



Woodland
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



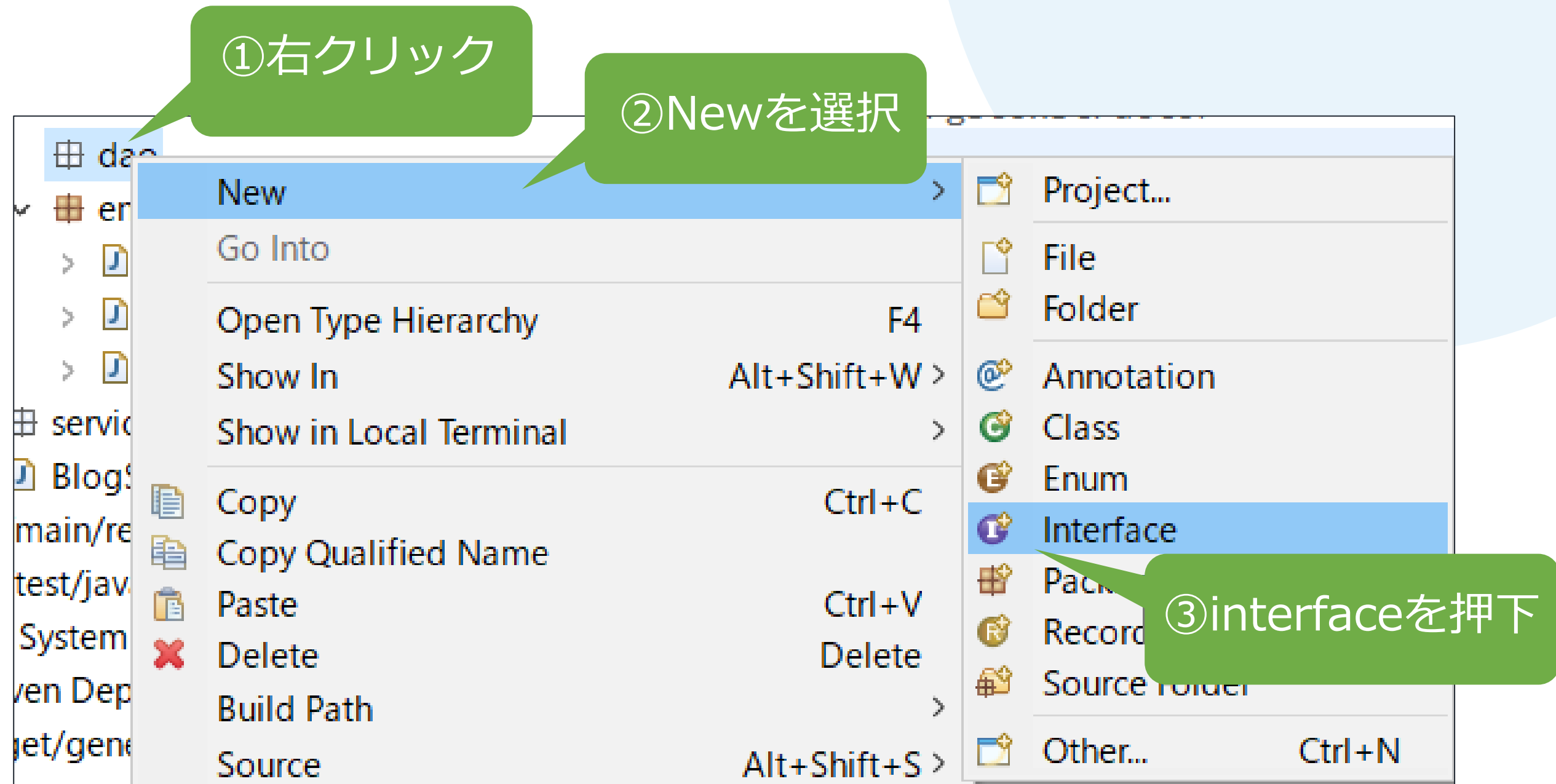
Shape Your Future

目次

- 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

