



DAM2 ETHAZI
4. Taldea

Pablo Lopez, Markel Salgado, Aritz Izarcelaya eta Joseba Hernandez

Aurkibidea

1.- Erronkaren Aurkezpena	4
1.1.- Sarrera	4
1.2.- Azalpena	4
1.3.- Sistemaren Instalazioa	5
1.4.- Sistemaren Funtzionamendua	6
2.- MongoDB Datu-basea	8
2.1.- Pokemon Kolekzioa	8
2.2.- Type Kolekzioa	9
2.3.- Users Kolekzioa	9
3.- Java RESTful API	10
3.1.- Azalpen Orokorra	10
3.2.- Swagger	10
3.3.- Javadoc	10
4.- ASP.NET Web Aplikazioa	11
4.1.- Nabigazio Barra eta Footer-a	11
4.2.- Index-a	12
4.3.- Pokemon List	13
4.4.-Login eta Register	15
4.4.1.- Login Aukerak: Insert Pokemon	17
4.4.2.- Login Aukerak: Delete	18
5.- Ondorioak	19
5.1.- Iturriak	19
5.2.- Konplexutasuna	19
5.2.1.- ID automatikoa	19
5.2.2.- Ausazko Pokemon-a	20
5.2.3.- Tipoen Lista	20
5.2.4.- Erabiltzaileak	22
5.3.- Erroreak eta egin gabekoak	23
5.4.- Teknologien balorazioa	24
5.4.1.- MongoDB	24
5.4.2.- Java RESTful API	25
5.4.3.- ASP.NET Web Aplikazioa	25

1.- Erronkaren Aurkezpena

1.1.- Sarrera

Gu Markel Salgado, Aritz Izarcelaya, Joseba Hernandez eta Pablo Lopez gara, eta 4. Taldea sortu dugu DAM 2. mailako 'Interfazeen Garapena' eta 'Datu Atzipena' ikasgaien arteko ETHAZI proiektuan aurkeztu zaigun ikasturte amaierako erronka egiteko. Lan honen GitHub errepositorioa [hemen](#) aurkitu dezakezu.

Erronka honen helburua datu-base baten gordetako datuekin web-orri bat garatzea da, bitartekari moduan RESTful API bat erabiliz. Hurrengo teknologiak erabili behar izan ditugu:

- MongoDB datu-basea.
- Java Spring Boot Framework-en egindako RESTful API-a.
- ASP.NET MVC egindako Web Orria (C# lengoaia).

Baldintza hauek kontuan izanda, gure lanarekin hasi baino lehen, teknologia hauei buruz ikasteko tutorial batzuk jarraitu ditugu:

- [RESTful API Azalpena](#)
- [RESTful Web Zerbitzurako Tutoriala](#)
- [MongoDBren atzipena REST bidez](#)
- [Aurreko pausuko REST API-a kontsumituko duen ASP.NET aplikazioa](#)

Behin tutorial hauekin bukatuta, benetako proiektuarekin hasteko ordua heldu da.

1.2.- Azalpena

Lehenik eta behin, beste edozerekin hasi baino lehen, proiektu honen gaia aukeratu behar izan dugu. Horretarako, interneten dagoeneko eginda dauden JSON fitxategiak bilatu ditugu, zerotik hasita egin behar den lan guztia aurrezteko.

REST API-aren tutoriallean MongoDB-ko ObjectID eremuarekin arazoak izan genituenenez (Azalpen gehiago [5.3. atalean](#)), beste mota bateko identifikadorea duen fitxategi bat bilatzea komenigarria zela pentsatu dugu. Hau kontuan izanda, GitHub-en Pokemon buruzko [JSON fitxategi hau](#) aurkitu genuen. Fitxategi honetan hainbat aldaketa egin ditugu eta proiektu honen GitHub errepositorioan aurkitzen den JSON fitxategia erabili dugu.

Pokemonak benetako animalietan, intsektuetan, objektuetan, landareetan edo izaki mitologikoetan inspiratutako fikziozko izakiak dira, izen bereko bideojokoetan agertzen direnak.

1.3.- Sistemaren Instalazioa

Proiektu honen elementuak martxan jarri ahal izateko, hurrengo baldintzak bete behar dira:

- MongoDB datu-basea instalatzea eta bertan GitHub-eko hiru JSON fitxategiak (pokemon.json, types.json eta users.json) inportatzea 'pokédex' izeneko datu-basean (MongoDB Compass erabiltzea gomendatzen dugu, oso sinplea delako). Bestela, mongoimport erabiliz, JSON fitxategiak "C:\Program Files\MongoDB\Server\4.4\bin" itsatsi eta hurrengo komandoak exekutatu Komando Lerrotik.

```
mongoimport --host=localhost:27017 --db=pokédex --collection=pokemon --file=pokemon.json
mongoimport --host=localhost:27017 --db=pokédex --collection=types --file=types.json
mongoimport --host=localhost:27017 --db=pokédex --collection=users --file=users.json
```

Java IDE bat izatea (Netbeans edo Eclipse, adibidez) eta bertan RESTful API-a exekutatzea.

- Visual Studio instalatuta izatea ASP.NET Web Orria exekutatu ahal izateko.

Honez gain, hurrengo aldaketak egin behar izan ditugu programa hauetan:

- MongoDB (Sare lokalean erabiltzeko):
 - "C:\Program Files\MongoDB\Server\4.4\bin\mongod.cfg" fitxategian aldaketa bat egin behar da "#network interfaces" atalean. Ezkerrean localhost ip dator eta eskuinean gure ekiparen sare ip-a jarriko dugu.

```
# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1,192.168.72.10
```

- Netbeans:
 - pom.xml fitxategiko dependentziak eta plugin-a:

