

# innova patika spring bootcamp 4. hafta ödevi

---

öğrenci : ozan aydoğan

öğretmen : hamit mızrak

github :

# UserEntity

---

Bu yapı sayesinde, aslında User adında bir nesne oluşturulabilir diyebiliriz. her User'ın kullanıcı adı, Email, Parola, Userid, gibi özellikleri bulunur. Bu “Özellik nesnesini” tanımlayarak, bu classtan userlar oluşturabiliriz.

@Table(name="WorldofWarcraft") yapısıyla, bu yapımızın database'de göstereceğimiz table adını vermiş oluruz.

@Column yapısıyla database'deki her bir satırın(modelin, userentity'nin), Userentityde tanımlanan özelliklerini kolonlarda göstermiş oluruz.

# UserEntity

---

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder

//entity
@Entity
@Table(name="WorldofWarcraftusers")
public class UserEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "user_id")
    private Long userId;

    @Column(name = "user_name")
    private String userName;

    @Column(name = "user_email")
    private String userEmail;

    @Column(name = "user_password")
    private String userPassword;

    @Column(name = "user_race")
    private String userRace;

    @Column(name = "user_created_date")
    @Temporal(TemporalType.TIMESTAMP)
    @CreationTimestamp
    private Date userCreatedDate;
}
```

# Data Transfer Object (DTO)

---

AutoMapper, projemizde Entity nesnelerini database'den çektiğimiz haliyle değil, bu nesneleri istediğimiz (UI'da bizim için gerekli olacak) formata çevirmemizi sağlayan basit bir kütüphanedir. DTO (Data Transfer Object) ise AutoMapper'ın dönüştürmesini istediğimiz format modelidir

arayüzümüzde, register sayfası için registerdto, login sayfası için logindto yapısı oluşturdum. bu dto yapıları, view'den girilen değerleri bir dto'ya aktararak taşınmasını sağlar

# LoginDTO

---

```
package com.innova.dto;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.extern.log4j.Log4j2;

import javax.validation.constraints.NotEmpty;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Log4j2
@Builder
public class LoginDto {

    @NotEmpty(message = "Kullanıcı adı boş olamaz")
    private String userName;

    @NotEmpty(message = "Şifre Boş olamaz")
    private String userPassword;
}
```

# UserController

---

Usercontroller yapısıyla, web sitemizi @Getmapping annotation ile sayfalandırıp, bu sayfalarda bulunacak olan model yapısını tanımlamış oluruz, MVC (Model View Control) yapısına göre, Model, View yardımıyla (ekran, arayüz olabilir) kullanıcı tarafından girilen dataların tutulduğu veya Database'den alınan verilerin view ekranında gösterilmesini sağlayan bir objedir. Model yapısı DB ile sürekli iletişim halindedir. View'den girdigimiz degerleri Database'ye aktarmaya yarayan, database'den aldığı verileri View üzerinde kullanıcılara göstermeye yarayan bir yapıdır.

# UserController

---

@GetMapping'de tanımlanan model ile view'den verilere erişilir ve bu veriler, yine Getmappingde oluşturduğumuz DTO yapılarına aktarılır. @PostMapping yapısıyla, Builder tasarım kalıbı sayesinde veriler bir Entity yapısına aktarılır ve bu aktarılan veriler, ilişkilendirdiğimiz veritabanına aktarılır. Model yapıları, veritabanımızdaki tablomuzda bulunan her bir satırdır diyebiliriz.

# usercontroller ve database

The screenshot displays the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows a tree view of the database structure. The 'worldof\_warcraftusers' table is selected under the 'innova\_patika\_project\_1' database. The 'Columns' pane for this table lists: user\_id (bigint AI PK), user\_created\_date (datetime), user\_email (varchar(255)), user\_name (varchar(255)), user\_password (varchar(255)), and user\_race (varchar(255)).

The main query editor shows the following SQL query:

```
SELECT * FROM innova_patika_project_1.worldof_warcraftusers;
```

The 'Result Grid' pane displays the query results. The first row is highlighted in red and labeled 'model' with red arrows pointing to it. The data in this row is as follows:

user_id	user_created_date	user_email	user_name	user_password	user_race
18	2022-02-11 13:10:30	asdfaew@gmail.com	sedfawer	qweqopkqopewikq	HORDE
19	2022-02-11 13:17:29	asdfaew@gmail.com	sedfawer	erterter	HORDE
20	2022-02-11 13:19:06	asdfaew@gmail.com	sedfawer	erterter	HORDE
21	2022-02-11 13:24:46	asdfaew@gmail.com	sedfawer	erterter	HORDE
22	2022-02-11 13:26:08	qweqw@gmail.com	awsdqw	qwpeiqqewqw	ALLIANCE
23	2022-02-11 13:26:26	qweqw@gmail.com	awsdqw	eawqfqwrq	HORDE
24	2022-02-11 14:30:50	qpwkeolopgwe@hotmail.com	qweqweqw	qokweowiqe	HORDE
25	2022-02-11 14:41:14	qweqweqw@gmail.com	asfdaqwedqw	weqweq	ALLIANCE

The 'Output' pane at the bottom shows the execution log with 5 rows of results, each indicating a successful SELECT query and the number of rows returned.

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.



# IUserRepository

---

Bu yapı sayesinde, CRUD işlemleri gerçekleştirmemiz için gerekli olan metotları CrudRepository interface yapısından implement edip Kullanmamızı sağlayan yapıdır. Ayrıca kendimiz, özel bir DB sorgusu yazmak istersek IUserRepository yapısında bunu oluşturabiliriz.

UserController yapısında oluşturduğumuz, IUserRepository yapısı ile IUserRepository.save metodunu kullanıp, verilerimizi database'ye aktarabildik.

# kayıt ekranı

← → ↻ ⓘ localhost:8080/WoW/register

SYLVANASLA BİRLİKTE SAVAŞMAK İÇİN KAYIT OL

Kullanıcı Adı

E-mail

Şifre

Tarafını Seç!

sign-up

Zaten Hesabın var mı ? o zaman  
sylvanasın yanına koş!

**Sylvanasın yanına koş!**

# kayıt ekranı

---

UserController yapısında oluşturduğumuz, @GetMapping("register") ile bu sayfaya erişebiliyoruz,

Controller yapısını tanımlarken kullandığımız, @RequestMapping(value = "WoW") yapısı ile localhost:8080/ kısmından sonraki mapping işleminin WoW olması gerektiğini belirttik, yani bu sayfaya erişebilmek için <http://localhost:8080/WoW/register> url adresine gitmeliyiz

# kayıt ekranı

---

giriş ekranında doldurulması gereken herhangi bir alanı doldurmazsak, bu alanların doldurulması gerektiğini söyleyen bir uyarı mesajı gelir. Eğer hali hazırda bir kaydımız varsa giriş ekranına yönlendirilmek için “Sylvanas’ın yanına koş” butonuna tıklayabiliriz. bu sayfada oluşturduğumuz bir model yapısıyla, bu sayfada girilen verilere erişilir.

register.html sayfasındaki `<form>` yapıları içerisinde tanımladığımız `th:object="${register_form}"` yapısıyla, bu html sayfasını, login\_form modeliyle ilişkilendiririz.

```
<body>

<h1>Sylvanasla birlikte Savaşmak için kayıt ol</h1>

  <div class="login-form">

    <form id="registerForm" autocomplete="off" th:method="post" th:action="@{/WoW/register}" th:object="${register_form}">

      <p>Kullanıcı Adı</p>
      <input th:field="*{userName}" type="text" class="form-control" placeholder="Kullanıcı adı" autofocus>
      <div th:if="${#fields.hasErrors('userName')}" th:errors="*{userName}"></div>
      <p>E-mail</p>
      <input th:field="*{userEmail}" type="text" class="form-control" placeholder="E-mail" autofocus>
      <div th:if="${#fields.hasErrors('userEmail')}" th:errors="*{userEmail}"></div>
      <p>Şifre</p>
      <input th:field="*{userPassword}" type="password" class="form-control" placeholder="Şifre" autofocus>
      <div th:if="${#fields.hasErrors('userPassword')}" th:errors="*{userPassword}"></div>
      <p>Tarafları Seç</p>
      <select th:field="*{userRace}" th:required="required">
        <option value="">Tarafları seç</option>
        <option value="ALLIANCE">FOR THE ALLIANCE</option>
        <option value="HORDE">FOR THE HORDE</option>
      </select>
      <div th:if="${#fields.hasErrors('userRace')}" th:errors="*{userRace}"></div>
      <button id="registerButton" >
        sign-up
      </button>
    </form>