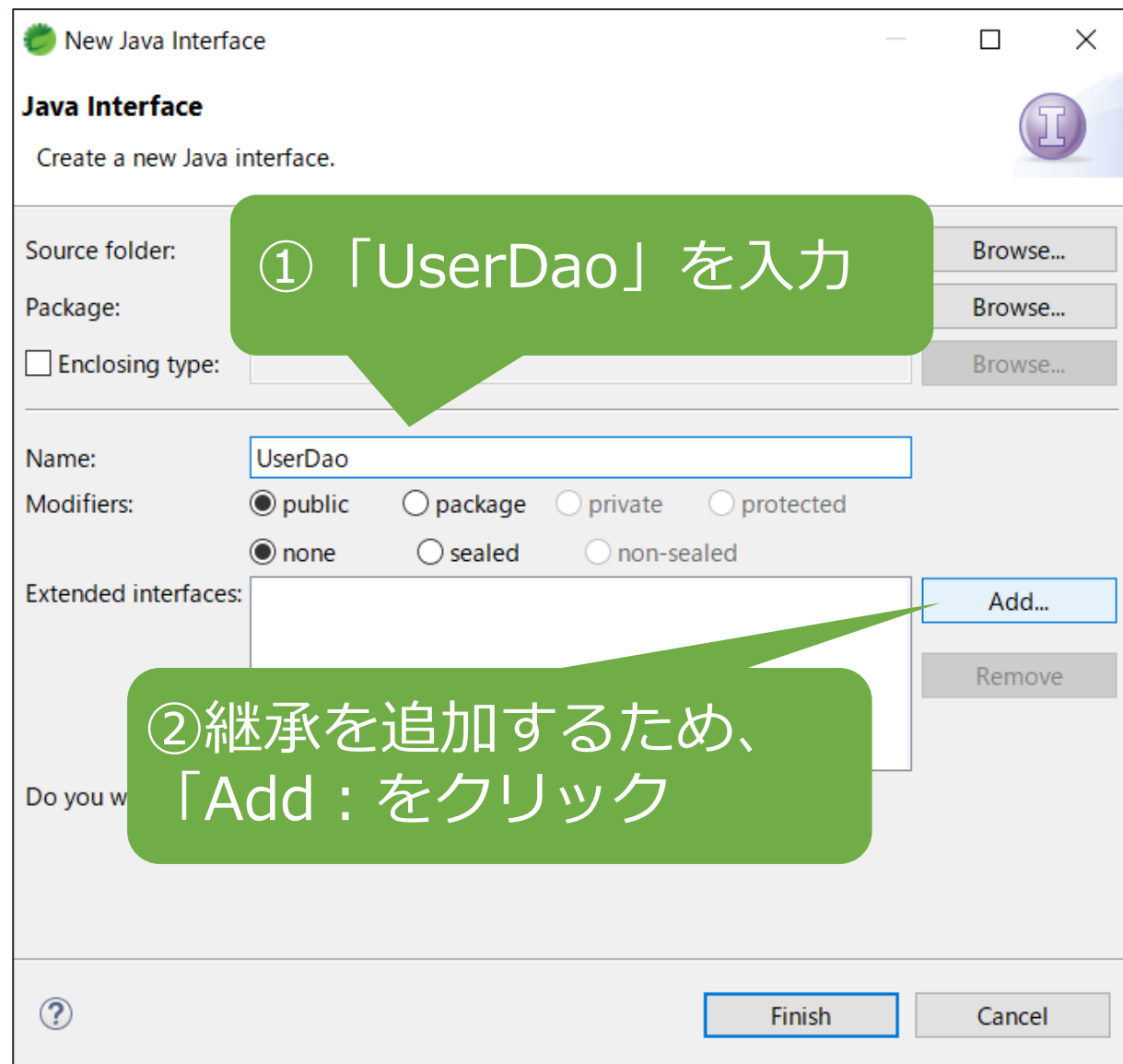
UserDaoインターフェースの作成①

- 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。

New Java Interface

Java Interface
Create a new Java interface.