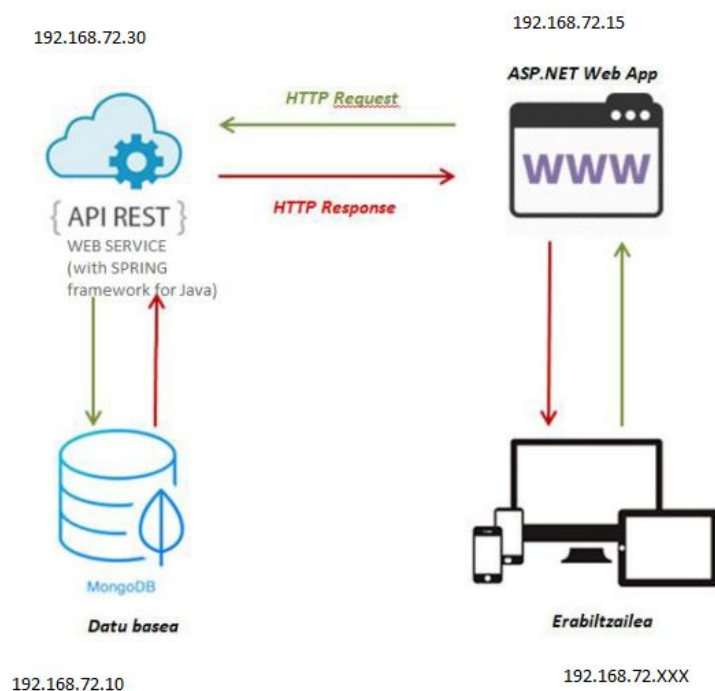
GroupId	ArtifactId	Bertsioa
org.mongodb	mongodb-driver-sync	3.11.1
org.springframework.boot	spring-boot-starter-web	-
org.springframework.boot	spring-boot-starter-test	-
io.springfox	springfox-swagger2	2.9.2
io.springfox	springfox-swagger-ui	2.9.2
org.junit.jupiter	junit-jupiter-engine	5.5.2
org.junit.vintage	junit-vintage-engine	5.5.2
org.assertj	assertj-core	3.13.2

org.springframework.boot (plugin)	spring-boot-maven-plugin	-
org.apache.maven.plugins (plugin)	maven-surefire-plugin	3.8.1

- Visual Studio:
 - Hurrengo NuGet paketeak instalatu:
 - System.Net.Http: Http klase basikoa da, Http eskaera bidaltzeaz eta Http erantzun-baliabideak jasotzeaz arduratzen da.
 - Microsoft.AspNet.WebApi.Client: Pakete honek Http formatu eta eduki aldaketa euskarriak eransten ditu.
 - PagedList eta PagedList.Mvc: Paginazioa ahalbidetzen duen paketea (Azalpen gehiago [4.3 puntuan](#)).

1.4.- Sistemaren Funtzionamendua

Erronka honen baldintza bat teknologia desberdinak gure klaseko ekipo desberdinetan banatzea da, eta hori kontuan izanda, ondorengo egitura planteatu dugu.






192.168.72.10 IP-a daukan ekipoak (Josebaren ekipoa) MongoDB datu-basea izango du, 192.168.72.30 ekipoak (Pablarena) Java RESTful API-a zerbitzatuko du, eta azkenik, 192.168.72.15 ekipoak (Markelen ekipoa) ASP.NET Web aplikazioa izango du.

Egitura honek funtziona dezan, Sistemaren Instalazioa atalean azaldutakoaz gain, hiru ekipoen artean konexioa dagoela ziurtatu behar dugu, hirurak sare berean izanik, eta MongoDB datu-basea daukan ekipoen 27017 portua ireki behar da, Firewall-ean sarrera arau bat ezarriz.

2.- MongoDB Datu-basea

Gure Proiektuaren datuak ‘**pokedex**’ izeneko MongoDB datubase bakarrean gorde ditugu. Datu base honetan 3 kolekzio ditugu: ‘**pokemon**’, ‘**types**’ eta ‘**users**’. Kolekzio bakoitzean datu ezberdinak gordetzen dira.

pokemon	151	276.9 B	40.8 KB	1	36.0 KB	
types	18	38.3 B	690.0 B	1	32.0 KB	
users	6	54.5 B	327.0 B	1	36.0 KB	

2.1.- Pokemon Kolekzioa

Gure kolekzio nagusia ‘**pokemon**’ da. Kolekzio honetan Pokemon desberdinen datuak gordeko ditugu, egitura honekin:

Eremua	Datu Mota	Deskribapena	Adibidea
_id	Integer	Pokemon bakoitzaren identifikadorea	2
name	String	Pokemon baten izena	Ivysaur
img	String	Pokemon baten irudiaren esteka	http://www.serebii.net/pokemon/art/002.png
type	String []	Pokemon baten Tipoa(k) (gutxienez 1, gehienez 2)	["Grass", "Poison"]
height	String	Pokemon baten altuera	0.99 m
weight	String	Pokemon baten pisua	13.0 kg
weaknesses *	List<String >	Pokemon baten ahuleziak, kasu honetan ez dago gutxiengo kopururik.	["Fire", "Ice", "Flying", "Psychic"]
prev_evolution **	List<Pokemon>	Pokemon baten aurreko eboluzioa(k), Pokemon objektu moduan (0-tik 2-ra)	[Pokemon("001", "Bulbasaur")]
next_evolution **	List<Pokemon>	Pokemon baten hurrengo eboluzioa(k), Pokemon objektu moduan	[Pokemon("003", "Venusaur")]

***Pokemon** bakoitzak bere **Tipoen** arabera beste **Tipo** batzuen aurka ahula da. Normalean taula baten bidez atera ahal dira ahuleziak, baina inplementatzeko oso zaila izango zen ([taula ikusi](#)).

Izan liteke **Pokemon batek aurreko edo hurrengoko **eboluziorik** ez izatea. Eboluzioren bat badu, **Zenbakia** eta **Izena** gordeko dugu. **Pokemon** bakoitzak 0-2 eboluzio izan ditzake, 'Eevee'-ren kasua izan ezik, berak 3 eboluzio desberdin izan ditzake.

2.2.- Type Kolekzioa

Kolekzio hau irakurtzeko soilik da. **Pokemon** munduan **18 Tipo** desberdin daude (hau aldaezina da), eta **Pokemon** batek **Tipo** horietatik **2** izan ditzake gehienez. Era honetan egitea aukeratu dugu norbaitek tipo bateko **Pokemon** guztiak borraraten baditu, **Tipo** horretako **Pokemon** berriak sartzeko aukera izateko.

```
_id: ObjectId("600fd1773591441e545ffdf6")  
name: "Bug"
```

```
_id: ObjectId("600fd1773591441e545ffdf7")  
name: "Dragon"
```

2.3.- Users Kolekzioa

Erabiltzaileen kudeaketaz arduratzen den kolekzioa da, eta eremu hauek osatzen dute: **id**, **username**, **password**. **Id**-a **int** eta beste biak **string** motakoak izango dira.

```
_id: 0  
username: "ByManolo69XD"  
password: "abc123"
```

3.- Java RESTful API

3.1.- Azalpen Orokorra

Gure MongoDB datu-baseko edukia atzikitze eta Web Aplikazioan erakusteko, Javan egindako REST API bat erabiliko dugu. Lehenik eta behin, tutorial [honetan](#) agertzen den [GitHub errepositorioa](#) klonatu dugu eta adibide hori oinarri moduan hartuta gure datu-basera moldatu dugu.