    <p>
      <span style="color: aqua">Zaten Hesabın var mı ? o zaman sylvanasın yanına koş</span>
      <button style="color: red" onclick="forwardLogin();" class="pointer">Sylvanasın yanına koş!
    </button>
  </p>

</div>
```

# kayıt işlemi

Herhangi bir kayıt işlemi gerçekleştirdiysek, bu sayfada bulunan model yapısında bulunan veriler RegisterDTO ile @PostMapping yapısında ilişkilendirilir. oluşturulan userentity yapısıyla, builder kalıbını kullanarak, dto yapısından alınan veriler entity yapısına aktarılır ve bu entity database'ye gönderilir.

```
// http://localhost:8080/WoW/register
@PostMapping("/register")
public String postRegister(@Valid @ModelAttribute("register_form") RegisterDto registerDto, BindingResult bindingResult){
    if(bindingResult.hasErrors()){
        log.error("Hata");

        log.info(registerDto);

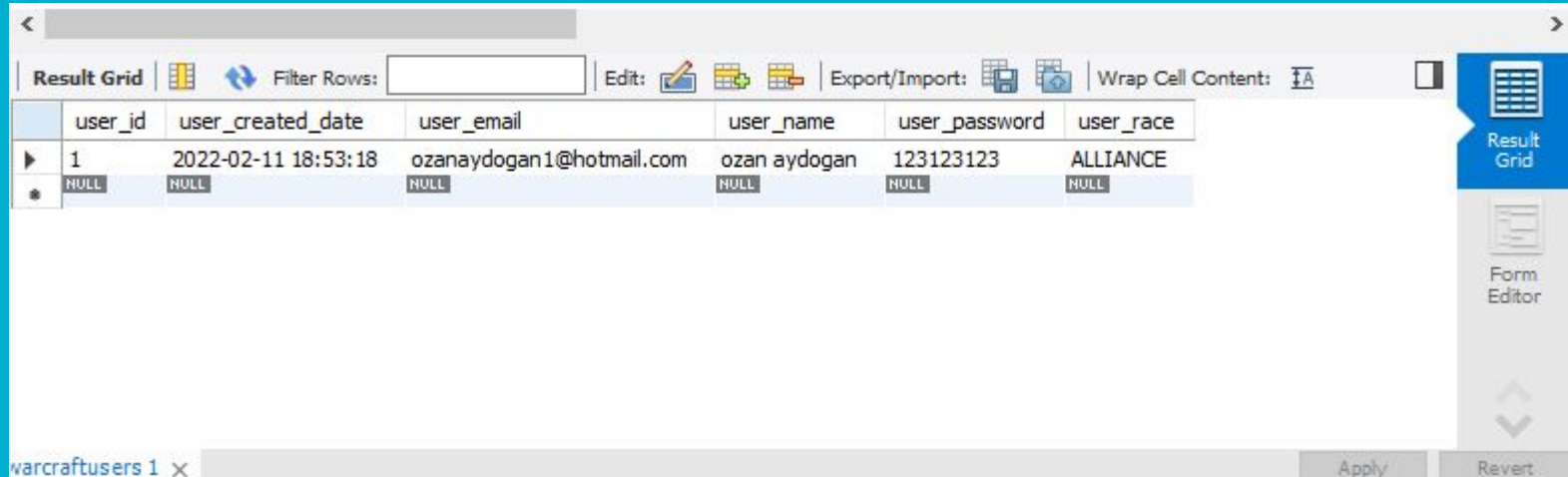
        return "register";
    }
    //Dto olarak alınan veri maplenerek Entity'e eklendi
    UserEntity userEntity = UserEntity
        .builder().userId(0L).userName(registerDto.getUserName()).userPassword(registerDto.getUserPassword())
        .userEmail(registerDto.getUserEmail()).userRace(registerDto.getUserRace()).build();

    iUserRepository.save(userEntity);

    log.info(registerDto);
    return "redirect:login";
}
```

# kayıt işlemi

iUserRepository.save(userentity) yapıyla veriler database'ye aktarılır. kayıt başarılı!



	user_id	user_created_date	user_email	user_name	user_password	user_race
▶	1	2022-02-11 18:53:18	ozanaydogan1@hotmail.com	ozan aydogan	123123123	ALLIANCE
*	NULL	NULL	NULL	NULL	NULL	NULL

# kayıt işleminden sonra

---

kayıt işleminden hemen sonra login sayfasına yönlendiriliz. eğer kaydımız yoksa yine kayıt işlemini gerçekleştirmek için kayıt ol ve sylvanasın yanına koş butonuna tıklayarak yine kayıt ekranına dönüş yapabiliriz. bu sayfalar arası geçiş işlemi bir script yapısıyla sağlanır.