☐ Enclosing type:

Name: UserDao

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

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

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

Finish **Cancel**

① 「UserDao」が入力されているかを確認

② JpaRepositoryが追加されていることを確認

③ 「Finish」をクリック

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
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 public interface UserDao extends JpaRepository<T, 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す

```

9 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

```

1 UserDao.java ×
2
3 package blog.example.model.dao;
4
5 import org.springframework.data.jpa.repository.JpaRepository;
6 import org.springframework.stereotype.Repository;
7 import blog.example.model.entity.UserEntity;
8
9 @Repository
10 public interface UserDao extends JpaRepository<UserEntity, Long> {
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あることをSpring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 ×
1 package blog.example.model.dao;
2
3 import java.util.List;
4
5 import org.springframework.data.jpa.repository.JpaRepository;
6 import org.springframework.stereotype.Repository;
7
8 import blog.example.model.entity.UserEntity;
9
10 @Repository
11 public interface UserDao extends JpaRepository<UserEntity, Long> {
12     |
13     // 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
14     UserEntity findByUserEmail(String userEmail);
15
16     // UserServiceから渡されるユーザ情報を基にDBへ保存する
17     UserEntity save(UserEntity userEntity);
18
19     // ユーザ情報一覧を取得
20     List<UserEntity> findAll();
21 }
```

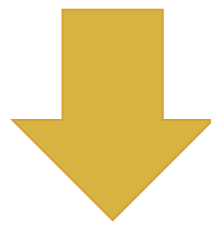
他にもメソッドがたくさんあります。下記のリンクを参考に他のJPAメソッドも調べてみよう！

<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=?
```



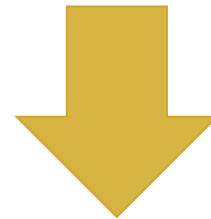
上のSQL文を省略している

```
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);
```

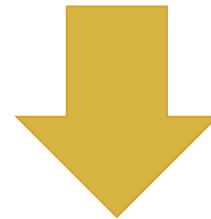


saveメソッドは要注意
insertとupdateの両方の役割を持っている
1：更新対象のキーでセレクト
2：同じキーのレコードがあるかをチェック
 1：あったらupdate
 2：なかったらinsert
という順で処理をしている。

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

- 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

UserServiceクラスの作成①

```
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.model.dao.UserDao;
9 import blog.example.model.entity.UserEntity;
10
11 @Service
12 public class UserService {
13     @Autowired
14     private UserDao userDao;
15     //ユーザの情報を保存する
16     public boolean createAccount(String userName, String userEmail, String password) {
17         //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
18         UserEntity userEntity = userDao.findByUserEmail(userEmail);
19         //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
20         //なかった場合には、保存
21         if (userEntity == null) {
22             userDao.save(new UserEntity(userName, userEmail, password));
23             WebSecurityConfig.addUser(userEmail, password);
24             return true;
25         } else {
26             return false;
27         }
28     }
29     //ユーザの一覧を取得する
30     public List<UserEntity> getAllAccounts() {
31         return userDao.findAll();
32     }
33
34     //idを見つけるために
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。

```
10
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？



Userさん



⑧無事に登録できたのか！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、RegisterControllerさん戻り値はtrueになります。

⑤もちろんです！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。

Userさん



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から渡される
    //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、manager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Controller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

- 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

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;

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

```
);
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

```
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.UserDetailsService;
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alwaysUseがtrueの場合に、認証に成功した後にユーザ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         }
26     }
27
28     //ユーザの一覧を取得する
29     public List<UserEntity> getAllAccounts() {
30         return userDao.findAll();
31     }
32
33     //idを見つけるために
34     //コントローラーでわたってきたuserEmailを基にしてDBを検索
35     public UserEntity selectById(String userEmail) {
36         return userDao.findByUserEmail(userEmail);
37     }
38
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
31     //ユーザの一覧を取得する
32     public List<UserEntity> getAllAccounts() {
33         return userDao.findAll();
34     }
35
36     //idを見つけるために
37     //コントローラーでわたってきたuserEmailを基にしてDBを検索
38     public UserEntity selectById(String userEmail) {
39         return userDao.findByUserEmail(userEmail);
40     }
41
42 }

```

importが追加されていることを確認

目次

- 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

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;
2
3 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 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     }
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 }