Gure REST API-aren klase diagrama [GitHub-en](#) aurkitu dezakezu. Datu-basean planteatutako egiturarekin jarraitzeko, klase bi erabiliko ditugu: Pokemon eta User. Tipo klase bat sortzea zentzurik gabekoa dela uste dugu, eremu bakarra duelako (Tipo izena).

Aurrekoa kontuan izanda, **PokemonRepository** izeneko interfasea erabili dugu, eta **MongoDBPokemonRepository** klaseak, interfase hori inplementatuz, Mongo datu-basean eragiketak egin ditzake. Beste alde batetik, PokemonController klaseak ASP aldetik helduko diren eskaerak jasoko ditu eta erantzun bat emango die, API-aren Mapping desberdinak erabiliz (adibidez, '/api/pokemon' Pokemon guztien datuak itzuliko ditu).

3.2.- Swagger

Horrez gain, Swagger konfiguratu dugu. Swagger-ren bidez API-ak eskaintzen dituen aukera guztiak probatu eta ikusi daitezke, probako datu batzuekin edo zuk aukeratutako datuekin. Swagger-rera sartzeko, behin zerbitzua hasita <http://localhost:8080/swagger-ui.html> helbidera joango gara.

3.3.- Javadoc

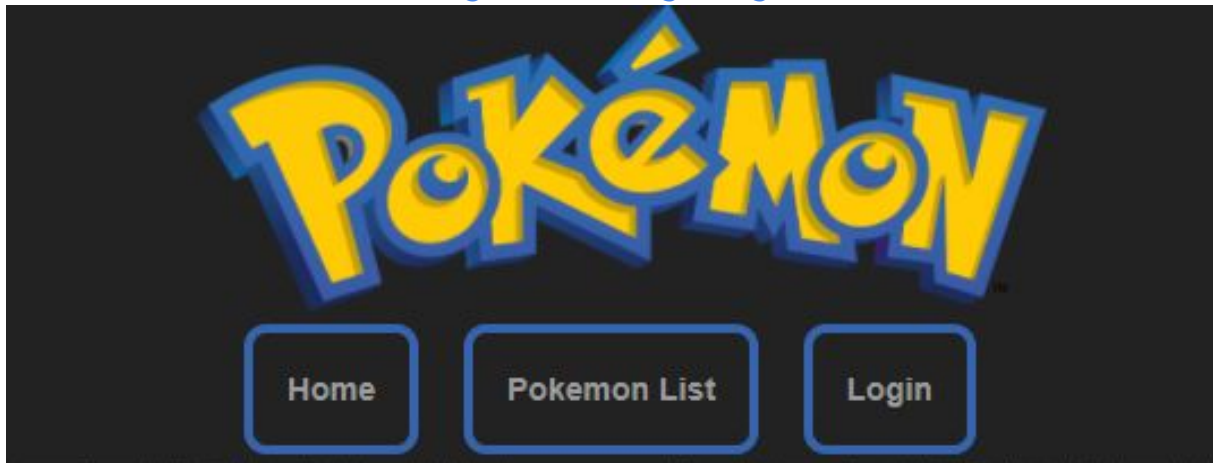
Amaitzeko, Javadoc-a prestatu dugu baita ere. GitHub errepositorioko helbide [honetan](#) aurki dezakezu.

4.- ASP.NET Web Aplikazioa

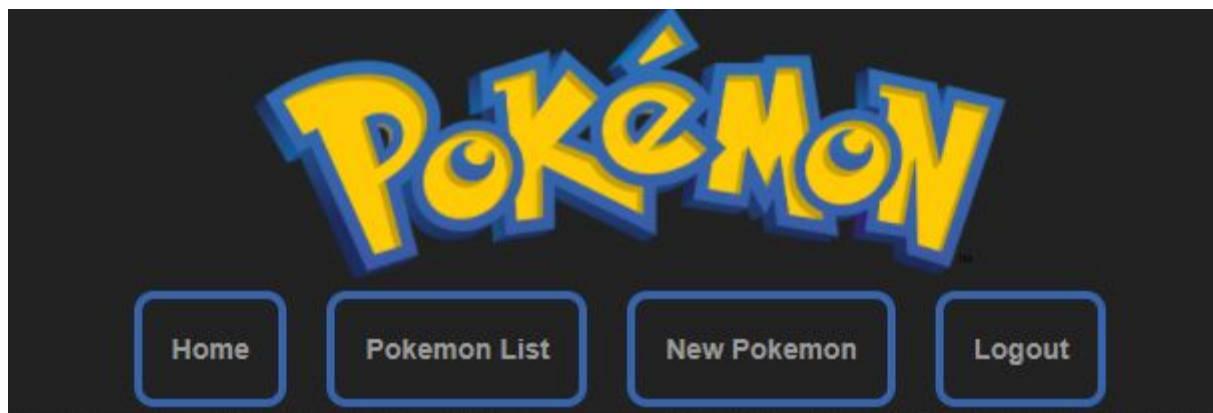
4.1.- Nabigazio Barra eta Footer-a

‘_Layout.cshtml’ fitxategian beste fitxategietarako erabiliko dugun plantila modukoa egin dugu, navbar eta footer batekin. Erabiltzailea logeatuta dagoenean, menuko aukerak aldatu egingo dira.

