/ JAVA DB SCHOOL
Repository

Data persistence in Java

- JDBC
- JdbcTemplate
- JPA (EntityManager)
- Spring Data JPA
  - CrudRepository
  - JPARepository

/ Spring Data JPA

Spring Data JPA (1)

- Makes it easy to easily implement JPA based repositories
- Features
  - Support for Querydsl predicates and thus type-safe JPA queries
  - Transparent auditing of domain class
  - Pagination support, dynamic query execution, ability to integrate custom data access code
  - Validation of @Query annotated queries at bootstrap time
  - Support for XML based entity mapping
  - @EnableJpaRepositories
- https://spring.io/projects/spring-data-jpa

Spring Data JPA (2)

- Starter package:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Spring Data JPA (3)

```
@Entity
public class Customer {

  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  private Long id;
  private String firstName;
  private String lastName;

  protected Customer() {}

  public Customer(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }
}
```

CrudRepository (1)

```
import java.util.List;

import org.springframework.data.repository.CrudRepository;

public interface CustomerRepository extends CrudRepository<Customer, Long> {
    Customer findById(long id);
}
```

CrudRepository (2)

- The generic parameters of the CrudRepository are type of entity and type of id
- By extending CrudRepository, CustomerRepository inherits several methods for working with Customer persistence
- You don’t need to write an implementation of the repository interface, Spring Data JPA creates an implementation when you run the application
- https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html

JPARepository (1)

- Extend CrudRepository and brings extra functionalities
- The JPA module supports defining a query manually as a String or having it being derived from the method name

- Example:
  List<Customer> findByFirstNameAndLastName(String firstName, String lastName);

  Generates a method for finding records by first name and last name

JPARepository (2)

- Supported keywords inside method names
  - Distinct
  - And, Or
  - Is, Equals
  - Between, LessThan, LessThanEqual, GreaterThan, GreaterThanEqual
  - After, Before
  - IsNull, IsNotNull
  - Like, NotLike
  - StartingWith, EndingWith, Containing
- https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods

@Query

- Can declare custom queries
- Defined in the repository

- JPQL
  - @Query("select c from Cutomer c where c.firstName like %?1")
  - List<User> findByFirstNameEndsWith(String firstName);

- Native query
  - @Query(value = "SELECT * FROM customer WHERE firstName = ?1", nativeQuery = true)
  - User findByFirstName(String firstName);

Pagination

- JPQL

```
@Query(value = "SELECT u FROM User u WHERE u.lastName = ?1")
Page<User> findByLastName(String lastName, Pageable pageable);
```

- Native
- We can enable pagination for native queries by declaring an additional attribute countQuery.
- This defines the SQL to execute to count the number of rows in the whole result:

```
@Query(value = "SELECT * FROM customer WHERE lastName = ?1",
   countQuery = "SELECT count(*) FROM customer WHERE lastName = ?1",
   nativeQuery = true)
 Page<User> findByLastname(String lastname, Pageable pageable);
```

Sorting

- Example

```
@Query("select u from User u where u.lastname like ?1%")
 List<User> findByAndSort(String lastName, Sort sort);
```

- Usage

```
repo.findByAndSort("lannister", Sort.by("firstname"));
repo.findByAndSort("stark", Sort.by("LENGTH(firstname)"));
```

/ Logging

Logging (1)

- By default Spring Boot uses Logback library

```
@RestController
public class LoggingController {

    Logger logger = LoggerFactory.getLogger(LoggingController.class);

    @RequestMapping("/")
    public String index() {
        logger.trace("A TRACE Message"); // debug, ingo, warn, error
        return “Hello world!!";
    }
}
```

Logging (2)

- Adding a different logger

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-log4j2</artifactId>
</dependency>
```

/ ResponseEntity

ResponseEntity

- Used when needing to control aspects of the response other than the body

- Create resource
  - Needs to also include an URI to the newly created resource

Create

