Petistaan

Pet Management System

About Internship

This internship aims to simulate real-world software development experience by upgrading an existing console-based Java Spring Boot application into a RESTful API-based backend system. This opportunity is ideal for learners looking to strengthen their backend development skills with hands-on experience in Spring Boot, MySQL, REST APIs, testing, localization, and documentation.

This is a completely FREE internship – no payment required or provided. It is a self-paced learning opportunity designed solely for skill enhancement.

You will not receive guided training; instead, you are encouraged to refer to our video tutorials on the Abhishek Verma YouTube Channel - https://www.youtube.com/@abhishekvermaa10 as needed. You can check the complete roadmap at https://abhishekvermaa10.github.io to see which topic you are missing. Use Google and AI tools only as a last resort – the goal is to develop self-reliance and problem-solving skills, so that when you will start working in real world project in company you do not face it difficult.

Objective

Upgrade the existing Petistaan console-based Spring Boot application into a REST API-based backend system adhering to Level 2 of the Richardson Maturity Model. Before initiating the enhancement, ensure the existing setup is properly cloned, configured, and verified.

Initial Setup Instructions (Pre-Requisite)

Announce your project start!

To stay motivated throughout this internship, especially with the variety of tasks involved, we recommend making a public commitment. A fantastic way to do this is by announcing your journey on LinkedIn.

Create a short post sharing that you have officially started this internship experience. This public declaration not only keeps you accountable but also encourages you to continue even when you hit roadblocks.

Don't forget to tag me https://www.linkedin.com/in/abhishekvermaa10 in your post and include a link to this video - https://youtu.be/MYpHl71hdU4. This will also help others who are interested to join and begin their journey alongside you.

Clone the Repository

- URL: https://github.com/abhishekvermaa10/Spring-Data
- Clone the repository.
- Delete all other demos except Petistaan.

Database Setup

- Ensure MySQL Community Server is installed along with a GUI tool like DBeaver, MySQL Workbench, etc.
- Create a fresh schema:
  ```
  CREATE DATABASE petistaan;
  ```

Recommended: Delete any old schema with the same name to avoid confusion.

Application Configuration

- Import the Petistaan project into an IDE (e.g., Spring Tool Suite, IntelliJ IDEA).
- The project requires Java 21.
- Make sure the following tools and plugins are supported:
  - Lombok
  - JUnit
  - JaCoCo
  - Cucumber
  - UML diagram generator
- Set environment variables for database credentials:
  - MYSQL_USERNAME
  - MYSQL_PASSWORD
- Ensure your application.yml contains:
  ```
  spring.datasource.url=jdbc:mysql://localhost:3306/petistaan
  ```
- Watch the Petistaan video from the Spring Data JPA playlist for overview.

Run the Application

- Run the app.
- Hibernate should auto-generate required tables.

Insert Sample Data

- Navigate to src/main/resources.
- Locate data.sql script.
- Run the script manually to populate all the 4 tables one by one.
- Execute below query to verify 72 rows have been created:
  ```
  SELECT ot.id, ot.first_name, ot.last_name, ot.gender, ot.city,
  ot.state, ot.mobile_number, ot.email_id, ot.pet_id,
  pt.name, pt.gender, pt.type, dpt.date_of_birth,
  wpt.place_of_birth
  FROM owner_table ot
  JOIN pet_table pt ON pt.id = ot.pet_id
  ```

```
LEFT JOIN domestic_pet_table dpt ON pt.id = dpt.id
LEFT JOIN wild_pet_table wpt ON pt.id = wpt.id;
```

Functional Verification

- Evaluate all 8 console options.
- Ensure workflow is functional and data is being processed correctly.
- Verify logs are getting generated at logs/Petistaan.log.
- Execute unit test cases and verify all are passing.
- Build JAR file locally.
- Only proceed to enhancement once existing features are verified and JAR file is getting created.

Troubleshooting Tips

- Ensure there are no compilation errors. Since project makes use of Lombok, so if your IDE do not have it then errors will appear.
- Force update to download dependencies. Good internet connection is required for this step.
- Ensure a clean schema before executing app to avoid confusion in data.
- MySQL default port is 3306 – update application.yml if using a different port.
- If errors persist, delete the project and re-clone it from GitHub.