Nabigazio barra logeatu gabe



Nabigazio barra logeatuta zaudenean



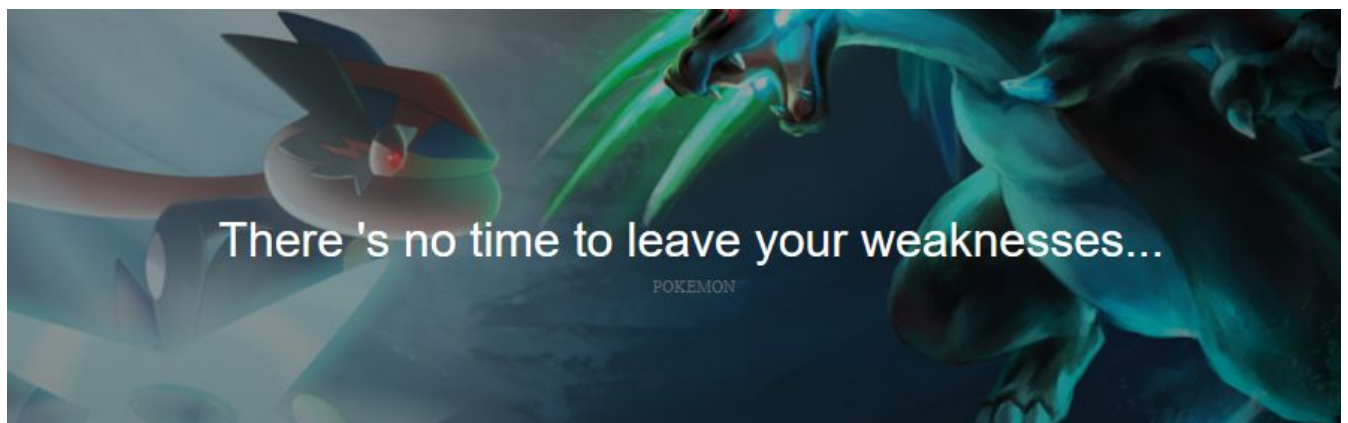
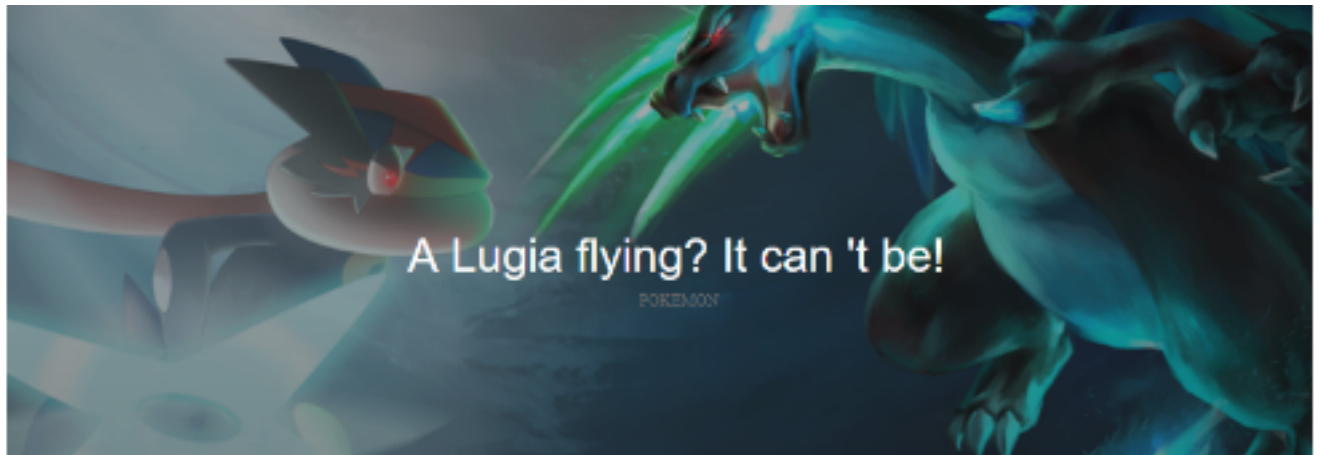
Footer-a



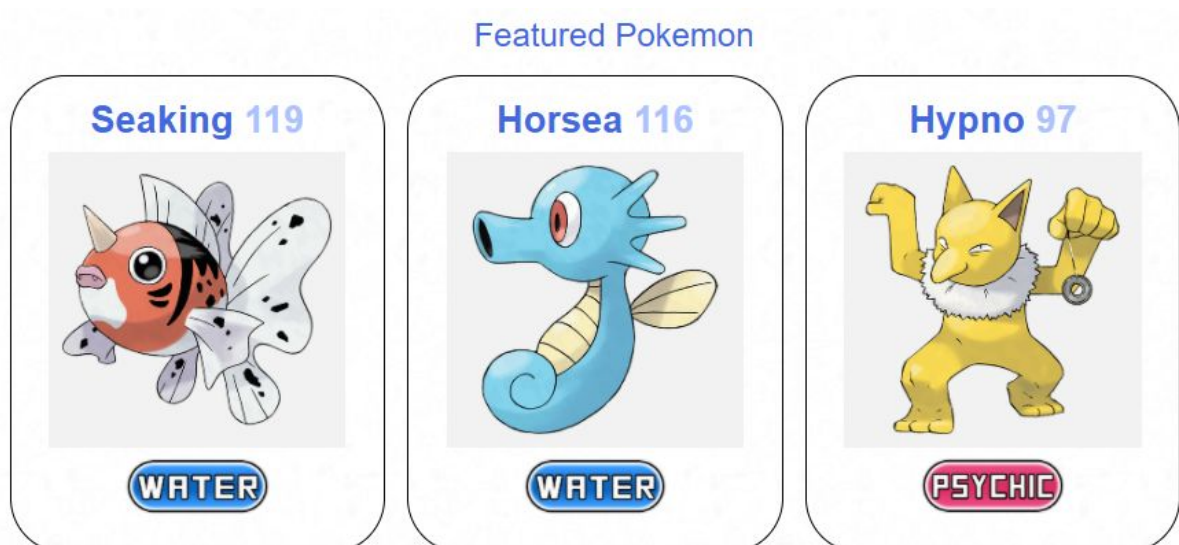
4.2.- Index-a

Web aplikazioaren orrialde nagusia Index-a da. Estiloari erreparatuz, berezitasun batzuk ditu:

- Irudi bat daukagu, eta gainean automatikoki aldatzen den testua dago. Hau egiteko JavaScript erabili dugu.

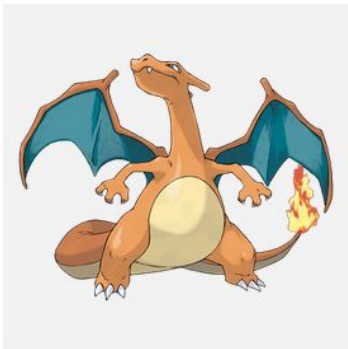


- Bestalde, orria kargatzen duzun bakoitzean, 'Featured Pokemon' izeneko atalean datu-baseko hiru Pokemon agertuko dira ausaz.



4.3.- Pokemon List

Datu-baseko Pokemon-en edukia erakusteko erabili dugu. Atal honetan egin dugun lehendabiziko ekintza datu guztiak lortzea eta imprimatzea izan da, eta ondoren gure gustuko *itxura* eman diogu:



Charizard N.º 6

Type:

FIRE **FLYING**

Weaknesses:

WATER **ELECTRIC** **ROCK**

Height:	Weight:	Prev Evos:
1.70 m	90.5 kg	4 - Charmander 5 - Charmeleon


Datu ugari ditugunez, zehazki, 151 Pokemon, nabigazioa errazteko Paginazioa gehitu diogu aplikazioari:

[Home](#) [Pokemon List](#) [Login](#)

Showing All Pokemon

DARK **DRAGON** **ELECTRIC** **FAIRY** **FIGHTING** **FIRE** **FLYING** **GROUND** **ICE** **NORMAL** **POISON** **PSYCHIC** **ROCK** **STEEL** **WATER**

1 2 3 4 5 6 7 8 9 10 ... » »»



Bulbasaur N.º 1

Type:

GRASS **POISON**

Weaknesses:

FIRE **ICE** **FLYING** **PSYCHIC**

Height:	Weight:	Next Evos:
0.71 m	6.9 kg	2 - Ivysaur 3 - Venusaur

Paginazioa egiteko, 1.3 puntuan azaldu dugun bezala, pakete batzuk instalatzea beharrezkoa da.

