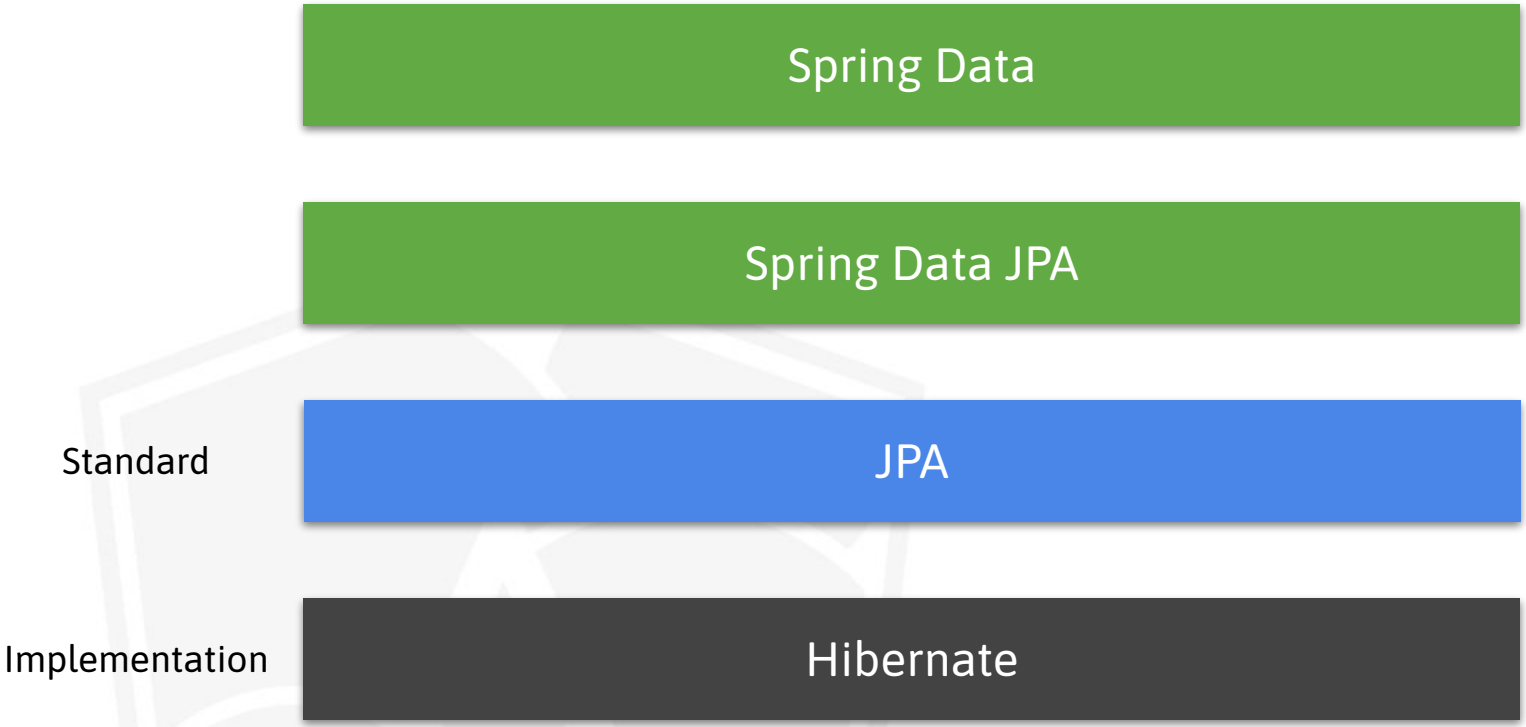
ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Spring Workshop

4 - Data


Spring Data
Spring Data JPA
Standard
JPA
Implementation
Hibernate

@Entity and other Annotations

- Associates a database table to a class
- Classname can be different from table name
- Be aware of Naming Strategies
  - Implicit
  - Physical
  - Spring

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Entity

- @Entity
- @Table
- @Id
- @GeneratedValue
  - IDENTITY: auto increment MySQL, identity in SQL Server
  - SEQUENCE: Oracle
  - (TABLE)
- @Transient
- @Column
- BeanValidation

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Repository

- Interface implemented by Spring Data
- Manages database specific actions
- Common Subtypes
  - (List)CrudRepository
  - (List)PagingAndSortingRepository
  - JpaRepository
- Customisable via find* methods

