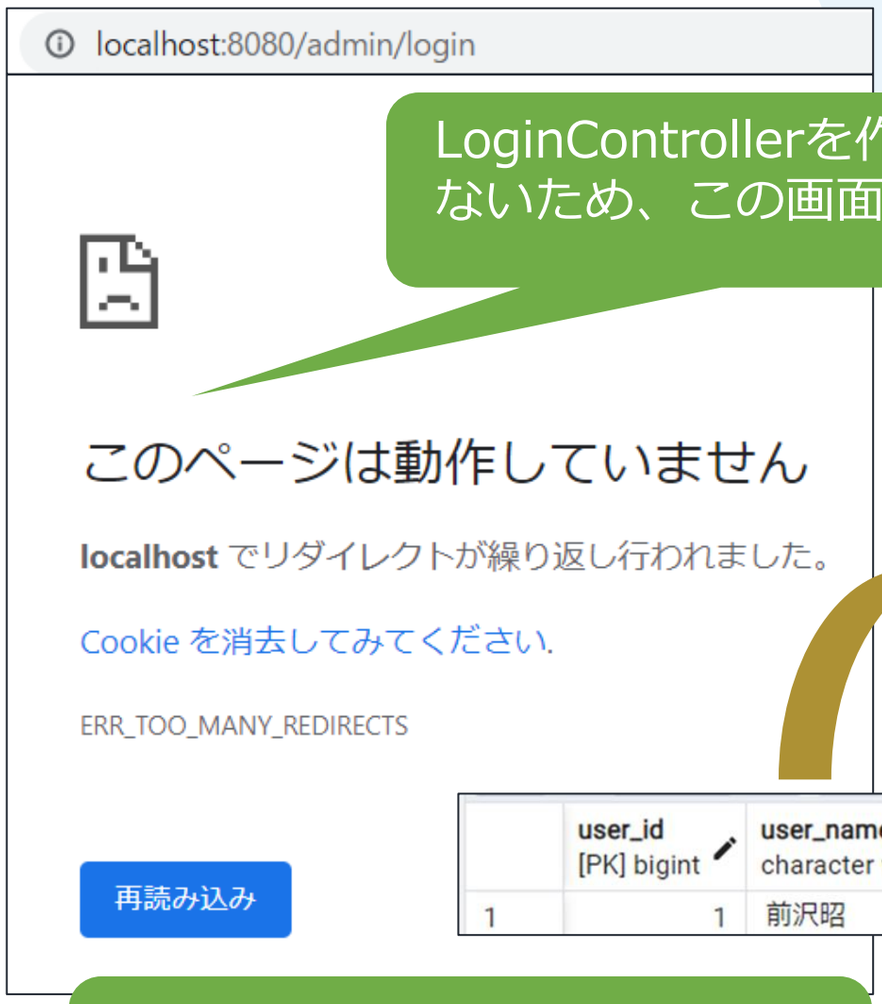

使用するブラウザーは
「chrome」にしてください。
Baiduは使用を禁止します。

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。




LoginControllerを作成していないため、この画面になる。

| | 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 |

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 Your Career.*

# THANK YOU!