```
<script>
    function forwardRegister()
    {
        window.location.href="/WoW/register";
    }
</script>
```



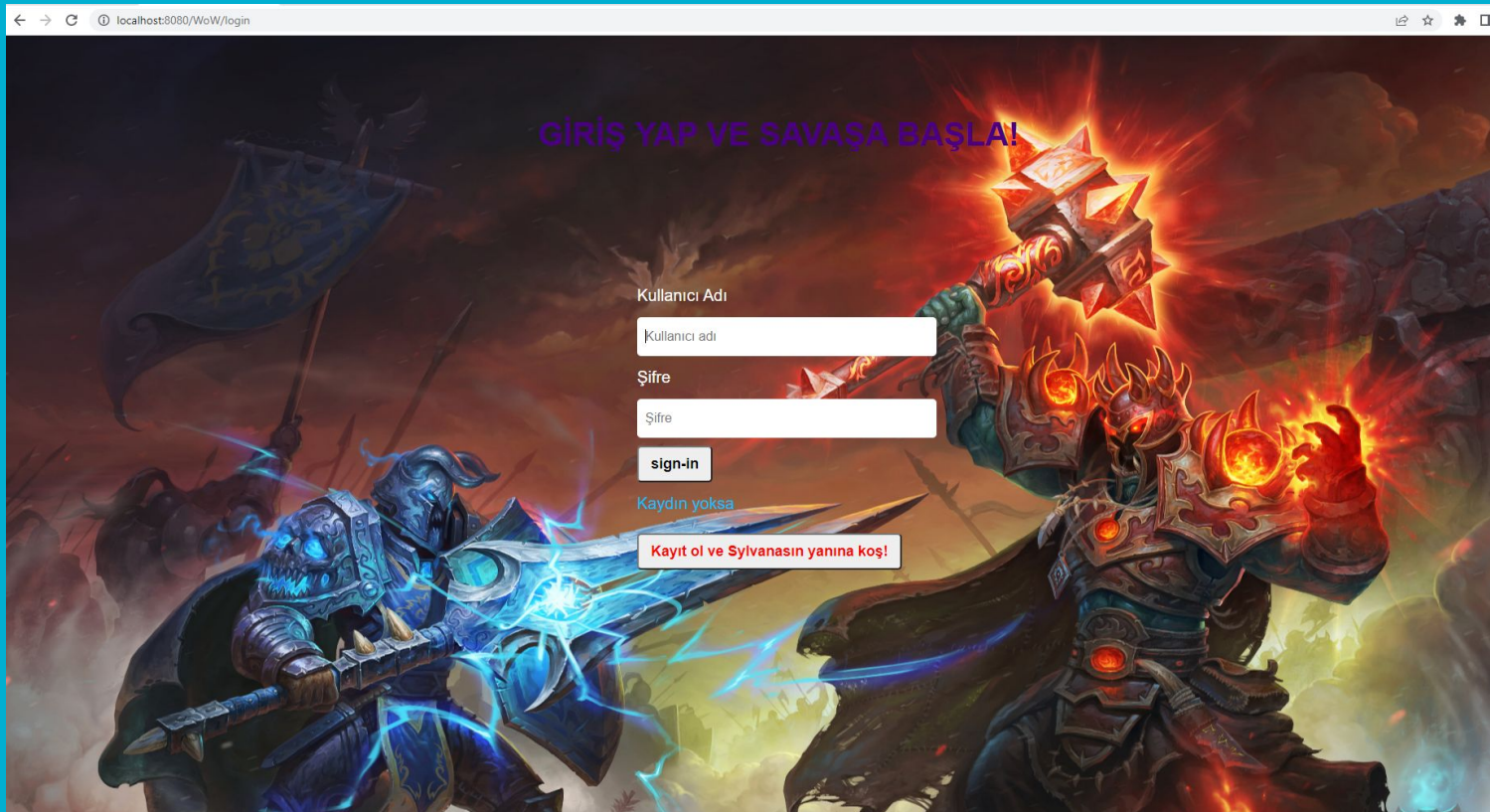
# giriş ekranı

---

giriş ekranındaki model yapısı yine UserController yapısında bulunan @GetMapping annotation ile gerçekleşir.

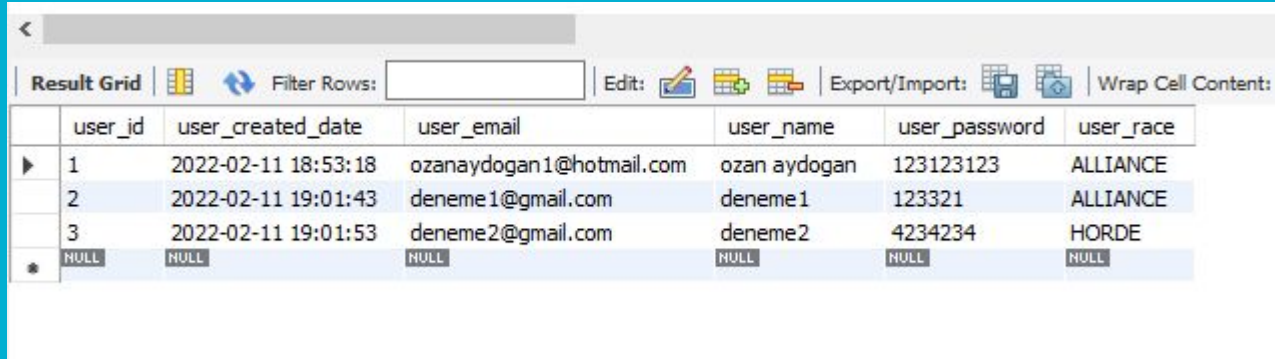
```
@GetMapping("/login")
public String getLogin(Model model){
    model.addAttribute("login_form", new LoginDto());
    return "login";
}
```

# giriş ekranı



# kayıtlı tüm kullanıcıların rest olarak gösterilmesi

veri tabanımızda 3 kayıt bulunuyor olsun



	user_id	user_created_date	user_email	user_name	user_password	user_race
▶	1	2022-02-11 18:53:18	ozanaydogan1@hotmail.com	ozan aydogan	123123123	ALLIANCE
	2	2022-02-11 19:01:43	deneme1@gmail.com	deneme1	123321	ALLIANCE
	3	2022-02-11 19:01:53	deneme2@gmail.com	deneme2	4234234	HORDE
✱	NULL	NULL	NULL	NULL	NULL	NULL

# kayıtlı tüm kullanıcıların rest olarak gösterilmesi

---

bu kayıtların json tipinde gösterilebilmesi için bir restcontroller yapısı oluşturuk