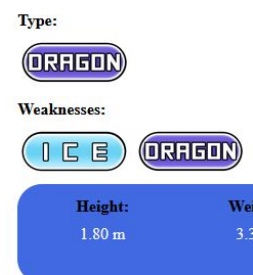
```
public async Task<ActionResult> PokemonList(int? page, string type)
{
    List<Pokemon> PokInfo = new List<Pokemon>();
    using (var client = new HttpClient())
    {
        client.BaseAddress = new Uri(models.Pokemon.BaseURL);
        client.DefaultRequestHeaders.Clear();
        client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue('
        HttpResponseMessage Res;
        if (type != "0")
        {
            Res = await client.GetAsync("api/pokemon/type/" + type);
        }
        else
        {
            Res = await client.GetAsync("api/pokemon");
        }
        if (Res.IsSuccessStatusCode)
        {
            var PokResponse = Res.Content.ReadAsStringAsync().Result;
            PokInfo = JsonConvert.DeserializeObject<List<Pokemon>>(PokResponse);
        }
        int pageSize = 10;
        int pageNumber = (page ?? 1);
        return View(PokInfo.ToPagedList(pageNumber, pageSize))
    }
}
```

Horrez gain, atal honetan filtraketa aukera bat egin dugu, dauzkagun Pokemon-ak tipoarengatik filtratzeko. Tipo baten irudia agertzen den edozein tokitan filtroa gauzatuko da. Hau lortzeko **GetTypeList()** izeneko metodo bat egin dugu (Azalpen gehiago [5.2.3 atalean](#)):

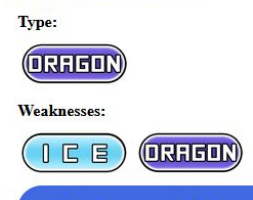
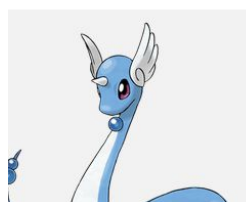


Showing **DRAGON** Type Pokemon

Kasu honetan, 'Dragon' tipoko Pokemonak agertuko dira.

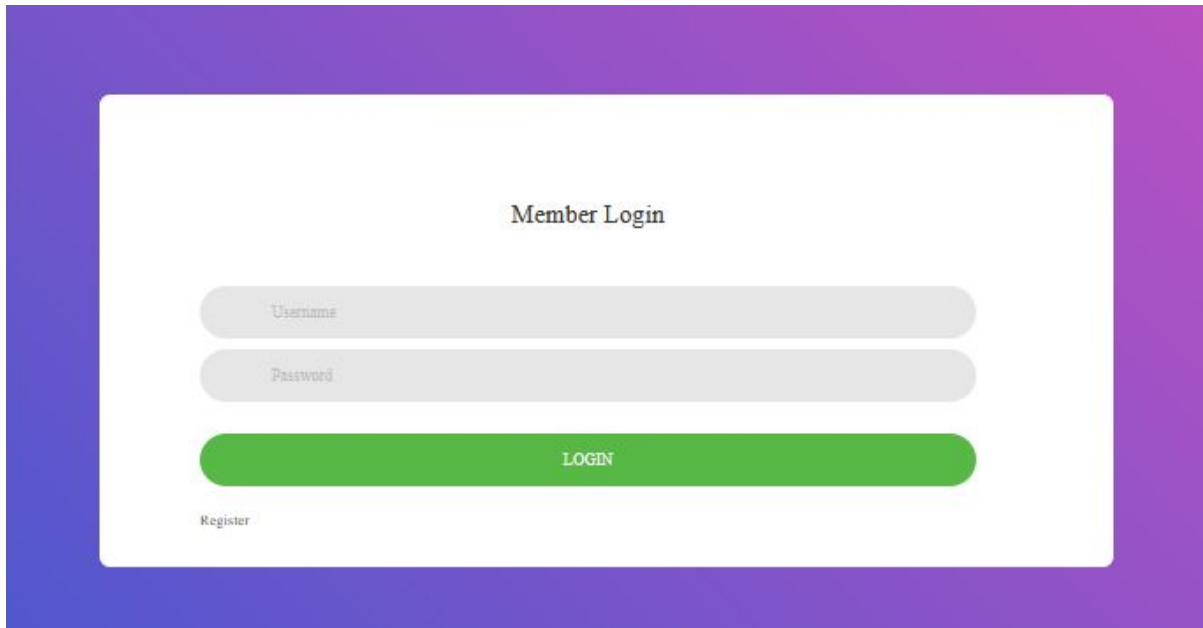


Dragonair N.º 148



4.4.-Login eta Register

Login eta Register formularioak egiteko, formulario plantilla bat sortu dugu eta atal bietarako moldatu ditugu:



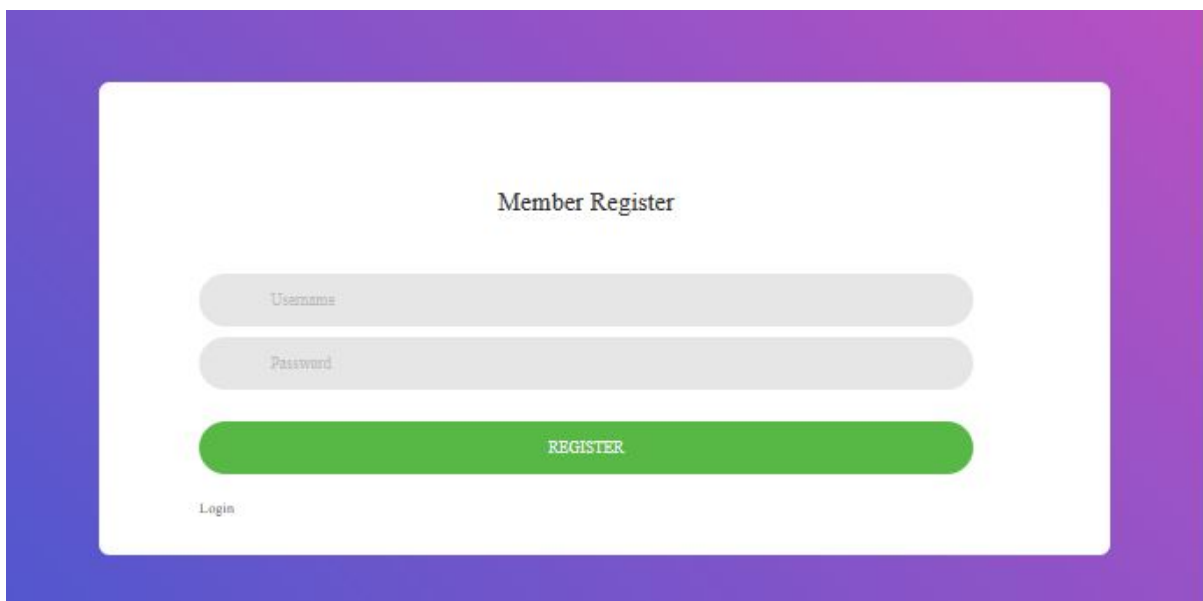
Member Login

Username

Password

LOGIN

Register



Member Register

Username

Password

REGISTER

Login

Formulario hauetan Username edo Password-ak utzik utziz gero, mezu bat agertzen dea, hauek bete arte:

Member Register

Username

El campo username es obligatorio.

Password

El campo password es obligatorio.

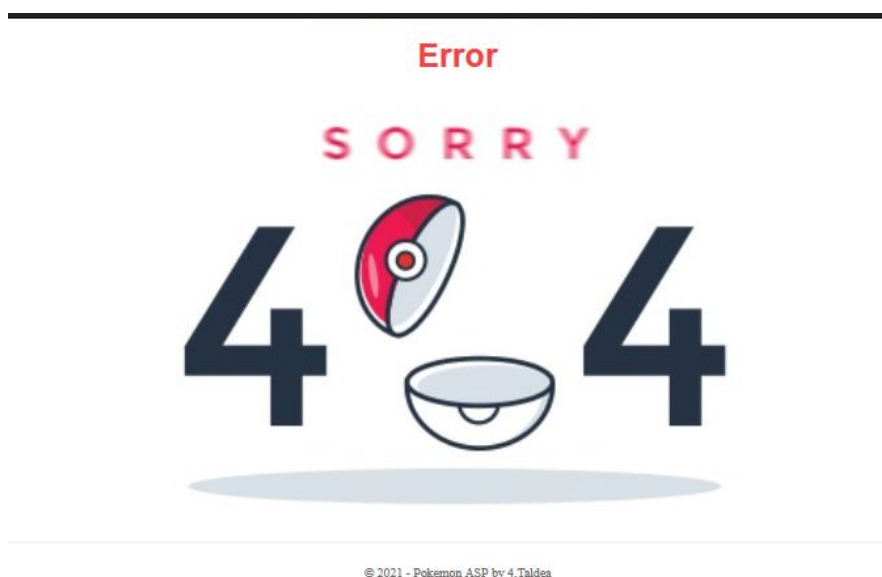
REGISTER

Login

Hau lortzeko, User klasean eremu hauei **[Required]** propietatea jarri diegu.

```
public class User
{
    [Required]
    public string username { get; set; }
    [Required]
    public string password { get; set; }
}
```

Horretaz gain, logeatzerako orduan datuak gaizki egonez gero errore orri batera eramaten zaitu:



Logeatzeko orduan, erabiltzailea eta pasahitza zuzenak badira, Index-era bidaltzen digu eta orain bi aukera gehiago agertuko dira menuan.

4.4.1.- Login Aukerak: Insert Pokemon

Menuan agertu den aukera berrietako bat Pokemon berri bat sortzeko formularioa da. Formulario honek bi atal ditu, Orokorra eta Eboluzioen atala. Atal orokorreko eremu guztiak betetzea derrigorrezkoa da 'Secondary Type' izan ezik. Tipoezin zerikusia duten eremuen aukerak betetzeko **GetTypeList()** metodoa erabili dugu berriro ere. 'Weakness(es)' eremua Select Multiple motakoa da, eta bertan jartzen duen moduan Ctrl tekla sakatuta bat baino gehiago aukeratu ditzakegu. Height eta Weight eremuetan 'm' eta 'kg' automatikoki gehituko ditugu.

Insert Pokemon

Name	<input type="text" value="Name"/>
Image	<input type="text" value="Image URL"/>
Primary Type	<input type="text" value="Bug"/>
Secondary Type	<input type="text" value="(None)"/>
Height	<input type="text" value="Height"/> m
Weight	<input type="text" value="Weight"/> kg
Weakness(es)	<div><div>Bug</div><div>Dark</div><div>Dragon</div><div>Electric</div><div>Fire</div></div> <div>Press CTRL to select more than one</div>

Eboluzioen atala aukerakoa da. Pokemon batek bi eboluzio izan ahal dituen gehienez, aukera hori jarri dugu. Lehenik eta behin eboluzio mota aukeratu behar

Evolutions (Optional)


First Evolution	<input type="text" value="(None)"/>
Second Evolution	<input type="text" value="(None)"/>
<input type="button" value="Create"/>	

da, hau da, aurreko edo ondorengo eboluzioa den. Jarraian, JavaScript-en bidez beste bi eremu agertuko dira, zenbakia eta izena gordetzeko.

First Evolution	Previous Evolution
First Evolution Number	Number
First Evolution Name	Name
Second Evolution	Next Evolution
Second Evolution Number	Number
Second Evolution Name	Name

4.4.2.- Login Aukerak: Delete

Logeatuta bagaude, eta Pokemon List-era joaten bagara, Pokemon bakoitzaren ondoan zakarrontzi ikono bat agertuko da. Ikono hau sakatuz gero, Pokemon hori datu-basetik borratuko dugu.



Bulbasaur N.º 1


Type:

GRASS POISON

Weaknesses:

FIRE ICE FLYING PSYCHIC

Height:	Weight:	Next Evos:
0.71 m	6.9 kg	2 - Ivysaur 3 - Venusaur



```
@if (Pokemon_ASP.Models.User.logged)
{
    <div class="delete">
        <a title="All Pokemon" href="@Url.Action("DeletePokemon", "Pokemon", new { id= item.id })">
            
        </a>
    </div>
}
```

5.- Ondorioak

5.1.- Iturriak

Lehenik eta behin, Pokemon datu-baserako erabili dugun [JSON-a](#) GitHub-eko Biuni erabiltzailearengandik hartu dugu. Beste alde batetik, Netbeans-en JavaDoc nola sortzeko informazioa hemen aurkitu dugu.

5.2.- Konplexutasuna

Atal honetan gure iritziz konplexuak izan diren edukiak eta haien kodea azalduko ditugu.