Note: Project has been uploaded only after due verification.

Enhancement Scope

Upgrade Spring Boot

- Update to the latest stable Spring Boot version.
- Fix resulting compilation errors.
- Evaluate application if needed and Rebuild the JAR file.
- You are expected to add required dependencies at every step own your own. You can refer video on our YouTube channel whenever you feel stuck.

Convert to RESTful APIs

- Introduce OwnerController class with base endpoint as /owners.
  - Options 1,2,3,4,5,8 should be converted to APIs in OwnerController
  - Changes needed for option1: API should accept OwnerDTO and return ownerId as JSON object and not simple integer. You may need to create OwnerIDDTO class and update return type in service layer also.
  - Changes needed for option2: API should accept ownerId and return OwnerDTO if successful else throw OwnerNotFoundException.
  - Changes needed for option3: API should accept ownerId and UpdatePetDTO and return void if successful else throw OwnerNotFoundException. You may need to create UpdatePetDTO class.
  - Changes needed for option4: API should accept ownerId and return void if successful else throw OwnerNotFoundException.

  - Changes needed for option 5: API should always return list of ownerDTO. Empty list if no owner.
  - Changes needed for option 8: API should accept Pageable object and always return page of OwnerPetInfoDTO. Empty page if no data. You may need to create OwnerPetInfoDTO class and update return type and argument in service layer and repository layer also.
- Introduce PetController class with base endpoint as /pets.
  - Options 6,7 should be converted to APIs in PetController class.
  - Changes needed for option6: API should accept petId and return PetDTO if successful else throw PetNotFoundException.
  - Changes needed for option 7: API should return average age as JSON object and not simple Double. 0 by default. You may need to create AverageAgeDTO class and update return type in service layer also.
- Delete InputUtil class once all APIs are ready and clean up Demo class.
- Define endpoints with proper HTTP methods and paths.
- Ensure all APIs return ResponseEntity with appropriate status codes.
- Maintain consistency in response structure.
- Update the context path of application to /petistaan. So that URLs look like below:
  - `http://localhost:8080/petistaan/owners`
  - `http://localhost:8080/petistaan/pets`

Input Validation

- Validate request arguments and DTOs.
- Validate all DTOs which are used as an input in any API developed.
- Refer messages section to update error message in messages.properties file.

Centralized Exception Handling

- Create an ErrorDTO with fields: timestamp, message, status, error.
- Return meaningful responses on validation or exceptions.

Localization (l10n)

- Add field formattedBirthDate in DomesticPetDTO which is used for only when individual owner or pet is returned.
- Create messages_isoCode.properties files for your regional language.
- If its English, then make any other language you know or would like to know.
- This means that there should be in total support for 2 languages for all error messages.
- Ensure that you use only iso code in file name.
- Localize error messages and formatted dates using Accept-Language header.
- Feel free to use any tool to change the language.

Enable Global CORS

- Allow requests from any origin for frontend integration.

Springdoc OpenAPI Integration

- Generate documentation at /documentation and UI at /swagger.html.
- Provide: Title, Description, Contact Info and Detailed descriptions for each Controller, DTO, and endpoint.
- If selected, your demo will be shown in video, but we will hide/blur your personal email id to protect your privacy.

Actuator Integration

- Expose all actuator endpoints.
- Ensure /actuator/env, /actuator/health, /actuator/configprops are accessible.
- Add info.app.name and info.app.message in configuration.
- They will be helpful for spring boot admin server later.

Language API Integration (Transliteration)

- Use Google Input Tools Transliterate API to convert input value in each language.

API: `https://inputtools.google.com/request?itc={itcCode}&text={input}`

where,

input = URLEncoder.encode(input_text, StandardCharsets.UTF_8)

encode = isoCode + "-t-i0-und"

- Use it for transliterating below values in response of individual owner and pet API only – firstName, lastName, city, state, petName, petBirthPlace.