Customer as Entity

```
public class Customer {

  private Long id;

  private String firstname;
  private String lastname;

  private Boolean signedGdpr;
}
```

Customer as Entity

```
@Entity
public class Customer {

  private Long id;

  private String firstname;
  private String lastname;

  private Boolean signedGdpr;
}
```

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Customer as Entity

```
@Entity
public class Customer {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private String firstname;
  private String lastname;

  private Boolean signedGdpr;
}
```

CustomersRepository

```
package com.softarc.eternal.data;

import com.softarc.eternal.domain.Customer;
import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CustomersRepository extends JpaRepository {}
```

CustomersRepository

```
package com.softarc.eternal.data;

import com.softarc.eternal.domain.Customer;
import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CustomersRepository extends JpaRepository<Customer, Long> {}
```

CustomersRepository

```
package com.softarc.eternal.data;

import com.softarc.eternal.domain.Customer;
import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CustomersRepository extends JpaRepository<Customer, Long> {
  List<Customer> findByFirstnameOrLastnameOrderByLastname(
    String firstname,
    String lastname
  );
}
```

Database Connection in application.yml for H2

```yaml
spring:
  datasource:
    url: jdbc:h2:./dev-db
    driver-class-name: org.h2.Driver
    username: sa
    password: 'Sag ich dir nicht :)'
```

- H2 is embedded/file/server
- Possibility to use multiple databases
- Make use of profiles
- Credentials
  - plaintext
  - encrypted
  - OS env

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Further useful configuration properties

```yaml
spring:
  jpa:
    hibernate:
      ddl-auto: create-drop

logging:
  level:
    org.hibernate.orm.jdbc.bind: trace

spring:
  jpa:
    show-sql: true
```

Impact on Testing

- No impact on Unit Tests (no DI)
- Override the datasource properties via @SpringBootTest
  - `jdbc:h2:mem:holidays-controller`
  - in-memory instance
- Mock the Repository via @MockedBean in Integration Tests


private void executeLoad(long timeout, int usersCount) {
showDebugInfo(timeout);
Load.setPages(URL, parsingTimeout);
Load.setTimeout(timeout);
List<Load> threads = new ArrayList<>();
for (int i = 0; i < usersCount; i++) {
threads.add(new Load(this.URL));
}
logger.info(s: usersCount +
for (Load thread : threads)
thread.start();
}
logger.info(s: "All threads are started");
progressInfo(timeout);
System.out.print(".....DONE\nProcessing with data...\n");
}
private void executeAvailability(long timeout,
}
Demo

Lab Time

@DataJpaTest

- In-Memory Database is normally used
- Transactional with Rollback
- SQL commands and queries are printed

@DataJpaTest

```
@DataJpaTest(
  properties = {
    "spring.datasource.url=jdbc:h2:mem:test",
    "spring.jpa.hibernate.ddl-auto=create-drop",
  }
)
public class CustomersControllerIntegrationTest {

  @Autowired
  CustomersRepository repository;

  @Test
  void testRepo() {
    var franz = CustomerMother.franz().build();
    repository.save(franz);
    Customer customer = repository.findAll().get(0);
    assertThat(customer)
      .usingRecursiveComparison()
      .ignoringFields("id")
      .isEqualTo(franz);
  }
}
```

Flyway

- Versioned Migration Tool
- Uses SQL
- Files in resources with convention V{n}__{name}.sql
- Liquibase (database agnostic)
- Careful with line ending (sql files are hashed)
- Rollback possible

Flyway / MySql Setup

1. Setup MySQL via Docker or in another way
2. Register datasource.url
   a. `jdbc:mysql://localhost:3306/eternal`
3. Add dependencies
   a.
```
implementation 'mysql:mysql-connector-java'
implementation 'org.flywaydb:flyway-core'
implementation 'org.flywaydb:flyway-mysql'
```
4. Create init schema in src/main/resources/db/migration/V1__init.sql
5. Restart Spring Boot

Relations

- Used quite often
  - @ManyToOne
  - @OneToMay
- Rarely
  - @OneToOne
  - @ManyToMany

Relations

```
@Entity
public class Country {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private String name;

  @OneToMany(mappedBy = "country")
  private List<Customer> customers;
}
```

Relations

```
@Entity
public class Customer {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private String firstname;
  private String lastname;

  private Boolean signedGdpr;

  @ManyToOne
  @JoinColumn(name = "country_id") // optional
  private Country country;
}
```