```


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。
@GetMapping("/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>」にアクセスするときのHTTPリクエストに回答した時、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回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

```

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 }

```

@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

```
<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。

```
26 //ユーザー情報の登録
27 @PostMapping("/register")
28 public String register(@RequestParam String userName, @RequestParam String userEmail,
29   @RequestParam String password) {
30   //もし保存をした場合には、login.htmlへ遷移する
31   if (accountService.createAccount(userName, userEmail, password)) {
32     return "redirect:/admin/login";
33   } else {
34     //そうでない場合には、register.htmlに遷移する
35     return "admin_register.html";
36   }
37 }
38
39 }
```


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

- 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

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>

Thymeleafを使用してcssを読み込む

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

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="#"></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=""></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。

Thymeleafを使用して
action先を設定

Jquery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/>

```
<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+30JU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4=" crossorigin="anonymous"></script>
<script th:src="@{/js/common.js}"></script>
</body>
</html>
```

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

```
▼ 📁 src/main/resources
  ▼ 📁 static
    > 📁 blog-image
    > 📁 css
    > 📁 images
    > 📁 js
  > 📁 templates
  🍃 application.properties
```

登録画面の作成⑤

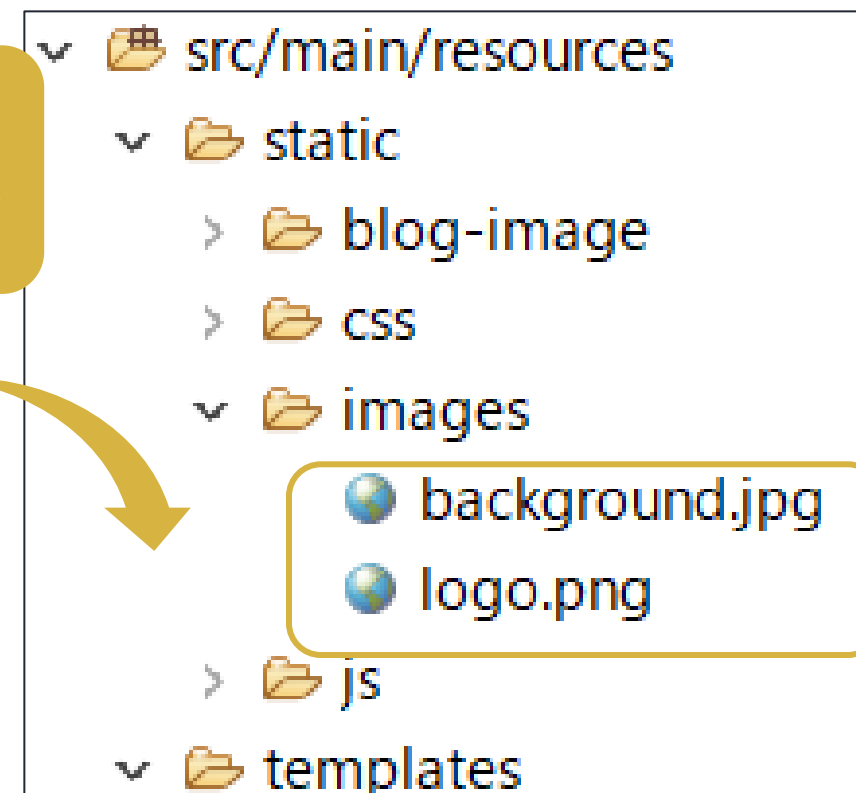
- staticフォルダーに指定したフォルダ画像を入れてください。



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/84e4acd264bd4cadba7df2a4d4ac4ba9>

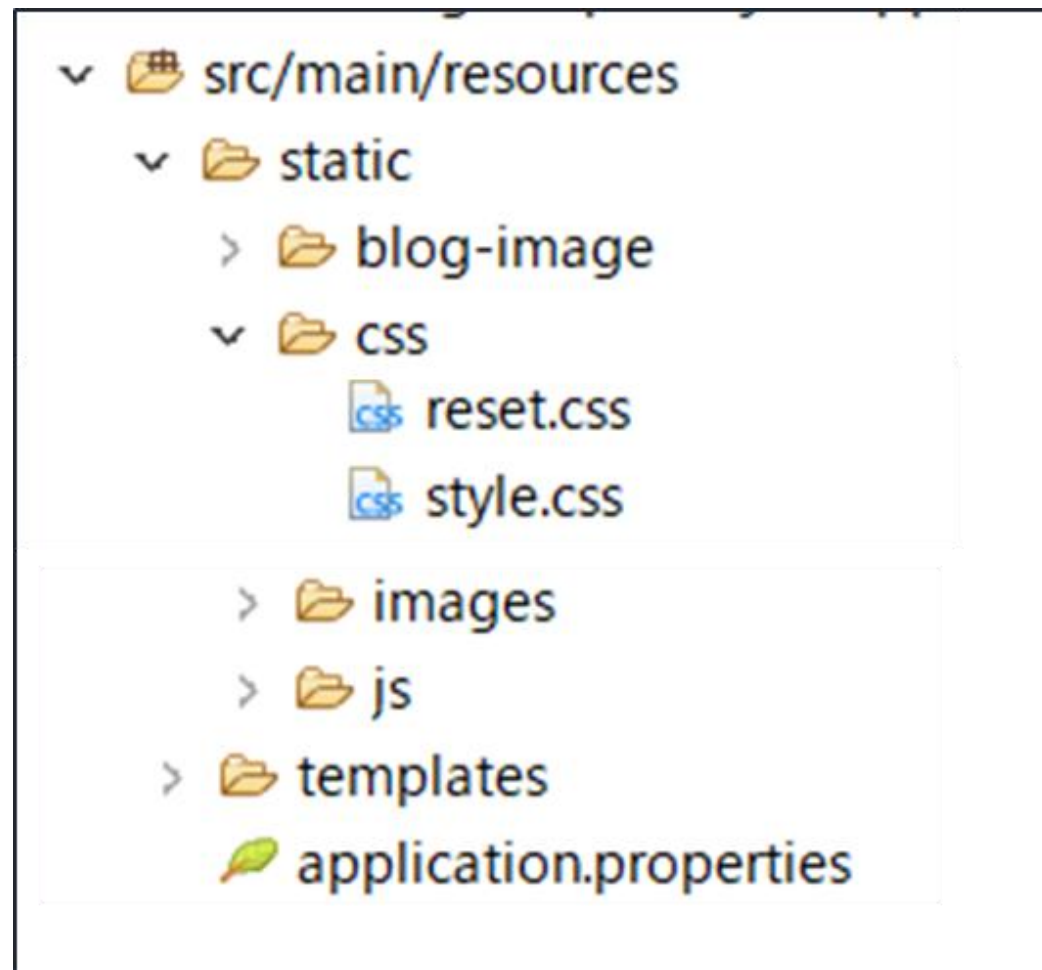
images.zipを解凍をしてください。

Drag & Drop
(ドラック&ドロップ)



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 ×
1 /* http://meyerweb.com/eric/tools/css/reset/
2    v2.0 | 20110126
3    License: none (public domain)
4 */
5
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;
20     padding: 0;
21     border: 0;
22     font-size: 100%;
23     font: inherit;
24     vertical-align: baseline;
25 }
26 /* HTML5 display-role reset for older browsers */
27 article, aside, details, figcaption, figure,
28 footer, header, hgroup, menu, nav, section {
29     display: block;
30 }
31 body {
32     line-height: 1;
33 }
34 ol, ul {
35     list-style: none;
36 }
37 a {
38     text-decoration: none;
39 }
40 blockquote, q {
41     quotes: none;
42 }
```