```
@PostMapping("/")
public ResponseEntity<Customer> create(@RequestBody Customer customer)
    throws URISyntaxException {
    Customer createdCustomer = service.create(customer);
    if (createdCustomer == null) {
        return ResponseEntity.notFound().build();
    } else {
        URI uri = ServletUriComponentsBuilder.fromCurrentRequest()
          .path("/{id}").buildAndExpand(createdCustomer.getId()).toUri();
        return ResponseEntity.created(uri)
          .body(createdCustomer);
    }
}
```

Read

```
@GetMapping("/{id}")
public ResponseEntity<Customer> read(@PathVariable("id") Long id) {
    Student foundCustomer = service.read(id);
    if (foundCustomer == null) {
        return ResponseEntity.notFound().build();
    } else {
        return ResponseEntity.ok(foundCustomer);
    }
}
```

Update

```
@PutMapping("/{id}")
public ResponseEntity<Customer> update(@RequestBody Customer customer,
@PathVariable Long id) {
    Student updatedCustomer = service.update(id, customer);
    if (updatedCustomer == null) {
        return ResponseEntity.notFound().build();
    } else {
        return ResponseEntity.ok(updatedCustomer);
    }
}
```

Delete

```
@DeleteMapping("/{id}")
public ResponseEntity<Object> deleteCustomer(@PathVariable Long id) {
    service.delete(id);
    return ResponseEntity.noContent().build();
}
```

/ Cookies

Cookies

- HTTP cookies
  - Small blocks of data created by a web server while a user is browsing a website
  - Browsers receive and send back cookies without changing or altering them
  - Saved on the user’s computer
- Purpose
  - Store stateful information help websites track visits and activity (e.g. username, shopping cart)

Type of cookies (1)

- Session cookie
  - In-memory cookie, transient cookie, non-persistent cookie
  - Exists in temporary memory while the user navigates a website
  - Expire or are deleted when the user closes the browser
  - Do not have a set expiration date
- Persistent cookie
  - Tracking cookie
  - Expires at a specific date
  - For the cookie’s lifespan, the information is transmitted to the webserver every time the user visits the website

Type of cookies (2)

- Secure cookie
  - Can only be transmitted over an encrypted connection (HTTPS)
- Http-only cookie
  - Cannot be accessed by client API (JavaScript)

Reference: https://networkencyclopedia.com/http-cookie/

Spring cookies (1)

- Creating a cookie

```
ResponseCookie springCookie = ResponseCookie.from("user-id", “1")
    .httpOnly(true)
    .secure(true)
    .path("/")
    .maxAge(60)
    .domain("example.com")
    .build();
```

Spring cookies (2)

- Returning a cookie

```
return ResponseEntity
    .ok()
    .header(HttpHeaders.SET_COOKIE, springCookie.toString())
    .build();
```

Spring cookies (3)

- Reading a cookie

```
@GetMapping("/cookie")
public String readCookie(
    @CookieValue(name = "user-id", defaultValue = "default") String userId) {
    return userId;
}
```

Spring cookies (4)

- Deleting a cookie

```
ResponseCookie deleteSpringCookie = ResponseCookie
        .from("user-id", null)
        .maxAge(0)
        .build();

 ResponseEntity
        .ok()
        .header(HttpHeaders.SET_COOKIE, deleteSpringCookie.toString())
        .build();
```

/ Lombok

Project Lombok

- https://projectlombok.org/
- Java library that helps with boilerplate code
- Features
  - @Getter @Setter @NoArgsConstructor
  - @ToString
  - @EqualsAndHashCode
  - @SneakyThrows
  - @Log @Slf4j @CommonsLog @Log4j @Log4j2 @XSlf4j

/ Practice, practice, practice

Practice

- Implement JPARepository for the project

- Implement register and login methods and use cookies to “remember” which use is logged in

/ Q&A


OBILE/ACADEMY