HogWartSOS

Salvo Nicotra 2024-01-05

HogWartSOS

HogWartSOS is a school magic where wizardevs are sorted into langhouses.

Langhouse is a magic entity defined by

- id
- name
- motto

Each wizardevs is defined by

- id
- name

- year
- langhouse id (related to Langhouse)

Create SpringBoot project

In VS Code (assuming the Spring Boot initializer extension is available)

1. Command Palette (CTRL+ALT+P on Linux)
2. Spring Initialzir: Create a maven project
3. Version 3.4.1
4. Java
5. GoupId: wiz.hogwartsos
6. Artifact: sortinghack
7. Jar
8. Java Version 17

Configure Database Connection

In application.properties (src/main/java/resources) add datasource properties

> [!NOTE]
>
> starting writing spring.datasource the autocompleter will show the possible options.

```
spring.datasource.url=jdbc:mysql://localhost:3306/exam
spring.datasource.username=user
spring.datasource.password=password
spring.jpa.hibernate.ddl-auto=update
```

Dry Run

Test application Open integrated terminal (or system terminal) `mvn spring-boot:run` and check http://localhost:8080/ (you should get a 404)

This step is useful to

- test database connection
- download jar from maven
- build cache for Java extensions suggestions

Create Models

As an optional (but recommended for readability) create a package models to contain our models, using New Java Package feature (right click con artifact name)

Langhouse

Inside models, add a new Java class named langhouse.java, the provided skeleton should have the correct package and the class name

Define an entity

- Add @Entity annotation to the class
- (optional) add @Table annotation to map the name of the table, e.g. @Table(name=langhouses)

  [!NOTE]

  Both annotation will add the correspondent object in jakarta.persistence

Add attributes

- Add the id attribute, annotated with @Id and @GeneratedValue(strategy = GenerationType.IDENTITY) (to leverage database auto increment) and initialize it to 0L
- Add the name and motto attributes as strings

Generate methods

- Generate constructors using Right click, Source action -> Generate constructors
  - One empty
  - One with all attributes selected
- Generate getters and setters

WizarDev

Same as Langhouse expect for the langhouse_id

- Add Entity Annotation
- Add attributes
- Add a new attribute using
  - Object class of the related table i.e. `private langhouse langhouse_id;`
  - Annotate with @ManyToOne
  - Annotate with @JoinColumn(name = "langhouse_id")
- Optional: initialize attribute acts as default in database, i.e. `java private int year=1;`

Run to test if tables are created / mapped correctly

Hierarchies can be mapped in database using advanced features:

- Hibernate

Langhouses management

Create Views

Views are created using Thymeleaf and defined in src/main/resources/templates, it can be useful to create a directory to contains views related to a specific entity

We need two templates in a directory langhouses in src/main/resources/templates:

1. list all langhouses
2. create/edit a langhouse

List Langhouses

Create a file list.html

- Add a basic html structure, remember to include the thymeleaf namespace `xmlns:th="http://www.thymeleaf.org"`
- Add a table with a header and a row for each langhouse
  - Add a tr and th tag for each attribute i.e. id, name, motto
  - Add a th tag for actions (edit, delete)
  - Close the header row
  - Add a loop tr tag for each langhouse using th:each attribute

        * Add a td tag for each attribute i.e. id, name, motto
        * Add a td tag for actions (edit, delete)
            · Use and th:href attibute to create a link to edit/delete a langhouse, i.e. `<a th:href="@{/langhouses/edit/{id}(id=${langhouse.id})}">Edit</a>`

- Outside the table add a link to create a new langhouse
    - Use and th:href attibute to create a link to create a new langhouse, i.e. `<a th:href="@{/langhouses/new}">New Langhouse</a>`

Optional: Add sorting and pagination capabilities, number of records…

Edit Langhouse

Create a file edit.html with same structure of list.html

- Add a form tag with
    - th:object attribute to bind the form to the langhouse object
    - th:action attribute to bind the form to the correct endpoint i.e. `th:action="@{/langhouses/}"`, we use the same endpoint for create and update
    - Add a label and input for each attribute i.e. id, name, motto
        * Use th:field attribute to bind the input to the correct attribute i.e. `th:field="*{name}"`
        * Use th:disable attribute to disable the input for the id if id = 0 or set to th:readonly if id != 0
    - Add a submit button