5.2.1.- ID automatikoa

Java API-ko **save()** metodoan, hau da, Pokemon berri bat gordetzeko metodoan, Pokemon berriari ID-a automatikoki sartuko zaio, bi irizpide jarraituz:

- 1.- Azken ID-a baino lehen ID zenbakiren bat hutsik badago, ID-a zenbaki hori izango da.
- 2.- Zenbaki librerik ez badago, azken ID-ari bat gehituko dio.

```
public Pokemon save(Pokemon pokemon) {
    pokemon.setId(0);
    List<Pokemon> pokemonGuztiak = findAll();
    for (int i = 0; i < pokemonGuztiak.size(); i++) {
        Pokemon p = pokemonGuztiak.get(i);
        if (p.getId() != i + 1) {
            pokemon.setId(i + 1);
            break;
        }
    }
    if (pokemon.getId() == 0) {
        Pokemon azkenPokemon = pokemonCollection.find().sort(new Document("id", -1)).first();
        int id = azkenPokemon.getId() + 1;
        pokemon.setId(id);
    }
    pokemonCollection.insertOne(pokemon);
    return pokemon;
}
```

5.2.2.- Ausazko Pokemon-a

Ausaz datu-basetik Pokemon bat hartzen duen metodoa egiteko MongoDB Aggregates erabili dugu.

```
@Override
public Pokemon findRandomPokemon() {
    return pokemonCollection.aggregate(Arrays.asList(Aggregates.sample(1))).first();
}
```

5.2.3.- Tipoen Lista

Beste alde batetik, Tipoak hartzeko metodoa konplexutasun maila nahiko altua duenez, era sakon baten azalduko dugu:

Lehenik eta behin, Javan Tipo guztiak hartu nahi ditugu datu-basetik, baina balio sinplea denez (hau da, String bakar bat da) ez dugu Objektorik behar. Horretarako, kontsulta egiterakoan **excludeID()** erabiliko dugu, izenak dioen bezala, ID-a kanpoan utzi dezan. Ondoren, MongoCursor bat erabiliz, Tipo guztiak ArrayList<String> baten gordeko ditugu eta Comparator bat erabiliz alfabetikoki ordenatuko ditugu.

```
@Override
public List<String> findTypes() {
    List<String> types = new ArrayList<>();
    MongoCursor<Document> cursor = typeCollection.find().projection(excludeId()).iterator();
    try {
        while (cursor.hasNext()) {
            types.add(cursor.next().get("name", String.class));
        }
    } finally {
        cursor.close();
    }
    types.sort(new Comparator<String>() {
        @Override
        public int compare(String s1, String s2) {
            return s1.compareToIgnoreCase(s2);
        }
    });
    return types;
}
```

API-aren erantzuna ondorengo forma izango du:

["Bug","Dark","Dragon","Electric","Fairy","Fighting","Fire","Flying","Ghost","Grass",
"Ground","Ice","Normal","Poison","Psychic","Rock","Steel","Water"]

Forma hau kontutan izanda, ASP ataletik irakurri egin beharko dugu metodo asinkrono baten bidez, baina hari nagusia blokeatuta ez gelditzeko, ConfigureAwait(false) gehitu beharko diogu.

```
static async Task<List<string>> GetTypeListAsync()
{
    List<string> TypeInfo = new List<string>();
    using (var client = new HttpClient())
    {
        client.BaseAddress = new Uri(BaseURL);
        client.DefaultRequestHeaders.Clear();
        client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
        HttpResponseMessage Res = await client.GetAsync("api/type").ConfigureAwait(false);
        if (Res.IsSuccessStatusCode)
        {
            var TypeResponse = Res.Content.ReadAsStringAsync().Result;
            TypeInfo = JsonConvert.DeserializeObject<List<string>>(TypeResponse);
        }
    }
    return TypeInfo;
}
```

Hala ere, metodo honek ez digu oraindik List<String> bat itzuliko, Task izeneko ‘tarea’ moduko bat baizik. Tarea honetatik Tipoen lista lortzeko, beste metodo bat sortu beharko dugu, Task-etik Result jasoko duena. Era berean, Pokemon bat aleatorioki hartzen duen metodoak eta Logina konprobatzen duen metodoak egitura bera erabiltzen dute.

```
public static List<string> GetTypeList()
{
    return GetTypeListAsync().Result;
}
```

Tipoekin lotuta, paginazioa eta filtraketa hasieran ez zuten ondo funtzionatzen. Tipo bat aukeratuta bazeneukan eta hurrengo orrira klikatzen bazenu, filtraketa galdu egiten zen. Hau aldatzeko asmoz, nabigazio barratik zein Tipo filtratzen ari garen gordeko dugu, eta paginazioaren botoiei Tipo hori gehituko diegu. Tiporik ez badago, 0 izango da.

```
var findtype = Request.QueryString["Type"];
```

```
<div class="pagination-container">
    @Html.PagedListPager(Model, page => Url.Action("PokemonList", new { Page = page, Type = findtype }))
</div>
```

Azkenik, Tipo bat aukeratzen duzunean, tipo hori filtraketa listatik kenduko dugu eta gorago jarriko dugu. Tipo horren tokian ‘Pokeball’ bat agertuko da, eta hau klikatzerakoan filtraketa kendu egingo da eta Pokemon guztiak agertuko dira berriz ere.



```

<div class="type-div white-background">
    @if (findtype != "0")
    {
        var imgName = findtype + ".png";
        <h2 style="text-align:center">Showing
            <span>
                
            </span> Type Pokemon
        </h2>
    }
    else
    {
        <h2 style="text-align:center">Showing All Pokemon</h2>
    }
    @foreach (var type in typeList)
    {
        if (type != findtype)
        {
            var imgName = type + ".png";
            <a title="@type" href="@Url.Action("PokemonList", "Pokemon", new { Page=1, Type = type })">
                
            </a>
        }
    }
    @if (findtype != "0")
    {
        <div class="pokeball-div">
            <a title="All Pokemon" href="@Url.Action("PokemonList", "Pokemon", new { Page=1, Type = "0" })">
                
            </a>
        </div>
    }
</div>

```

5.2.4.- Erabiltzaileak

Erabiltzaileak erabili ahal izateko, User klasean aldagai estatiko global bat sortu dugu, 'logged' izena duena. Aldagai hau boolean motakoa izango da eta Logina ahalbidetuko dugu. Hala ere, bezero bat baino gehiago badauzkagu eta bietako bat logeatzen bada, besteari ere logeatuta egongo balitz bezala aukerak agertuko zaizkio. Baita ere, nabigazio barra erabiliz aukera hauetara sartu ahalko zara konprobaziorik gabe. Hau konpontzeko sesioak erabili beharko genituzke.