続き

```
25 }
26 /* HTML5 display-role reset for older browsers */
27 article, aside, details, figcaption, figure,
28 footer, header, hgroup, menu, nav, section {
29     display: block;
30 }
31 body {
32     line-height: 1;
33 }
34 ol, ul {
35     list-style: none;
36 }
37 a {
38     text-decoration: none;
39 }
40 blockquote, q {
41     quotes: none;
42 }
43 blockquote:before, blockquote:after,
44 q:before, q:after {
45     content: '';
46     content: none;
47 }
48 table {
49     border-collapse: collapse;
50     border-spacing: 0;
51 }
```

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。

```
1 /*===== 共通設定CSS =====*/
2 img {
3   width: 100%;
4   height: auto;
5   vertical-align: top;
6 }
7
8 main {
9   margin-top: 6rem;
10 }
11
12
13 /*===== ヘッダーのCSS =====*/
14 header {
15   background-color: rgba(255, 255, 255, .7);
16   height: 6rem;
17   position: fixed;
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 {
26   width: 4rem;
27 }
28
29 .pc {
30   display: none;
31 }
32
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 }
```

続き

```
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 {
49   position: relative;
50   /*ボタン内側の基点となるためrelativeを指定*/
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 {
60   display: inline-block;
61   transition: all .4s;
62   /*アニメーションの設定*/
63   position: absolute;
64   left: 1rem;
65   height: 3px;
66   border-radius: 2px;
67   background: #EC6015;
68   width: 45%;
69 }
70
71 .menu-toggle_btn span:nth-of-type(1) {
72   top: 1.5rem;
73 }
```

続き

```
71 .menu-toggle_btn span:nth-of-type(1) {
72   top: 1.5rem;
73 }
74
75 .menu-toggle_btn span:nth-of-type(2) {
76   top: 2rem;
77 }
78
79 .menu-toggle_btn span:nth-of-type(3) {
80   top: 2.5rem;
81 }
82
83 /*activeクラスが付与されると線が回転して×に*/
84
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 }
92
93 .menu-toggle_btn.active span:nth-of-type(2) {
94   opacity: 0;
95   /*真ん中の線は透過*/
96   z-index: 1;
97 }
98
99 .menu-toggle_btn.active span:nth-of-type(3) {
100   top: 2.25rem;
101   left: 1.5rem;
102   transform: translateY(-6px) rotate(45deg);
103   width: 30%;
104   z-index: 1;
105 }
106
107
```

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。

```
250 /*-----*/
251 /*-----ジャンル一覧-----*/
252
253
254 .genre {
255     width: 90%;
256     margin: 0 auto;
257     padding-top: 2rem;
258 }
259
260 .genre-register__button {
261     text-align: right;
262 }
263
264 .genre table {
265     width: 100%;
266     margin: 2rem auto;
267     text-align: center;
268     border: 1px solid #gray;
269 }
270
271 .genre th, .genre td {
272     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
283 /*-----登録画面-----*/
284
285
286 .register-section {
287     width: 90%;
288     margin: 0 auto;
289     padding-top: 2rem;
290 }
```

続き

```
290 }
291
292 .register-section h2 {
293     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 {
301     width: 100%;
302     margin: 3rem auto;
303 }
304
305 svg {
306     width: 30px;
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 }
321 .register-section-details_flex div:first-child {
322     color: #594A4E;
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⑪

- style.cssファイルに下記のソースを書き写してください。

```

327 }
328 /*--登録のボタンの位置の設定--*/
329 #store_add_box{
330     margin-left: 15rem;
331 }
332 /*-----*/
333 input[type="text"] {
334     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 }
346
347 input[type="text"]:focus {
348     outline: 0;
349     box-shadow: 0 0 0 2px #71c7c8 inset;
350 }
351
352 input[type="time"] {
353     padding: 0.5rem 1rem;
354     border-radius: 4px;
355     border: none;
356     box-shadow: 0 0 0 1px #ccc inset;
357     appearance: none;
358     -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 }

```

続き