```
@RestController
public class UserRestController {

    @Autowired
    IUserRepository iUserRepository;

    @GetMapping("/rest/getAllUser")
    public Iterable<RegisterDto> getAllUser(){

        List<RegisterDto> dtoList = new ArrayList<>();
        Iterable<UserEntity> userList = this.iUserRepository.findAll();

        for (UserEntity user: userList) {

            RegisterDto dto = RegisterDto.builder()
                .userId(user.getUserId()).userName(user.getUserName()).userPassword(user.getUserPassword())
                .userEmail(user.getUserEmail()).userRace(user.getUserRace()).build();

            dtoList.add(dto);
        }

        return dtoList;
    }
}
```

# kayıtlı tüm kullanıcıların rest olarak gösterilmesi

controller yapısında tanımladığımız  
@GetMapping("/rest/getAllUser") ile bu url'ye  
bağlanıp, veri tabanında olan tüm verileri json  
formatında görebiliriz.

Key -> Value ilişkisi



```
[
  {
    "userId": 1,
    "userName": "ozan aydogan",
    "userEmail": "ozanaydogan1@hotmail.com",
    "userPassword": "123123123",
    "userRace": "ALLIANCE"
  },
  {
    "userId": 2,
    "userName": "deneme1",
    "userEmail": "deneme1@gmail.com",
    "userPassword": "123321",
    "userRace": "ALLIANCE"
  },
  {
    "userId": 3,
    "userName": "deneme2",
    "userEmail": "deneme2@gmail.com",
    "userPassword": "4234234",
    "userRace": "HORDE"
  }
]
```