5.3.- Erroreak eta egin gabekoak

Proiektua egiten eman dugun denbora guztian oztopo eta errore askorekin aurkitu gara, haietako gehienak stackoverflow.com foroari esker konpondu ditugu.

Proiektuaren hasieran, lehen aipatu dugun moduan, MongoDB-ko ObjectID eremuarekin arazoak izan genituen. Ez dugu errorearen jatorria aurkitu (gure ikaskide batzuek kode berdinarekin ez zeukaten errorerik) , beraz, gauzak errazteko, gure _id propioa sortu dugu, Integer motakoa. Hala ere, Tipoen kolekzioan ObjectID erabili dugu, baina kanpoan utzi egiten dugunez ez du errorerik ematen.

Beste alde batetik, Pokemon aleatorioak hartzeko metodoan hiru Pokemon hartzen ditugu banan banan. Gerta daiteke bi edota hiru Pokemon errepikatzea pantaila nagusian, baina ez dugu errore moduan aintzat hartzen.

Hala ere, proiektua orokorrean era natural baten garatu dela esango genuke, ez baitugu oztopo oso handiekin topo egin.

Denbora faltagatik edo konplexutasunagaitik, azkenean hainbat gauza proiektutik kanpo utzi behar izan ditugu:

- Update:
 - Pokemon-ak eguneratzeko aukerarik ez edukitzea erabaki dugu.
- Insert 'Polita':
 - Pokemon berri bat sortzeko formularioan, Tipoak sartzeko orduan, <select> aukeraketa listak erabili beharrean, CheckBox eta Irudi batzuekin egiteko asmoa geneukan. Hala ere, ez genekien nola egin PokemonController-etik zein CheckBox aukeratuta dauden jakiteko.



JavaScript-ekin bi tipo bakarrik aukeratzea egin genuen, baina ezin izan genuen gehiagorik egin.

5.4.- Teknologien balorazioa

Bukatzeko, proiektu honetarako erabili ditugun teknologia guztien balorazioa egingo dugu. Hurrengo irizpideak erabiliko ditugu teknologia desberdinak baloratzeko: Ulertzeko erraztasuna, Intuitibitatea eta erabilpena, Interneteko dokumentazioa eta Erroreen kudeaketa.

5.4.1.- MongoDB

- Ulertzeko Erraztasuna: **6**
 - Orain arte, guk erabili ditugun datu-base guztiak SQL datubaseak izan dira. Proiektuarekin hasi baino hilabete lehenago hasi ginen MongoDB erabiltzen. NoSQL egitura erraza dirudi, baina erabilpenean ez da horrela izango.
- Intuitibitatea eta erabilpena: **5**
 - Mongo Shell bertsioa ez zaigu oso erabilgarria iruditu. Aldiz, Mongo Compass-a erabili dugu, askoz sinpleagoa delako. Eragiketa desberdinak egiteko zailtasunak izan ditugu, eta komando eta hitz berezien erabilpena oso torpea dela uste dugu.
- Interneteko dokumentazioa: **3**
 - MongoDB buruzko informazio asko aurki dezakegu interneten, baina informazioa oso anbigua delakoan gaude. MongoDB bertsio desberdinetan eragiketak egiteko era aldatu egiten da. Horrez gain, Javarako driverreko tutorialetan eragiketak nola egin behar diren ez da oso argi gelditzen.
- Erroreen kudeaketa: **2**
 - Errore desberdinen aurrean ez dugu ezer egiteko aukerarik izan. Kode berbera ekipo desberdinetan era desberdinetan exekutatzen dela iruditu zaigu, eta batzutan berbera dirudien agindua guztiz desberdina da. ObjectId-ari buruz ez genuen ezer aurkitu.
- Azken nota: **4**
 - Orokorrean, MongoDB erabilgarria izan daitekela uste dugu, baina oso torpea iruditzen zaigu. Gure iritziz, SQL komandoekin konparatuta, oso zaila da.

5.4.2.- Java RESTful API

- Ulertzeko Erraztasuna: **5**
 - Hasieran, RESTful API bat zer zen ez genekien, eta kontzeptua ulertzeko buelta batzuk eman behar izan genion. Behin ulertuta, nahiko sinplea dela iruditu zaigu.
- Intuitibitatea eta erabilpena: **8**
 - Proiektu honetarako, Netbeans erabiltzeak abantaila asko dakartza. Metodo guztiak banatzeko errazak dira tutoriala ondo jarraituz gero, eta metodo horien implementazioa era bisual baten ikus daiteke Swagger-ri esker.
- Interneteko dokumentazioa: **9**
 - Nahiz eta interneten informazio asko ez aurkitu, begiratu ditugun tutorialak oso onak eta egokiak direla uste dugu. Horrez gain, tutorialetakoa adibideak erabili ditugu gure proiektuaren oinarri bezala, eta asko lagundu digu.
- Erroreen kudeaketa: **8**
 - Atal honekin ez ditugu ia errorerik izan. Beharbada, Mongo Driverraren zailtasunak zertxobait konplikatu egin du, baina horretaz aparte, ondo.
- Azken nota: **7.5**
 - RESTful API baten erabilpena ikasi eta ulertu dugu, eta gure etorkizunerako oso teknologia erabilgarria izan daitekela uste dugu.

5.4.3.- ASP.NET Web Aplikazioa

- Ulertzeko Erraztasuna: **9**
 - MVC estruktura dagoeneko ezagutzen genuen. Behin egitura honen funtzionamendua ulertuta, oso erraz hasi daiteke lanarekin.
- Intuitibitatea eta erabilpena: **8**
 - Visual Studio oso tresna egokia eta betea dela uste dugu. Plugin desberdinak instalatu ditzakegu funtzionalitate desberdinak gehitzeko, adibidez. Bestalde, C# lengoaia ez da oso konplikatua, Javaren oso antzekoa dela iruditu zaigu. Hala ere, metodo asinkronoen erabilpena zaila izan da, baina ondo irten zaigu.
- Interneteko dokumentazioa: **7**
 - Interneten funtzionalitate orokorreki buruzko dokumentazio ugari dago, baina arazo espezifikoentzat irtenbideak aurkitzea zaila izan ohi da.

- Erroreen kudeaketa: **9**
 - Nahiz eta errore ugari izan ditugun, Visual Studio-k erroreei buruzko informazio asko ematen du, eta horrela, hauek konpontzea asko errazten da.
- Azken nota: **8.25**
 - ASP web aplikazioak egiteko tresna oso egokia dela uste dugu, eta aukera asko eskaintzen ditu erraztasun maila handi batekin.