Sample Input: `https://inputtools.google.com/request?itc=hi-t-i0-und&text=Subscribe Abhishek Verma`
where,

isoCode = hi

itcCode =hi-t-i0-und

input = Subscribe Abhishek Verma

Sample JSON output:

```
[
   "SUCCESS",
   [
    [
     "Subscribe Abhishek Verma",
     [
```

```
      "सब्सक्राइब अभिषेक वर्मा"
    ],
    [],
     {
      "candidate_type": [
                          0
      ]
     }
    ]
  ]
]
```

Note: This API is free to use and hence we are preferring the same. You can suggest us if there is any better free alternative.

Spring Profiles

- Create profiles: dev, test, prod.
- dev:
  - Use dummy transliterate API.
  - Single log file
- test:
  - Used by test cases only.
- prod:
  - Use actual transliterate API.
  - Rolling log policy (Max 10MB per file OR new date)

Testing

- Unit Tests:
  - Write tests for all controller classes.
  - Update tests for Service and Repository layer if needed.
- Cucumber Tests:
  - Write Cucumber scenarios for all REST endpoints.

UML Diagram Update

- Update UML diagram to reflect all classes/interfaces/enums including testing related.

Spring Boot Admin

- Register the app as a Spring Boot Admin Client.
- Setup a Spring Boot Admin Server project on port 9090.
- Admin Server should be able to see – info, health, environment, configuration properties, logs, and trace requests.
- Configure email alerts for up/down status using Spring Boot Admin and Mail.

Announce your project completion!

Once you successfully complete this internship, we encourage you to share your overall experience on LinkedIn.

Write a short post highlighting what you learned, how the journey was, and how it helped you grow. This not only celebrates your achievement but also inspires others to take up the opportunity.

Make sure to tag me https://www.linkedin.com/in/abhishekvermaa10 in your post and include a link to this video - https://youtu.be/MYpHl71hdU4 so that others who come across it can also begin their own learning journey.

Deliverables

- You should be able to create a JAR file at the end and are expected to share updated Petistaan app with below functionalities:
  - Updated Spring Boot application with REST APIs.
  - Fully localized and internationalized application.
  - Profile-based third-party integration.
  - Swagger documentation.
  - Admin server project and alert configuration. (App should contain config to register itself on server running on port 9090, you do not need to share server, but you can make it locally to see if it is working fine or not)
  - Unit and Cucumber test suites.
  - UML diagram file.
  - Two LinkedIn posts – about start and finish.

Technology Stack

- Java 21
- Maven
- Spring Boot 3.x
- MySQL
- H2
- JUnit + Mockito
- Cucumber
- Springdoc OpenAPI
- Spring Boot Admin

messages.properties file

owner.not.found=Can't find owner with ownerId %s.

pet.not.found=Can't find pet with petId %s.

owner.id.positive=Owner id must be a positive number.

pet.id.positive=Pet id must be a positive number.

owner.first.name.required=First name of owner is mandatory.

owner.first.name.length=Maximum length of first name of owner can be {max}.

owner.last.name.required=Last name of owner is mandatory.

owner.last.name.length=Maximum length of last name of owner can be {max}.

owner.gender.required=Gender of owner is mandatory.

owner.city.required=City of owner is mandatory.

owner.city.length=Maximum length of city of owner can be {max}.

owner.state.required=State of owner is mandatory.

owner.state.length=Maximum length of state of owner can be {max}.

owner.mobile.number.required=Mobile number of owner is mandatory.

owner.mobile.number.length=Length of mobile number of owner needs to be {max}.

owner.email.required=Email id of owner is mandatory.

owner.email.invalid=Email id of owner is invalid.

owner.pet.required=Pet of owner is mandatory.

pet.name.required=Name of pet is mandatory.

pet.name.length=Maximum length of name of pet can be {max}.

pet.gender.required=Gender of pet is mandatory.

pet.type.required=Type of pet is mandatory.

pet.birth.date.required=Date of birth of pet is mandatory.

pet.birth.date.old=Date of birth of pet can be a past or present date only.

pet.birth.place.required=Place of birth of pet is mandatory.

pet.birth.place.length=Maximum length of place of birth of pet can be {max}.