Advanced Use Cases

Agenda

- ManyToMany
- JPQL, Joins & EntityGraph
- Projection
- Query by Example
- Query by Specification
- @Transactional
- Async
- Versioning

ManyToMany (Owner)

```
public class Holiday {
  @ManyToMany
  @JoinTable(
    name = "holiday_trip",
    joinColumns = @JoinColumn(name = "holiday_id", referencedColumnName = "id"),
    inverseJoinColumns = @JoinColumn(
      name = "guide_id",
      referencedColumnName = "id"
    )
  )
  public List<Guide> guides;
}
```

ManyToMany (Non-Owner)

```
public class Guide {

  @ManyToMany(mappedBy = "guides")

  private List<Holiday> holidays;

}
```

JPQL

```
public interface HolidaysRepository extends CrudRepository<Holiday, Long> {

  @Query(

    "select h from Holiday h inner join h.trips t where t.fromDate > current date"

  )

  List<Holiday> findUpcomingHolidays();

}
```

EntityGraph

```
public interface HolidaysRepository extends CrudRepository<Holiday, Long> {

  @Query(

    "select h from Holiday h inner join h.trips t where t.fromDate > current date"

  )

  @EntityGraph(attributePaths = { "trips.guide" })

  List<Holiday> findUpcomingHolidays();

}
```

EntityGraph

```
public interface HolidaysRepository extends CrudRepository<Holiday, Long> {

 @Query(
   """
     select h from Holiday h inner join h.trips t
     where t.fromDate > current date
       and t.guide.lastname like :guideName"""
 )
 @EntityGraph(attributePaths = { "trips.guide" })
 List<Holiday> findUpcomingHolidays(String guideName);
}
```

Simple Projections

```
public interface HolidayTeaser {
  public Long getId();
  public String getName();
  public String getCoverPath();
}

public interface HolidaysRepository extends CrudRepository<Holiday, Long> {
  @Query("select h from Holiday h")
  List<HolidayTeaser> findTeasers();
}
```

Advanced Projections

```
public record HolidayWithGuide(Long id, String name, String guide) {}

public interface HolidaysRepository extends CrudRepository<Holiday, Long> {
  @Query(
  """
    select new com.softarc.eternal.data.HolidayWithGuide(h.id, h.name, concat(g.firstname, ' ', g.lastname))
    from Holiday h inner join h.trips t inner join t.guide g"""
  )
  List<HolidayWithGuide> findHolidayGuides();
}
```

Query by Example

```
@GetMapping("by-example")
public List<String> byExample() {
  var holiday = new Holiday();
  holiday.setCoverPath("jpg");
  var exampleMatcher = ExampleMatcher
    .matching()
    .withMatcher("coverPath", matcher -> matcher.endsWith());

  var example = Example.of(holiday, exampleMatcher);

  return repository
    .findBy(example, q -> q.stream())
    .map(Holiday::getName)
    .toList();
}
```

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Query by Specification

```
public class HolidaySpecs {
  public static Specification<Holiday> isJpgCover() {
    return (
      (root, query, builder) -> builder.like(root.get("coverPath"), "%jpg")
    );
  }
}

public List<String> bySpecification() {
  return repository
    .findAll(HolidaySpecs.isJpgCover())
    .stream()
    .map(Holiday::getName)
    .toList();
}
```

@Transactional

- Uses internally a TransactionManager
- All changes to Entities will eventually be persisted
  - No Repository::save required
- Supports Nesting
  - By default a single transaction
  - Can be configured to be separated
- Never pass around Entity instances!

@Version

- Is incremented on every update
- Checks on every update if database has same version as entity
- Different locking modes possible
  - Optimistic: default
  - Pessimistic: Adds lock in Database

JPQL Alternatives

- JOOQ
- Querydsl
- JDBC Template


private void executeLoad(long timeout, int usersCount) {
showDebugInfo(timeout);
Load.setPages(URL, parsingTimeout);
Load.setTimeout(timeout);
List<Load> threads = new ArrayList<>();
for (int i = 0; i < usersCount; i++) {
threads.add(new Load(this.URL));
}
logger.info(s: usersCount +
for (Load thread : threads)
thread.start();
}
logger.info(s: "All threads are started");
progressInfo(timeout);
System.out.print(".....DONE\nProcessing with data...\n");
}
private void executeAvailability(long timeout,
}
Demo

Lab Time