```

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 {
392     padding: 0.5rem 1rem;
393     border-radius: 4px;
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;
405 }

```


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。

```

405 }
406
407 .checkbox__flex {
408     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 {
421     width: 80%;
422     box-shadow: 0 0 0 1px #ccc inset;
423 }
424 .photo_img video {
425     display: block;
426     width: 80%;
427     box-shadow: 0 0 0 1px #ccc inset;
428 }
429
430 /*-----
431                一覧画面
432 -----*/
433
434 .all-view-section {
435     width: 90%;
436     margin: 0 auto;
437     padding-top: 2rem;
438 }
439
440 .all-view-section h2 {
441     font-family: Kiwi Maru;
442     font-size: 1.3rem;
443     text-align: center;
444     color: #594A4E;
445     font-weight: bold;
446 }

```

続き

```

446 }
447
448 .all-view-register_button {
449     text-align: right;
450     margin: 3rem 0;
451 }
452
453 .all-view-article {
454     border: 1px solid #594A4E;
455     margin-bottom: 2rem;
456     box-shadow: 0 5px 0 #aaaaaa;
457 }
458
459 .all-view-colum {
460     position: relative;
461     font-family: Kiwi Maru;
462 }
463
464 .all-view-colum img {
465     background-color: #fff;
466     border-bottom: 1px solid #594A4E;
467 }
468
469 .all-view_category {
470     position: absolute;
471     z-index: 1;
472     left: 0;
473     background-color: #594A4E;
474     color: #fff;
475     padding: 0.5rem 1rem;
476 }
477
478 .message {
479     box-sizing: border-box;
480     width: 90%;
481     padding: 8px 19px;
482     margin: 2em auto;
483     color: #2c2c2f;
484     background: #fff;
485     border-top: solid 5px #594A4E;
486     border-bottom: solid 5px #594A4E;
487 }
488
489

```

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     カテゴリ設定
498 =====*/
499 .column_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 .column_box_noflex{
510     padding: 1rem;
511     background-color: #fff;
512     color: #594A4E;
513     font-family: Kiwi Maru;
514 }
515 .column_box_noflex p:nth-child(1) {
516     text-align: left;
517     font-size: 1rem;
518 }
519
520 .column_box_noflex p:nth-child(2) {
521     text-align: left;
522     padding: 1.2rem 0;
523     font-size: 1.2rem;
524 }
525
526 .column_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 .column__box p:nth-child(1) {
608     text-align: left;
609     font-size: 1rem;
610 }
611
612 .column__box p:nth-child(2) {
613     text-align: left;
614     padding: 1.2rem 0;
615     font-size: 1.2rem;
616 }
617
618 .column__box p:nth-child(3) {
619     text-align: right;
620     font-size: 1rem;
621 }
622
623
624 }

