POLITECHNIKA GDAŃSKA

Internet Services Architectures

Laboratory 1

Spring Data & Spring MVC

During second laboratory students should get familiar with basic components of Spring Data (SQL database access with repositories) as well as Spring MVC (REST services implementation). Project should be based on the code developed during previous laboratories.

The following tasks must be completed:

1. All business classes should be enhanced with appropriate JPA annotations. Name of the tables should be in plural form. Tables and columns names should be formatted using `snake_case`. All relationships should be bidirectional. Fetching category should **not** fetch elements by default. (1 point).
2. Implementation of DTO for creating, reading,updating as well as reading all for each business class. DTO for deleting is unnecessary. DTO for creating or updating need to contain only those fields which can be set. For example: primary keys cannot be changed after creation, auto generated primary keys can not be set while creating, element category can not be set or changed. DTO for read all operations should not contain all fields but only identifiers (eventually some additional name or description if identifier is generated key instead of business key). (1 point).
3. Single in-memory storage should be removed and replaced with H2 in-memory database configuration. All repositories should be replaced with JPA repository interfaces. This will probably required changes in the services implementation as a contract with repository will change but should not require any changes in code using services. (2 point).
4. Implementation of rest controllers as `@RestController` for each business class. Controllers should utilize services for business operations and translate between business entities and DTO objects. Each controller must allow for all CRUD operations (create, read, update, delete) and read all operation. The resource addresses of the REST services must be well formed and hierarchical (example below). (3 point).

Attention: It is required to show that HTTP requests are working as assumed. The task does not required building view layer. It is enough to use HTTP request testing tool, for example HTTP Client delivered with Intelij IDEA.

Example data model – RPG game heroes representation (access modifiers, accessors, constructors and other methods omitted for the sake of simplification):

```
class Profession {          class Character {

    String name;                String name;

    double moveSpeed;           Profession profession;

    int baseArmor;              int level;

}                           }
```

Example requests and responses:

Request	Response body	Response code

`GET /api/professions`	`{`
	`  "professions": [`
	`    "bard",`
	`    "warrior",`
	`    "cleric",`
	`    "rogue"`
	`  ]`
	`}`
		200

`GET /api/professions/bard`	`{`
	`  "name": "bard",`
	`  "moveSpeed": 19.3,`
	`  "baseArmor": 10`
	`}`
		200

`GET /api/professions/fighter`		404

`GET /api/professions/bard/characters`	`{`
	`  "characters": [`
	`    "calvian",`
	`    "eloise"`
	`  ]`
	`}`
		200

`GET /api/professions/bard/characters/calvian`	`{`
	`  "name": "calvian",`
	`  "profession": "bard",`
	`  "level": 12`
	`}`
		200

```
GET /api/professions
```

```
{
  "professions": [
    {
      "id": 1,
      "name": "bard"
    },
    {
      "id": 2,
      "name": "warrior"
    },
    {
      "id": 3,
      "name": "cleric"
    },
    {
      "id": 4,
      "name": "rogue"
    }
  ]
}
```

```
200
```

```
GET /api/professions/1
```

```
{
  "id": 1,
  "name": "bard",
  "moveSpeed": 19.3,
  "baseArmor": 10
}
```

```
200
```

```
GET /api/professions/1/characters
```

```
{
  "characters": [
    {
      "id": 1,
      "name": "calvian"
    },
    {
      "id": 2,
      "name": "eloise"
    }
  ]
}
```

```
200
```

```
GET /api/professions/1/characters/1
```

```
{
  "id": 1,
  "name": "calvian",
  "profession": "bard",
  "level": 12
}
```

```
200
```

Request	Request body	Response code
`POST /api/professions`	`{ "name": "thief", "moveSpeed": 22.4, "baseArmor": 12 }`	201
`POST /api/professions`	`{ "name": "bard", "moveSpeed": 22.4, "baseArmor": 12 }`	400
`PUT /api/professions/bard`	`{ "moveSpeed": 22.4, "baseArmor": 12 }`	204
`PUT /api/professions/fighter`	`{ "moveSpeed": 22.4, "baseArmor": 12 }`	404
`DELETE /api/professions/bard`		204
`DELETE /api/professions/fighter`		404