Create JPA Repository

- Create a new package repositories to contain our repositories
- Create a new interface langhouseRepository extending JpaRepository<langhouse, Long>
    - At this stage we don't need to add any method, JpaRepository will provide all the basic CRUD operations

Create Controller

- Create a new package controllers to contain our controllers
- Create a new class langhouseController
- Annotate with @Controller
- Add a private attribute to store the repository i.e. `private langhouseRepository repository;`

- Add a constructor using Source action to inject the repository i.e. `public langhouseController(langhouseRepository repository) { this.repository = repository; }`

List Langhouses

- Add a method to list all langhouses – Digit @GetMapping this will autogenerate the code template to handle the request, but needs to be adjusted — Define the mapping endpoint i.e. `@GetMapping("/langhouses")` — Define the function name i.e. `public String read(Model model)` — Change the params to Model model — Add a model attribute to store the list of langhouses i.e. `model.addAttribute("langhouses", repository.findAll());` — Return the proper view i.e. `return "langhouses/list";`

Time for a test

- Run application
- Go to endpoint http://localhost:8080/langhouses

Create a new Langhouse

- Add a method to create a new langhouse
  - Digit @GetMapping this will autogenerate the code template to handle the request, but needs to be adjusted
  - Define the mapping endpoint i.e. `@GetMapping("/langhouses/new")`
  - Define the function name and correct parameters i.e. `public String create(Model model)`
  - Pass a an empty langhouse object to the model i.e. `model.addAttribute("langhouse", new langhouse());`
  - Return the proper view i.e. `return "langhouses/edit";`
- Add a method to save a new langhouse
  - Digit @PostMapping this will autogenerate the code template to handle the request, but needs to be adjusted
  - Define the mapping endpoint i.e. `@PostMapping("/langhouses/")`
  - define the function name and correct parameters i.e. `public String cr(@ModelAttribute langhouse house)`
  - @ModelAttribute annotation to bind the form to the langhouse object
  - Save the langhouse object i.e. `repository.save(house);`
  - Redirect to the list of langhouses i.e. `return "redirect:/langhouses";`

Edit a Langhouse

- Add a method to edit a langhouse
  - Digit @GetMapping this will autogenerate the code template to handle the request, but needs to be adjusted
  - Define the mapping endpoint i.e. `@GetMapping("/langhouses/{id}/edit")`
  - Define the function name and correct parameters i.e. `public String edit(@PathVariable Long id, Model model)`
  - Retrieve the langhouse object i.e. "'model.addAttribute("langhouse", repository.getReferenceById(id));
  - Return the proper view i.e. `return "langhouses/edit";`

Delete a Langhouse

Are we sure we can do it ?

WizarDevs management

Add a directory wizardevs in src/main/resources/templates

Add/Edit WizarDevs

Create a file edit.html in thymeaf format (see langhouses/edit.html)

- Add a form tag with
  - th:object attribute to bind the form to the wizardev object
  - th:action attribute to bind the form to the correct endpoint i.e. `th:action="@{/wizardevs/}"`, we use the same endpoint for create and update
  - Add a label and input for each attribute i.e. id, name, year
  - Add as select for the langhouse attribute
    - Use th:field attribute to bind the input to the correct attribute i.e. `th:field="*{langhouse_id}"`
    - Use th:each attribute in option to bind the select to the list of langhouses i.e. `th:options="@{langhouse.id} langhouse.name for langhouse in ${langhouses}"`
    - Note that langhouses is a new model attribute that needs to be passed to the view

Create JPA Repository for WizarDevs

Copy langhouseRepository to wizardevRepository and adjust the class name and the model

Create Controller for creating/editing WizarDevs

Copy langhouseController to wizardevController - adjust the class name - adjust the repository (keep the other import and the constructor) - comment all the methods - add methods to create/edit/save wizardevs - in the new/edit method add a model attribute to pass the list of langhouses i.e. `model.addAttribute("langhouses", lRepository.findAll());` - do the same for edit - and to save