```


動作確認①

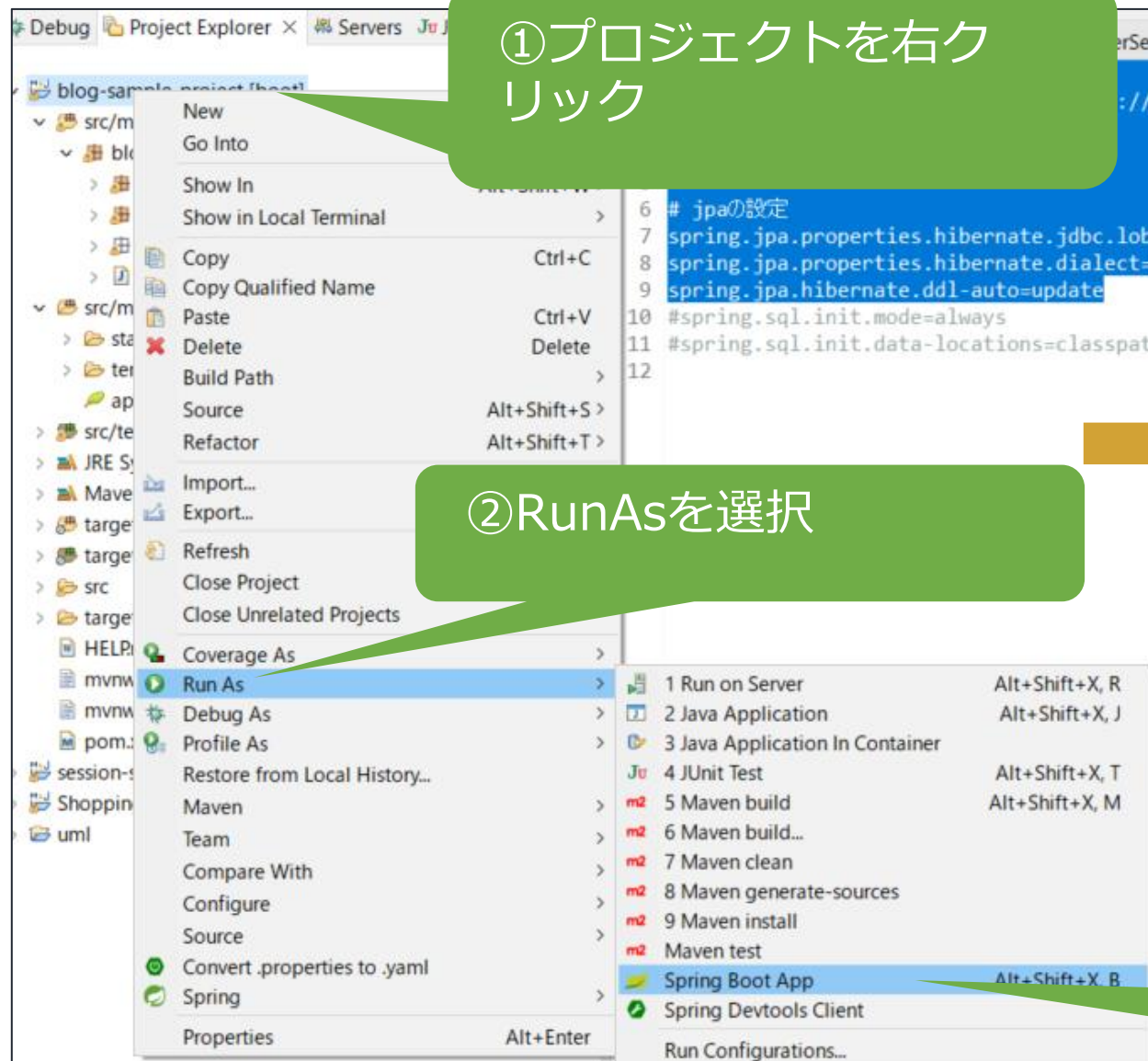
- 最後は、登録できるかどうかの確認をしていきます。

①プロジェクトを右クリック

②RunAsを選択

③SpringBootApplicationをクリック

④エラーがないかを確認



動作確認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`

← → ↻ ⓘ localhost:8080/admin/register

アプリ Stageee - ステージ Chatwork ショッピングリスト COACHTEC

ログイン

管理者登録画面

UserName

UserEmail

Password

登録 戻る

使用するブラウザは「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。

管理者登録画面


UserName 前沢昭

UserEmail matest@test.com

Password password123

登録 戻る

localhost:8080/admin/login



LoginControllerを作成していないため、この画面になる。

このページは動作していません

localhost でリダイレクトが繰り返し行われました。

Cookie を消去してみてください。

ERR_TOO_MANY_REDIRECTS

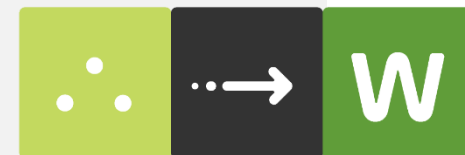
再読み込み

	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

Thank you!

From Seeds to Woodland — Shape Your Future.



Shape Your Future