Time for a test

Create View/Controller to list WizarDevs

- Create a file list.html in thymeleaf format (see langhouses/list.html)
  - Change fields to show wizardevs attributes
  - For the langhouse attribute use th:text to show the nested name of the langhouse i.e. `<td th:text="${wizardev.langhouse_id.name}"></td>`
- Create a new method in the controller to list all wizardevs

Test the application

Add a method to delete a WizarDev

In the controller add a new method to delete a wizardev, copy the edit method

- adjust the mapping and the function name
- get the model using getReferenceById
- delete the object using the repository

Add a method to delete a Langhouse

We cannot do the same for the langhouse controller, since we have a foreign key constraint in the wizardev table. We have two options:

1. Delete all the wizardevs related to the langhouse
2. Set the langhouse_id to null in all the wizardevs related to the langhouse

TODO

Improvements

Add the total number of wizardevs in the wizardevs list

Since we are not (yet) paginating we have the full list in the view.

Thymeleaf allows to use the size method to get the size of a list

In the list.html file add ${#lists.size(wizardevs)} in a html tag to show the total number of wizardevs

Filter by langhouse

Add a filter in the langhouse list

- Add a link in the langhouses/list.html file to filter by langhouse (same as edit/delete)
- Add a controller method to filter wizardevs by langhouse id
  - Use `findByLanghouseId` and define the method in the repository just adding the method signature
  - add the filtered langhouse to the model
  - return the list of wizardevs
- Add a optional section in wizadevs/list.html to show the filter
  - Use th:if to show the filter only if the list of langhouses is not empty
  - Use + to concatenate strings in thymeleaf i.e. `Filter by: + ${langhouse.name}`

Add a search bar per name in the wizardevs list

- In wizerdevs/list.html add a form with a text input and a submit button
- Add a new method in the controller to filter wizardevs by name
  - Use `findByNameContainingIgnoreCase` and define the method in the repository just adding the method signature
  - add the filtered wizardevs to the model as well as the search string
  - return the list of wizardevs

Filter by year

Create a filter by year in the wizardevs list

Fancy UI

Add some css to make the UI more appealing and some value test (powered by ChatGPT

```
INSERT INTO exam.langhouses (id, name, motto) VALUES
(1, 'Javafy', 'Code once, run anywhere.'),
(2, 'Pythorium', 'Elegant and cunning.'),
(3, 'Phavenclaw', 'Knowledge is power.'),
(4, 'Gofflepuff', 'Hard work and resilience.');

INSERT INTO exam.wizardevs (id, name, year, langhouse_id) VALUES
(1, 'Harry Devter', 1, 1),  -- Javafy (Gryffindor)
(2, 'Hermiona Codenger', 2, 1),  -- Javafy (Gryffindor)
(3, 'Ron Bytezley', 1, 1),  -- Javafy (Gryffindor)
(4, 'Draco Linqueful', 3, 2),  -- Pythorium (Slytherin)
(5, 'Luna Loopyloop', 2, 3),  -- Phavenclaw (Ravenclaw)
(6, 'Neville Debugbottom', 1, 1),  -- Javafy (Gryffindor)
(7, 'Ginny Compilson', 3, 1),  -- Javafy (Gryffindor)
(8, 'Cedric Variable', 4, 4),  -- Gofflepuff (Hufflepuff)
(9, 'Cho Exception', 3, 3),  -- Phavenclaw (Ravenclaw)
(10, 'Fred Moduley', 2, 1);  -- Javafy (Gryffindor)

INSERT INTO exam.wizardevs (id, name, year, langhouse_id) VALUES
(11, 'George Moduley', 3, 1),  -- Javafy
(12, 'Molly Moduley', 4, 1),   -- Javafy
(13, 'Arthur Moduley', 4, 1),  -- Javafy
(14, 'Bill Moduley', 5, 1),    -- Javafy
(15, 'Charlie Moduley', 5, 1), -- Javafy
(16, 'Percy Moduley', 3, 1);   -- Javafy
```